Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

В.В. Коваленко, Е.В. Гайдукова, Н.В. Викторова

ПРАКТИКУМ ПО ДИСЦИПЛИНЕ «МОДЕЛИРОВАНИЕ ГИДРОЛОГИЧЕСКИХ ПРОЦЕССОВ. ЧАСТЬ III. ЧАСТИЧНО ИНФИНИТНОЕ МОДЕЛИРОВАНИЕ»

(на базе языка C++ Builder)

Допущено Учебно-методическим объединением по образованию в области гидрометеорологии в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности Гидрология

РГГМУ

Санкт-Петербург
2013

УДК 556.072(076.5)
ББК 26.22
К56

Коваленко В. В., Гайдукова Е. В., Викторова Н. В. Практикум по дисциплине «Моделирование гидрологических процессов. Часть III. Частично инфинитное моделирование» (на базе языка C++ Builder). – СПб.: изд. РГГМУ, 2013. – 102 с.

ISBN 978-5-86813-338-1

Рецензент: д-р физ.-мат. наук С. А. Кондратьев (зам. директора Института озероведения РАН).

Рассматриваются примеры частично инфинитного моделирования развивающихся гидрологических объектов, находящихся в условиях неустойчивости (или имеющих возможность попасть в такие условия при изменении климата). Решение задач проводится на базе визуального и событийного программирования с использованием языка C++ Builder, основы которого также даны в пособии. В данном пособии часть III продублировано решение некоторых задач из первых двух частей I и II в среде Borland C++ Builder.

Пособие предназначено студентам и магистрантам высших учебных заведений, обучающихся по направлению «Прикладная гидрометеорология» (в основном – специализирующихся по гидрологии), а также специалистам-гидрологам.

Kovalenko V. V., *Gaidukova E. V., Victorova N. V*. Practical work at the discipline «Modeling of hydrological processes. Part III. Partially infinity modeling» (on the basis of language C++ Builder). – St. Petersburg, RSHU Publishers, 2013. – 102 pp.

Reviewers: the doctor of physical and mathematical sciences S. A. Kondratyev (the deputy director of Institute of lake study the Russian Academy of Sciences).

In the manual the examples of partially infinity modeling of developing hydrological objects taking place in conditions of instability (or having an opportunity to get in such conditions at change of a climate) are considered. The decision of tasks will be carried out on the basis of visual and of event programming with use of language C ++ Builder, which bases also are given in the manual. In this tutorial, part III duplicated solution of some problems in the first two parts I and II among the Borland C + + Builder.

The manual is intended by the students and magisters of higher educational institutions training on a direction « Applied hydrometeorology (in basic – specializing on a hydrology), and also experts–hydrologists.

ISBN 978-5-86813-338-1

Введение

В настоящем издании Практикума часть III рассматриваются примеры частично инфинитного моделирования развивающихся гидрологических объектов, находящихся в условиях неустойчивости (или имеющих возможность попасть в такие условия при изменении климата). В качестве базового языка визуального и событийного программирования использован C++ Builder, который является естественным расширением языка C++, основы которого изложены в первых двух частях Практикума: часть I [7], часть II [8].

В связи с использованием разнообразных компонентов (особенно визуализирующих решения), логика части III отличается от таковой, использованной в предыдущих частях (ручная реализация – элементы языка – программная реализация). Гидрологические примеры сопутствуют изложению элементов C++ Builder.

Теоретические аспекты решаемых задач рассмотрены в действующем учебнике для вузов по «Моделированию гидрологических процессов» [5, 6] и в монографии [4]. Логика изложения элементов C++ Builder соответствует, в основном, книге Н. Б. Культина «Самоучитель C++ Builder» [9], но использовались и другие книги [1, 2, 3].

Авторы выражают благодарность участникам семинара по «Частично инфинитной гидрологии», на котором подробно обсуждались представленные в пособии примеры.

1. Интегрированная среда разработки (ИСР) C++ Builder

1.1. Конкретизация некоторых понятий объектно ориентированного программирования (ООП) в C++ Builder

1.1.1. Графический («мягкий») интерфейс

До сих пор мы имели дело с довольно «жестким» интерфейсом. В консольном приложении (на которое, в частности, ориентирован компилятор Dev C++) программа взаимодействует не непосредственно с Windows («окнами»), а с DOS. Дисковая операционная система представляет собой для программы, почти что в прямом смысле слова инфинитную реальность, частично раскрывающуюся в командной строке.

Более «мягким» является появившийся гораздо позже графический (визуальный) интерфейс, при котором не только все, что есть на экране монитора (окна, формы и т. д.) является объектами (в духе языка C++), но и сам человек – объект (хотя и «внешний»), см. рис. 1.1.

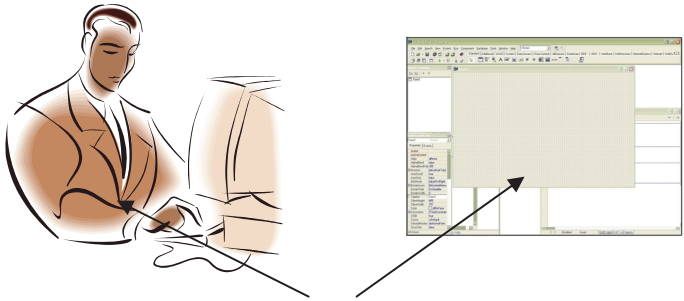


Рис. 1.1. Объекты в визуальном интерфейсе.

Внешний объект является и источником событий (хотя и не единственным). Все это делает ситуацию более финитной или, по крайней мере, менее инфинитной (за счет «выжигания» – рационализации программными средствами инфинитного окружения программы, т. е. используемой операционной системы).

Облегчение этого «выжигания» (т. е. технологии разработки графического интерфейса) было достигнуто с помощью двух шагов:

1. Появление пользовательских интерфейсов API (Application Programming Interface), т. е. создание системных функций, переменных и констант, к которым разработчик может обращаться из своего созданного приложения. Благодаря инкапсуляции методов реализации системных (операционных) функций, пользовательское приложение перестало зависеть от смены вариантов операционных систем (Windows 3, Windows 95 и т. д.). Благодаря этому (т. е. появлению универсальной частично инфинитной границы между пользователем и операционными системами, а также между самими пользователями) была создана совместимость программного обеспечения, разработанного на разных языках, и доступ к библиотекам различных фирм.

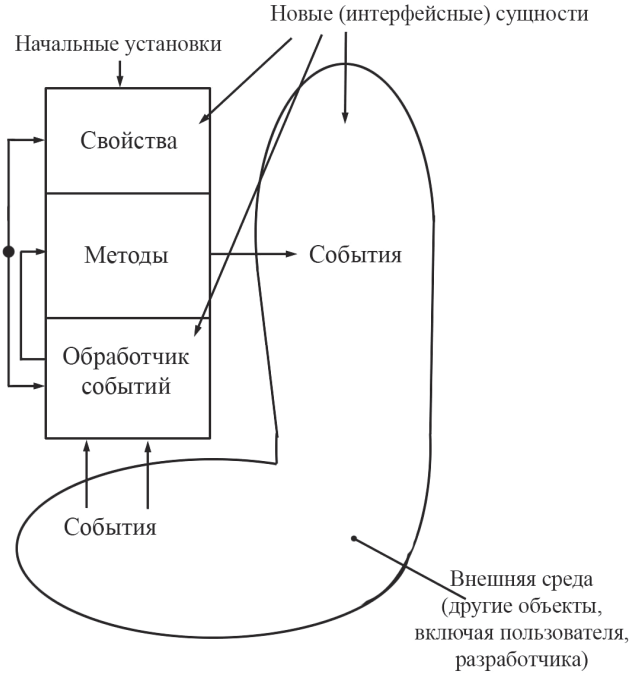
2. Создание визуального программирования, возникшего в Visual Basic (мы его будем изучать в версии C++ Builder). В двух словах, все это программирование сводится к созданию так называемой **формы** (окна), на которой размещаются различные **компоненты** (кнопки, диалоговые окна и т. п.). С их помощью можно задавать различные **свойства** и проводить разнообразные манипуляции: инициализировать **события** (например, нажимать кнопки) и писать алгоритмы их обработки (**обработчик** событий).

Все эти компоненты (формы, кнопки и т. д.) есть **объекты**, причем их типы (т. е. классы) создает сам программист. Хотя очень много классов создано другими. Эти классы синтаксически немного отличаются от классов C++ (там: данные и методы; здесь: свойства, методы, события), но они также могут наследоваться, что создает широкие возможности для полиморфизма.

1.1.2. Понятие объекта

Объект – ключевое понятие в C++ Builder. Его конкретизацию при переходе от C++ к C++ Builder иллюстрирует рис. 1.2.

а) class
{
данные (поля записи); // данные
конструкторы;
деструкторы; } методы
методы доступа;
другие методы;
}
б) class
{
данные + методы доступа = свойства;
конструкторы;
деструкторы; } методы
другие методы;
}
в)



г) объекты, как динамическое образование:

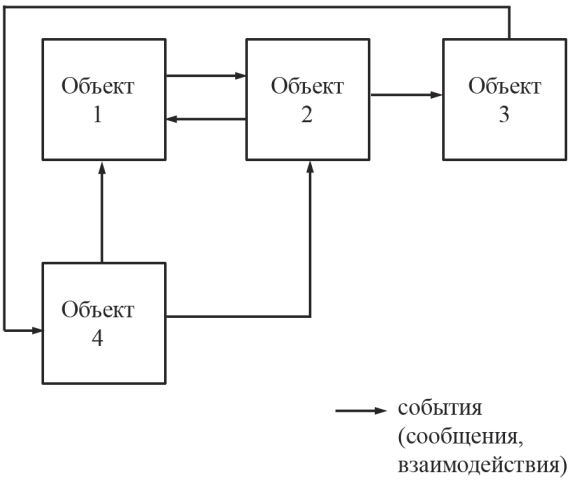


Рис. 1.2. Понятие объекта.

Переход от рис. 1.2, *а* к рис. 1.2, *б* хорошо поясняется следующими листингами, в которых доступ к полям в стандартном C++ (листинг 1.1) с помощью двух методов доступа заменен одним свойством (листинг 1.2).

Листинг 1.1.

```
0       class RiverBasin
1       {
2       private:
3          int itsF; //площадь водосбора
4       public:
5          int GetitsF () {return itsF; }; //метод доступа, возвращающий
        значение закрытой переменной-члена класса
6          void SetitsF (int F) //метод доступа, устанавливающий значение
        закрытой переменной-члена класса
7          {
8             if ((F >= 1000) && (F <= 50000))
9             itsF = F;
10         }
11      };
```

Листинг 1.2.

```
0     class RiverBasin
1     {
2     private:
3        int itsF;
4        int GetitsF () {return itsF; };
5        void SetitsF (int F)
6        {
7           if ((F >= 1000) && (F <= 50000))
8           itsF = F;
9        };
10    public: //свойство
11       __property int Friverbasin = {read= GetitsF,
      write= SetitsF };
12    };
13    //...
14    RiverBasin Oredez; //объявление объекта класса
15    Oredez.Friverbasin = 3220;
16    cout << Oredez.Friverbasin;
17    //...
```

Со свойством Friverbasin можно обращаться как с обычным полем класса (присвоение значений при этом выполняется методом SetitsF, а чтение – методом GetitsF).

Что касается событий, то о них еще будет речь впереди (как и о свойствах). Но вообще-то, события (как и обработчики событий) – это более детальная расшифровка интерфейса (рис. 1.2, *в*), реализуемого в C++ с помощью методов. События наступают в результате действия на объект других объектов (чаще всего – пользователя), например нажатия кнопки на форме. Они осуществляют взаимодействия (сообщения) объектов друг с другом. **Значением** события является указатель на некоторый метод (обработчик событий, рис. 1.2, *в*).

Программа в C++ Builder – это не просто жесткий набор строчек (предписаний), реализующий тот или иной алгоритм, а некая совокупность (подчас, «рыхлая») объектов и способов их взаимодействия. Причем эта совокупность динамическая (рис. 1.2, *г*): объекты могут «умирать» и «рождаться» по ходу работы программы.

1.2. Структура файлов в проектах языка C++ Builder

1.2.1. Блок-схема файлов

Программы в C++ Builder состоят из множества файлов. Большинство из них создается автоматически (без участия программиста) в процессе визуального проектирования. Файлы объединяются в **модули** (термин языком C++ Builder «заимствован» из Pascal). Этот факт важен тем, что можно менять содержание отдельных модулей, не затрагивая при этом другие модули и головную программу. Условно блок-схему файлов можно изобразить, как показано на рис. 1.3.

Каждой, создаваемой программистом форме (на жаргоне операционной системы Windows – окну) соответствует модуль (на блок-схеме показан штриховой линией), включающий в себя: заголовочный файл модуля и файл реализации модуля. Каждый модуль (проект) сопровождают файлы ресурсов проекта (содержат рисунки, «музыку», пиктограммы и т. д.) и файлы формы (содержат информацию о размере формы и т. п.). Кроме этих основных файлов проект содержит много других (создаваемых автоматически) – они будут «всплывать» ниже по тексту по мере необходимости.

Исполняемый файл (модуль) создается в несколько этапов. Препроцессор с соответствующими директивами включает тексты одних файлов в тексты других, а также разворачивает макросы (сокращенные обозначения различных выражений) и выполняет другие преобразования. Затем компилятор переводит текст модуля (модулей) в машинный (объектный) код (создает объектный файл модуля с расширением .obj). Компоновщик объединяет объектные файлы в единый загрузочный выполняемый модуль с расширением .exe).

Двойной рамкой на схеме обозначены файлы, с которыми обычно приходится иметь дело (чем жирнее рамка, тем чаще) разработчику, причем с файлом реализации дело приходится иметь всегда).

Программа инициализируется и запускается на выполнение главным (головным) файлом проекта, который C++ Builder создает для главной функции WinMain (в C++ и в консольных приложениях C++ Builder таковой является функция main ()).

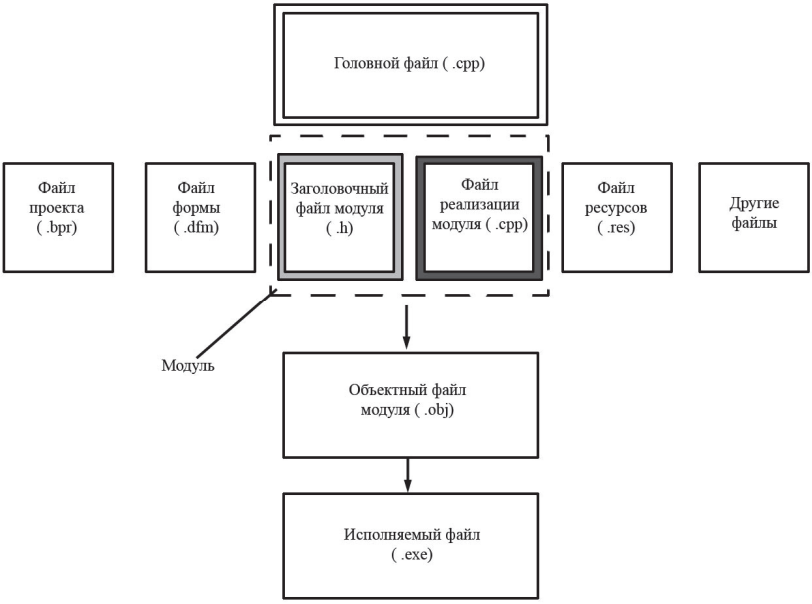


Рис. 1.3. Блок-схема файлов.

1.2.2. Синтаксис основных файлов

Рассмотрим синтаксическое оформление основных файлов (головного, файла реализации и заголовочного). Типичный головной файл проекта (создается автоматически) имеет вид, показанный в листинге 1.3.

Листинг 1.3

```
0    # include <vcl.h>
1    # pragma hdrstop
2    USERES ("Project.res"); //макрос для подключения к проекту файла ресурсов
3    USEFORM ("Unit1.cpp", Form1); //макрос для подключения к проекту формы 1
4    USEFORM ("Unit2.cpp", Form2); //макрос для подключения к проекту
```

```
    формы 2
5   WINAPI WinMain (HINSTANCE, HINSTANCE, LPSTR, int)
6   {
7     try
8     {
9      Application->Initialise();
10     Application->CreateForm(_classid(TForm1), &Form1);
11     Application->CreateForm(_classid(Tform2), &Form2);
12     Application->Run();
13    }
14    catch (Exception &exception)
15    {
16     Application->ShowException(&exception);
17    }
18    catch (...)
19    {
20       try
21       {
22          trow Exception(" ");
23       }
24       catch (Exception &exception)
25       {
26        Application-> ShowExceptoin (&exception);
27       }
28    }
29  return 0;
30  }
```

В первой строчке препроцессор подключает заголовочный файл vcl.h, содержащий объявления, используемые в библиотеке визуальных компонентов C++ Builder (об этом ниже). Новой является директива # pragma (вторая строчка), специфика которой состоит в том, что если компилятор не поддерживает имя директивы (в данном случае hdrstop), то он просто игнорирует директиву # pragma, не выдавая никаких сообщений об ошибке.

В строчках 2–4 идут макросы для подключения к проекту файлов ресурсов и форм (возможно и других).

В пятой строчке находится главная функция программы. WINAPI – что-то вроде «типа возвращаемого значения»: указатель на пользовательский интерфейс APIWindows. Первый параметр HINSTANCE – дескриптор – указатель данного экземпляра при-

ложения, а второй – предыдущего экземпляра (если выполняется одновременно несколько экземпляров приложений). LPSTR – указатель на строку с нулевым символом в конце, содержащую параметры, передаваемые в программу через командную строчку. Последний параметр определяет окно приложения.

Операторы тела функций заключаются в структуры, связанные с обработкой исключений. Это операторы (по пронумерованным строчкам):

9 Application -> Initialize() – инициализирует объекты компонентов;

10, 11 – создают объекты форм;

12 – начинает выполнение программы, после чего она ждет событий, которые управляют ее ходом.

Приведем пример заголовочного файла формы TForm1, на которой размещены два компонента (метка типа TLabel и кнопка типа TButton), листинг 1.4.

Листинг 1.4.

```
0    //Пример заголовочного файла
1    # ifndef Unit1H
2    # define Unit1H
3    // Автоматически подключаются копии файлов с описанием содержащихся в них компонентов
4    # include <Classes.hpp>
5    # include <Controls.hpp>
6    # include <StdCtrls.hpp>
7    # include <Forms.hpp>
8    // сюда можно самим помещать дополнительные директивы, не включенные в файл автоматически
9    class TForm1 : public TForm
10   {
11   __published:  //IDE-managed Components
12   // кнопка
13   TButton *Button1;
14   // статический текст
15   TLabel *Label1;
16   //Обработчик события OnClick объекта Button1
17   void __fastcall Button1Click (TObject *Sender);
18   private:  //помещаются объявления типов, переменных, функций, включенных в класс формы, но не доступных для других модулей
```

```
19    public:  //аналогично (но доступных)
20    //Конструктор класса TForm1
21    __fastcall TForm1 (TComponent * Owner);
22    };
23    //...
24    extern PACKAGE TForm1 *Form1;
25    // сюда помещается информация, не включенная в класс формы
26    # endif
```

К тому, что указано в комментариях можно добавить следующее. Все, что имеется в разделе **__published** включается автоматически. Новым в синтаксисе является опция компилятора **_fastcall**, которая ускоряет вызов функции TForm1 (для функций-элементов классов эту опцию надо указывать всегда).

Имеет смысл обратить внимание на строчку, объявляющую обработчик событий **void __fastcall** Button1Click (TObject *Sender). Тело этой функции будет представлено в файле реализации. В этом теле показывается, что должно быть выполнено при наступлении события (нажатии Click на кнопке Button1). Само имя функции генерируется автоматически. В качестве параметра функции используется Sender – указатель на объект, в котором происходит событие. Так как TObject является базовым классом VCL C++ Builder, а в VCL имеется группа классов-контейнеров для реализации общих алгоритмов и структур данных, то используют полиморфизм для их реализации с любым классом VCL.

Параметром конструктора **__fastcall** TForm1 (TComponent *Owner); является указатель Owner («владелец») на класс TComponent (его потомками являются все компоненты, так как в иерархии классов VCL имеет место следующая цепочка:
TObject -> TPersistent -> TComponent -> …)

Строчка
extern PACKAGE TForm1*Form1;
объявляет класс TForm1 доступным из других модулей.

Файл реализации модуля имеет структуру, представленную в листинге 1.5.

Листинг 1.5.

```
0     #include <vcl.h>
```

```
1    #pragma hdrstop
2    #include "Unit1.h"
3    //---------------------------------------------------------------------------
4    #pragma package(smart_init)
5    #pragma resource "*.dfm"
6    // могут помещаться и другие директивы
7    TForm1 *Form1; // объявление объекта формы Form1
8    //---------------------------------------------------------------------------
9    __fastcall TForm1::TForm1(TComponent* Owner)
10         : TForm(Owner) // вызов конструктора
11   {
12   // можно помещать различные операторы, связанные с созданием
     формы
13   }
14   //---------------------------------------------------------------------------
15   // объявления, реализации объявленных функций, объявления и реали-
     зация дополнительных функций
16   void __fastcall TForm1::Button1Click(TObject *Sender)
17   {
18   Form1->Close (); // закрыть форму
19   }
```

Текст файла реализации ясен из комментариев, в дальнейшем эти тексты будем рассматривать подробнее на конкретных примерах.

1.3. Основные элементы интегрированной среды разработки

При запуске C++ Builder на экране монитора открывается основное окно ИСР, показанное на рис. 1.4.

В ИСР есть все необходимое для проектирования, запуска и тестирования **приложения** (так в C++ Builder называют разрабатываемые программы).

Панель инструментов содержит **быстрые кнопки**, дублирующие наиболее востребованные команды меню.

В **палитре компонентов** VCL содержатся «страницы» (верхний ряд), группирующие близкие по назначению компоненты, иконки которых находятся в нижнем ряду.

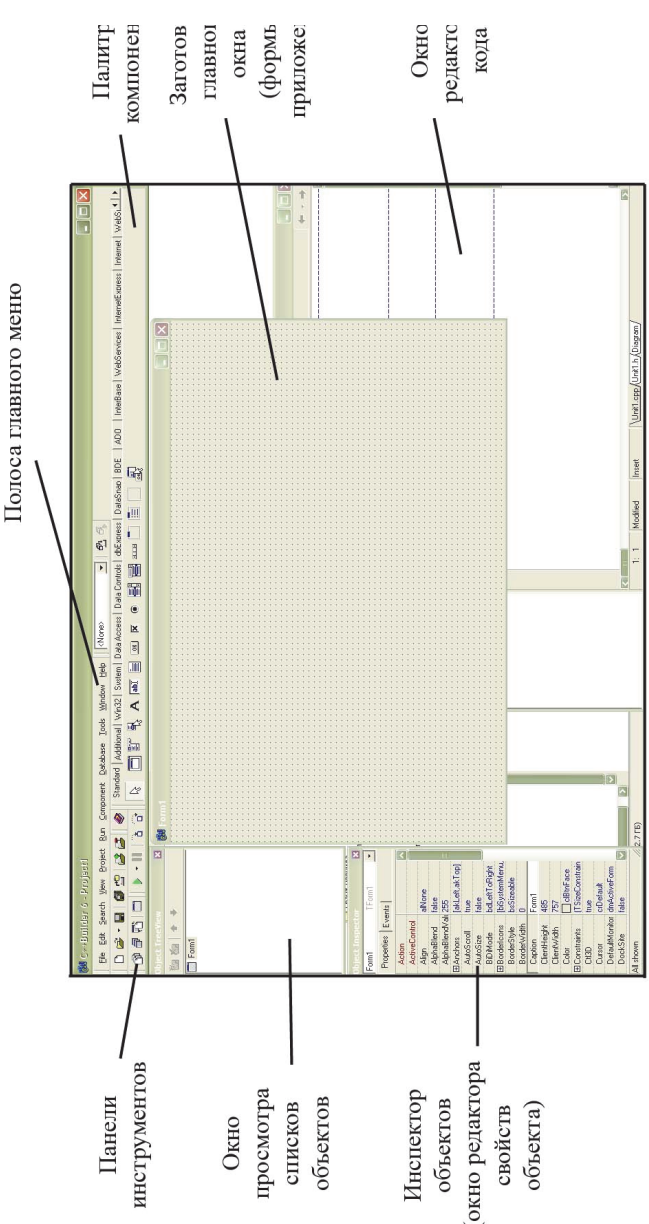


Рис. 1.4. Основное окно ИСР C++ Builder.

Основным элементом приложения является **форма**. Также как и окно Windows она может иметь меню, кнопки развертывания и свертывания и т. п. Можно менять его размеры, закрывать и т. д. На форме размещаются все другие компоненты, задаваемые в приложении.

Окно редактора кода, имеет три страницы, закладки которых видны в нижней части. Они отображают коды файлов реализации (Unit1.cpp), файла Diagram, заголовочного файла (Unit.h). Страница Diagram позволяет строить диаграммы, иллюстрирующие взаимодействия компонентов в приложении.

Окно просмотра списков объектов – Дерево объектов (Object Tree View) отображает иерархическую взаимосвязь визуальных и невизуальных компонентов и объектов, задействованных в приложении.

Инспектор объектов (Object Inspector) – основной инструмент для задания свойств компонентов и обработчиков событий. Это окно имеет две страницы: свойств (Properties) и событий (Events). **Свойство** (например цвет – выделяется на левой части страницы Properties) может иметь разные значения (например синий, красный и т. д. – выделяется на правой части этой страницы). На странице Events указаны **события**, на которые может реагировать выбранный объект. Например, если надо чтобы при нажатии на кнопку Button1 форма закрывалась, необходимо (выделив в Object Tree View кнопку Button1) «щелкнуть мышью по событию» OnClick страницы Events. Сделав двойной щелчок на пустом окне списка, мы автоматически попадаем в окно Редактора кода, в котором уже будет заготовлен шаблон

void _fastcall TForm1::Button1Click (TObject * Sender) { }

Остается только вписать в тело этого шаблона (обработчика событий) функцию Close ();.

Рассмотрим (пока очень кратко, опуская нюансы технического характера) технологию создания приложения, позволяющего вычислять уклон водной поверхности при равномерном течении реки.

1.4. Пример создания приложения

1.4.1. Формулировка задачи вычисления уклона водной поверхности

Создадим программу для вычисления уклона потока i при равномерном течении воды в реке. В основу положим формулу Шези: $i = u^2/C^2R$, где u – осредненная по сечению потока скорость (м/с); C – коэффициент Шези ($м^{0,5}/с$); R – гидравлический радиус (м).

Вид диалогового окна может быть, например, таким, как показано на рис. 1.5.

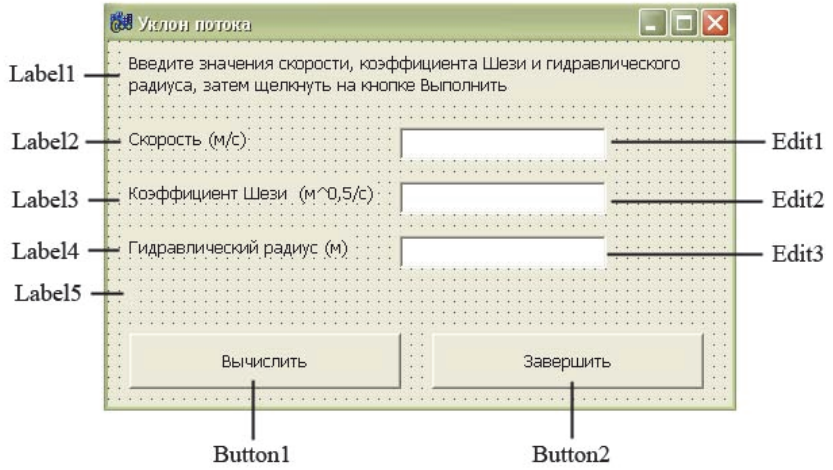


Рис. 1.5. Вид диалогового окна.

Создадим **проект** (приложение, т. е. прикладную программу), реализующий сформулированную задачу.

1.4.2. Форма

При активации C++ Builder на мониторе появляется стартовая форма Form1, которую надо «довести» до представленного вида диалоговго окна путем изменения **значений** ее **свойств** и добав-

ления ряда компонентов, позволяющих вводить и выводить информацию (**поля** ввода–вывода текстовой информации) и управлять работой программы (командные кнопки). Основными свойствами формы являются: Name (имя), Caption (текст заголовка), BorderStyle (вид границы; значения этого свойства позволяют манипулировать размерами окна); BorderIcons (кнопки управления окном); Icon (кнопка вывода системного меню); Color (цвет фона); Font (шрифт); Width, Height, Top, Left (геометрические характеристики положения окна).

Заменим значения свойства Caption с текста Form1 на «Уклон потока». Для этого надо щелкнуть по строке Caption на левой стороне вкладки Properties и набрать на клавиатуре текст «Уклон потока» на правой стороне этой вкладки.

Для некоторых свойств имеются дополнительные значки (▼ – для выбора значения свойства из раскрывающегося списка; + – для раскрытия списка уточняющих свойств; … – возможность задавать значение свойства в дополнительном диалоговом окне, например для свойства **шрифт**).

Зададим для нашего окна следующие значения свойств: BorderStyle(bs.Single); BorderIcons.biMinimize (False); BorderIcons.biMaximize (False); Font.Name(Tahoma); Font.Size(10). (Применение оператора доступа указывает на тот факт, что само свойство является объектом.) Что касается размеров формы, то при манипуляции мышью они устанавливаются автоматически.

1.4.3. Компоненты

В соответствии с желаемым видом диалогового окна, на форме надо разместить компоненты: 3 – Edit (поля редактирования), 5 – Label (поля вывода текста), 2 – Button (командные кнопки). Все эти компоненты находятся на вкладке Standart палитры компонентов (см. рис. 1.6) и устанавливаются по одинаковой технологии: щелчок на значке компонента; затем щелчок на форме в месте, где необходимо поместить компонент; манипуляция мышью, чтобы окончательно определиться с местом установки компонентов и их размерами.

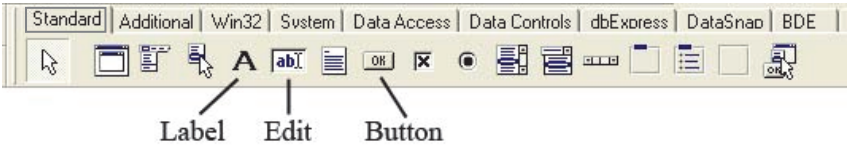


Рис. 1.6. Вкладка Standard.

Для компонентов Edit в нашем примере значением свойства Text является «пустая строка», которая будет заполняться при вводе информации. Остальные свойства остаются без изменения.

Для компонентов Label нам потребуются следующие значения их свойств (табл. 1.1).

Таблица 1.1

Свойства компонента Label

Label / Свойство	1	2	3	4	5
AutoSize	false	true	true	true	false
WordWrap	true	true	false	false	true
Caption	*	Скорость (м/с)	Коэффициент Шези (м^0.5/с)	Гидравлический радиус	

* Ввести значения скорости, коэффициента Шези и гидравлического радиуса, затем щелкнуть на кнопке Вычислить.

Свойства AutoSize и WordWrap связывают размер поля с его содержимым и определяют возможность автоматического переноса слов на следующую строку, если они не помещаются в текущей.

Для командных кнопок Button необходимо задать свойство Caption (текст на кнопке): Вычислить и Завершить.

1.4.4. События и обработчик события

Одним из основных понятий C++ Builder является **событие** (Event) – то, что происходит во время работы программы. Примеры: OnClick (при щелчке кнопкой мыши), OnMouseMove (при перемещении мыши), OnCreate (при создании объекта) и т. д.

При возникновении события происходит вызов **обработчика события** – функции, реализующей реакцию на действия пользователя. В нашем случае возможно событие OnClick, возникающее при нажатии на кнопки Вычислить и Завершить. Технология создания обработчика события такова. Выделяем компонент, который надо «обработать» (это можно сделать либо в окне Object Inspector, либо щелчком на изображении компонента в форме). Затем на вкладке Events выделяем событие (в нашем случае – OnClick) для обоих компонентов Button1 и Button2). После этого делаем двойной щелчок в поле справа от выбранного события. При этом автоматически в окно Редактора кода будет добавлен шаблон обработчика, в тело которого надо вписать необходимый код обработчика. Имя его формируется из имени формы (TForm1), содержащей компонент, и автоматически сконструированного имени функции, например Button1Click. В нашем случае обработчики имеют вид, представленный в листинге 1.6.

Листинг 1.6.

```
0    void __fastcall TForm1::Button1Click(TObject *Sender)
1    {
2    float u, C, R, i;
3    u = StrToFloat (Edit1 -> Text);
4    C = StrToFloat (Edit2 -> Text);
5    R = StrToFloat (Edit3 -> Text);
6    i = (u*u)/(C*C*R);
7    Label5 -> Caption = "Уклон: " + FloatToStrF(i, ffGeneral, 7, 2);
8    }
9    //---------------------------------------------------------------------------
10
11   void __fastcall TForm1::Button2Click(TObject *Sender)
12   {
13   Form1 -> Close();
14   }
```

Здесь мы встречаемся с тремя стандартными функциями языка C++ Builder:

1. StrToFloat( ) – переводит строку свойства Text в число;
2. FloatToStr ( ) – преобразует число *i* по формату ffGeneral в строку символов;
3. Close ( ) – метод, закрывающий окно программы.

В Редакторе кода существует система контекстнозависимой подсказки, которая выводит список свойств, методов и параметров (список не выводится, если C++ Builder обнаруживает ошибки в набираемом программистом коде), рис. 1.7.

В левой части Редактора кода имеется **навигатор классов**, облегчающий навигацию по тексту программы. Программисту также предлагается стандартный набор шаблонов и возможность создавать свои собственные. Имеется справочная система.

После создания всех обработчиков, проект надо сохранить (File – Save Project As, при этом C++ Builder предлагает в начале сохранить содержание окна Редактора кода: Save Unit As), откомпилировать (Project – Compile) и запустить на выполнение (Run – Run). В результате появится окно программы «Уклон потока», в поля которого надо ввести значения *u*, *C*, *R*; нажать кнопку Вычислить; прочитать значение уклона *i* и нажать кнопку Завершить.

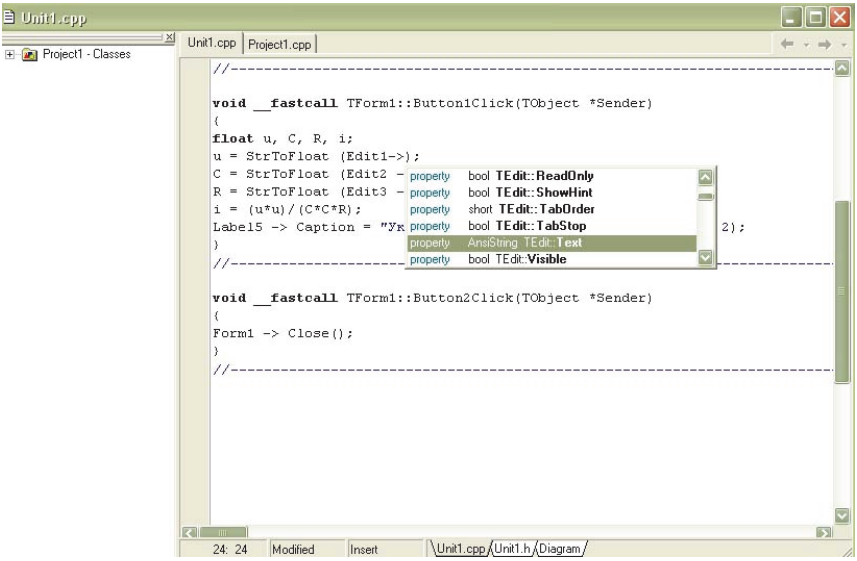


Рис. 1.7. Контекстнозависимая подсказка.

1.5. Усовершенствование проекта

1.5.1. Ошибки времени выполнения (исключения) и их обработка автоматически добавляемым в программу кодом

Во время выполнения программы могут возникать ожидаемые ошибки (исключения), которые обрабатываются автоматически добавленным в программу кодом (см. Головной файл проекта, листинг 1.3) **try** {**throw** } **catch** { }. Наиболее частыми причинами возникновения исключений являются: деление на ноль (EZeroDivide) (например, в нашем случае, при задании *C* или *R* равными нулю), а также ошибки преобразования (EConvertError) (например невозможность преобразования строки символов 10.5 в число при выполнении инструкции u = StrToFloat (Edit1 –> Text), если настройка Windows в качестве разделителя целой и дробной части использует запятую. В этом случае при запуске программы на экране появится сообщение, призывающее пользователя к определенным действиям, рис. 1.8.

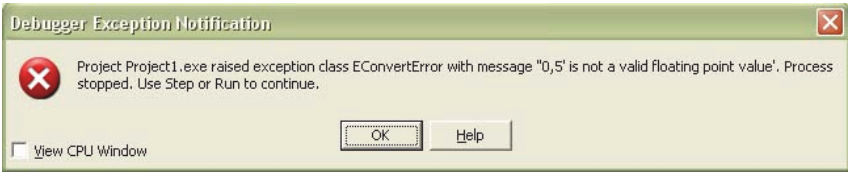


Рис. 1.8. Пример сообщения о возникновении исключения.

1.5.2. Авторская обработка исключений

Программист может самостоятельно задавать код для обработки исключений с выдачей требуемого сообщения. Для этого используют стандартные функции вызова диалоговых окон с сообщением, например ShowMessage ( ) или MessageDlg ( ). Если в первом случае отображаются окна, в которых пользователь, прочтя текст, просто нажимает Ok, то во втором случае пользователю задается вопрос и анализируется полученный ответ.

Внесем изменения в файл реализации (uklon.cpp), листинг 1.7.

Листинг 1.7.

```
0    void __fastcall TForm1::Button1Click(TObject *Sender)
1    {
2    float u, C, R, i;
3    try
4    {
5    u = StrToFloat (Edit1->Text);
6    C = StrToFloat (Edit2 -> Text);
7    R = StrToFloat (Edit3 -> Text);
8    }
9    catch (EConvertError&)
10   {
11   ShowMessage("Разделителем является запятая.");
12   return ;
13   }
```

Далее без изменений. В этом случае при выполнении программы сначала появится стандартное сообщение (рис. 1.8), а затем, после нажатия кнопки Ok (в окне сообщения) и Run, – «авторский вариант окна с сообщением», рис. 1.9.

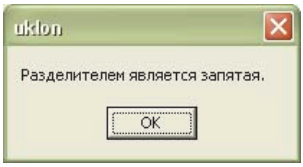


Рис. 1.9. Сообщение.

1.5.3. Модернизация программы «Уклон потока»

Сделаем два усовершенствования: 1) желательно, чтобы курсор, находящийся в поле Edit, после нажатия Enter переходил в «пустое» поле Edit; 2) попытаемся «запретить» нахождение в поле Edit всего, что не относится к числам и клавишам <Backspace> и Enter. Для этого надо: 1) внести дополнения в код, позволяющие проверять заполнения поля Edit*i* (i = 1, 2, 3); 2) внести три обработчика событий для компонентов Edit*i*: OnKeyPress. Листинг может выглядеть так:

Листинг 1.8.

```
0    void __fastcall TForm1::Button1Click(TObject *Sender)
1    {
2    float u;
3    float C;
4    float R;
5    float i;
6    if (((Edit1->Text).Length()==0) || ((Edit2->Text).Length()==0) ||
     ((Edit3->Text).Length()==0))
7    {
8    MessageDlg ("Надо ввести скорость, коэффициент Шези, гидравли-
     ческий радиус", mtInformation, TMsgDlgButtons() << mbOK, 0);
9    if ((Edit1->Text).Length()==0)
10        Edit1->SetFocus();
11   if ((Edit2->Text).Length()==0)
12        Edit2->SetFocus();
13   if ((Edit3->Text).Length()==0)
14        Edit3->SetFocus();
15   return ;
16   }
17   u = StrToFloat (Edit1->Text);
18   C = StrToFloat (Edit2 -> Text);
19   R = StrToFloat (Edit3 -> Text);
20   try
21   {
22        i = (u*u)/(C*C*R);
23   }
24   catch (EZeroDivide &e)
25   {
26        ShowMessage("На ноль делить нельзя");
27        return ;
28   }
29   Label5 -> Caption = "Уклон: " + FloatToStrF(i, ffGeneral, 7, 2);
30   }
31   //---------------------------------------------------------------------------
32
33   void __fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)
34   {
35   if ((Key >= '0') && (Key <= '9'))
36   return ;
37   if (Key == DecimalSeparator)
38   {
```

```
39          if ((Edit1->Text).Pos(DecimalSeparator) != 0)
40                Key = 0;
41          return ;
42    }
43    if (Key == VK_BACK)
44          return ;
45    if (Key == VK_RETURN)
46          {
47          Edit2->SetFocus();
48          return ;
49          }
50    Key = 0;
51    }
52    //---------------------------------------------------------------------------
53
54    void __fastcall TForm1::Edit2KeyPress(TObject *Sender, char &Key)
55    {
56    if ((Key >= '0') && (Key <= '9'))
57    return ;
58    if (Key == DecimalSeparator)
59    {
60          if ((Edit1->Text).Pos(DecimalSeparator) != 0)
61                Key = 0;
62          return ;
63    }
64    if (Key == VK_BACK)
65          return ;
66    if (Key == VK_RETURN)
67          {
68          Edit3->SetFocus();
69          return ;
70          }
71    Key = 0;
72    }
73    //---------------------------------------------------------------------------
74
75    void __fastcall TForm1::Edit3KeyPress(TObject *Sender, char &Key)
76    {
77    if ((Key >= '0') && (Key <= '9'))
78    return ;
79    if (Key == DecimalSeparator)
80    {
```

```
81          if ((Edit1->Text).Pos(DecimalSeparator) != 0)
82                Key = 0;
83          return ;
84    }
85    if (Key == VK_BACK)
86          return ;
87    if (Key == VK_RETURN)
88          {
89          Edit3->SetFocus();
90          Button1->SetFocus();
91          return ;
92          }
93    Key = 0;
94    }
95    //---------------------------------------------------------------------------
96    void __fastcall TForm1::Button2Click(TObject *Sender)
97    {
98    Form1 -> Close();
99    }
100   //---------------------------------------------------------------------------
```

В листинге убрана проверка ошибок (исключений) на предмет ввода «неправильной информации» (так как сделана специальная защита, исключающая «неправильный ввод»). Появились новые (для нас) методы Length ( ), SetFocus ( ), Pos (DecimalSeparator) (здесь DecimalSeparator – глобальная переменная, содержащая символ, используемый в качестве разделителя при задании дробных чисел), действия которых очевидны.

После модификации программы и ее отладки можно изменить значок, изображающий исполняемый файл приложения (в папке, на рабочем столе и на панели задач). Его можно выбрать из предлагаемого списка или «нарисовать» самим. На возможных проблемах, которые могут возникнуть при переносе приложения на другой компьютер, останавливаться не будем, см. [9].

Обратим внимание на изменения, которые автоматически произошли в файлах приложения.

В головном файле нет подключения файла ресурсов (3 строчка), но появился макрос для подключения формы Form1:: USEFORM («uklon1.cpp», Form1).

В заголовочном файле две первые строчки приняли вид

```
# ifndef uklonH
# define uklonH
```

Появился также перечень размещенных на форме компонентов и обработчиков событий:

```
__published: // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TEdit *Edit1;
        TEdit *Edit2;
        TEdit *Edit3;
        TButton *Button1;
        TButton *Button2;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Edit1KeyPress(TObject *Sender, char &Key);
        void __fastcall Edit2KeyPress(TObject *Sender, char &Key);
        void __fastcall Edit3KeyPress(TObject *Sender, char &Key);
```

Обратим также внимание на содержание папки, в которой находятся файлы, включенные в исполняемый файл приложения, рис. 1.10.

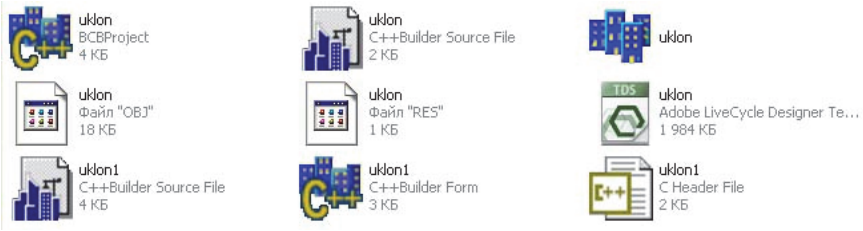


Рис. 1.10. Файлы проекта.

Обсуждением некоторых из них займемся по мере изложения материала.

2. Графика и мультипликация

2.1. Графические примитивы (пример логотипа РГГМУ)

Графическими примитивами будем называть: линии, эллипсы, дуги, прямоугольники и т. д. Из них можно создавать много рисунков. Рассмотрим их реализацию на примере логотипа РГГМУ, рис. 2.1.

Рис. 2.1. Логотип РГГМУ.

Методы, реализующие примитивы, применяются к свойству Canvas (холст) формы Form. Это свойство само является объектом, к которому применяются методы, реализующие примитивы: LineTo (x, y) – рисует линию из текущей точки в точку с координатами x и y (координаты на форме увеличиваются слева направо – x, и сверху вниз – y); Rectangle (x1, y1, x2, y2) – рисует прямоугольник с координатами x1, y1 (левый верхний угол) и x2, y2 (правый нижний угол); Ellipse (x1, y1, x2, y2) – рисует эллипс, причем x1, y1, x2, y2 – координаты прямоугольника, внутри которого вычерчивается эллипс; Polyline (p, n) – рисует ломанную линию из n звеньев (здесь p – массив координат) (а также другие примитивы). Например, оператор

Form1 –> Canvas – > Ellipce (15, 15, 20, 20);

рисует эллипс.

У объекта Canvas есть свойства Pen (карандаш) и Brush (кисть), с помощью которых можно менять цвет, толщину, тип линий и границы (Pen), а также – цвет и способы закраски внутренних фигур, нарисованных на основе примитивов (Brush).

Листинг модуля «rshu.cpp» (листинг 2.1), рисующего логотип РГГМУ, имеет вид.

Листинг 2.1.

```
0     #include <vcl.h>
1     #pragma hdrstop
2
3     #include "rshu.h"
4     //---------------------------------------------------------------------------
5     #pragma package(smart_init)
6     #pragma resource "*.dfm"
7     TForm1 *Form1;
8     //---------------------------------------------------------------------------
9     __fastcall TForm1::TForm1(TComponent* Owner)
10          : TForm(Owner)
11    {
12    }
13    //---------------------------------------------------------------------------
14    void __fastcall TForm1::FormPaint(TObject *Sender)
15    {
16    Canvas -> Pen -> Width = 5;
17    Canvas -> MoveTo (295, 196);
18    Canvas -> LineTo (295, 146);
19    Form1 -> Canvas -> Arc (184, 196, 417, 500, 417, 348, 184, 348);
20    Form1 -> Canvas -> Arc (242, 196, 355, 500, 355, 348, 242, 348);
21    Canvas -> MoveTo (417, 348);
22    Canvas -> LineTo (184, 348);
23    Canvas -> MoveTo (264, 146);
24    Canvas -> LineTo (332, 146);
25    Form1 -> Canvas -> Ellipse (224, 126, 264, 166);
26    Form1 -> Canvas -> Arc (200, 154, 401, 250, 200, 250, 401, 250);
27    TPoint p[6];
28    p[0].x = 332; p[0].y = 146;
29    p[1].x = 352; p[1].y = 126;
30    p[2].x = 382; p[2].y = 126;
31    p[3].x = 368; p[3].y = 146;
32    p[4].x = 382; p[4].y = 166;
33    p[5].x = 352; p[5].y = 166;
34    Canvas -> Polygon (p, 5);
35    AnsiString ms = "РГГМУ";
36    int x = 200;
37    int y = 350;
38    Canvas -> Font -> Name = "Arial";
39    Canvas -> Font -> Size = 48;
40    Canvas -> Brush -> Style = bsClear;
```

```
41    Canvas -> Font -> Color = clBlack;
42    Canvas -> TextOutA(x, y, ms);
43    }
44    //---------------------------------------------------------------------------
```

Прокомментируем элементы кода, с которыми встречаемся впервые. Событие OnPaint (см. четырнадцатую строку с обработчиком FormPaint) возникает в момент запуска программы, все инструкции, формирующие логотип на поверхности формы, помещены в функцию обработки этого события. Свойство Width объекта Pen имеет значение 5 пикселей (задает толщину линий логотипа). Метод MoveTo помещает карандаш в начало рисуемых линий (это начало задается параметрами метода; в нашем случае, например, аргументами 295, 196 – координатами точки начала линии).

Метод Arc (x1, y1, x2, y2, x3, y3, x4, y4) рисует дугу (пример см. рис. 2.2).

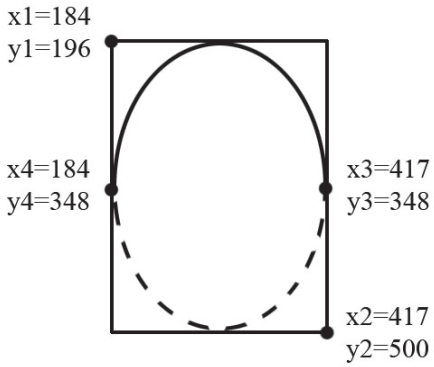


Рис. 2.2. Действия метода Arc (x1, y1, x2, y2, x3, y3, x4, y4).

Строчкой TPoint p[6] объявляется массив p[6] типа TPoint, а ниже этой строчки каждому элементу массива задаются координаты x и y.

Последними шестью строчками кода обеспечивается вывод надписи РГГМУ, рис. 2.3:

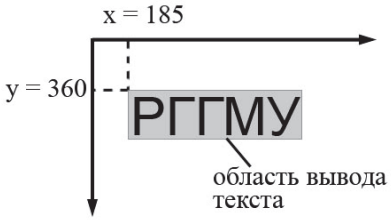


Рис. 2.3. Вывод надписи.

Кисти (Brush) «предлагается» область вывода текста не закрашивать (Canvas -> Brush -> Style -> bsClear: свойству Style «присваивается» значение bsClear). В последней строчке вызывается метод TextOutA ( ), который выводит текст (переменная ms) в области вывода с координатами левого верхнего угла x = 185, y = 360.

2.2. Построение графиков функций. Биффуркационные диаграммы

Начнем с самого простого – построения графиков, соответствующих информации, задаваемой в массивах. Пусть, например, задан (или спрогнозирован) ряд расходов воды в виде массива:
float Q[12] = {56, 67, 123, 42, 23, 14, 4, 56, 23, 11, 19, 41};

Проще всего визуализировать этот массив с помощью метода LineTo (,), как показано в листинге 2.2.

Листинг 2.2.

```
0     void __fastcall TForm1::FormPaint(TObject *Sender)
1     {
2     float y[12] = {56, 67, 123, 42, 23, 14, 4, 56, 23, 11, 19, 41};
3     float x[12] = {25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300};
4     Canvas -> Pen -> Width = 1;
5     Canvas -> Pen -> Color = clRed;
6     for (int i=0; i<12; i++)
7     {Canvas -> LineTo (x[i], y[i]);}
8     }
```

В этом листинге массив **float** y[12] = { } имитирует расходы воды, а x[12] = { } – координаты по оси времени (дискретность выбирается произвольно, исходя из удобства визуализации). Свойство карандаша Width позволяет менять ширину линии, а Color – ее цвет. Сам процесс вырисовывания осуществляется в цикле **for** ( ) { } (разумеется в данном примере фигурные скобки необязательны) график имеет вид, представленный на рис. 2.4.

Рис. 2.4. Вывод графика в листинге 2.2.

В данном примере имеется ряд несуразиц. Параметры в методе LineTo(,) должны иметь тип int, что не выполняется для переменных массива y[ ]. Компилятор может эту ситуацию и пропустить, но в определенных случаях возникают исключения (ошибки времени выполнения).

Корректно графики можно рисовать разными способами. Распространены варианты с использованием либо свойства Pixel, либо с помощью карандаша Pen (методом LineTo(x, y)). В первом случае используется двумерный массив типа TColor, несущий информацию о цвете точек графической поверхности. Синтаксис такой:
Canvas – > Pixels [10][10] = clRed;
В данном случае точка (пиксель) с координатами x = 10, y = 10 окрашивается в красный цвет. Перебирая в цикле пиксели, получим набор точек визуально напоминающую линию. Во втором случае карандаш прочерчивает более реальную линию из исходной точки в точку (x, y).

В обоих случаях «шаблон» кода практически одинаков:

```
float X, Y; // объявляются переменные, представляющие координаты функции, график которой надо нарисовать
int PX, PY; // координаты пикселей
for (PX = 0; PX < Image1 -> Width; PX++)
{
X = F1(PX); // абсциссе X (тип float) функции Y = f(X) ставится в соответствие пиксель с координатой PX (тип int)
Y = f(X); // вычисление ординат функции, для которой строится график
PY = F2(Y); // ордината Y функции ставится в соответствие координате пикселя PY
// далее следуют два возможных варианта получения графика
// или:
Image1 -> Canvas -> Pixels[PX][PY] = clRed; // устанавливается нужный цвет для пикселя
// или:
Image1 -> Canvas -> LineTo(PX, PY); // линия в точку (PX, PY)
}
// Преобразования X = F1(PX) и PY = F2(Y) производятся так:
X = Xmin + PX*(Xmax – Xmin)/Image1 -> Width;
PY = Image1 -> Height – (Y – Ymin)*Image1 -> Height/(Ymax – YMin);
```

Значения Xmin, Xmax, Ymin, Ymax должны тем или иным способом определяться программистом исходя из характера функции Y = f(X) и диапазона изменения ее аргумента X. Рис. 2.5 поясняет геометрический смысл использованных обозначений.

В листинге 2.3 представлен код, позволяющий строить график функции y = (sin X) * (sin(X*X*X)), а также бифуркационную диаграмму для отображения вида $y_{i+1} = y_i(1 - y_i + r)$. (Это отображение получается из модели линейного фильтра $dQ/dt = -(1/k\tau)Q + \dot{X}/\tau$, если принять $\tau = W/Q$, где $W = Q_0T$ емкость среды, Q_0 – норма стока, $T = 1$ год. При этом принимается $kQ_0 = 1$, что не важно с точки зрения решаемой задачи – построения графиков. Примем параметр бифуркации $r = \dot{X}/Q_0$ – величина обратная коэффициенту стока (в листинге это X/305).

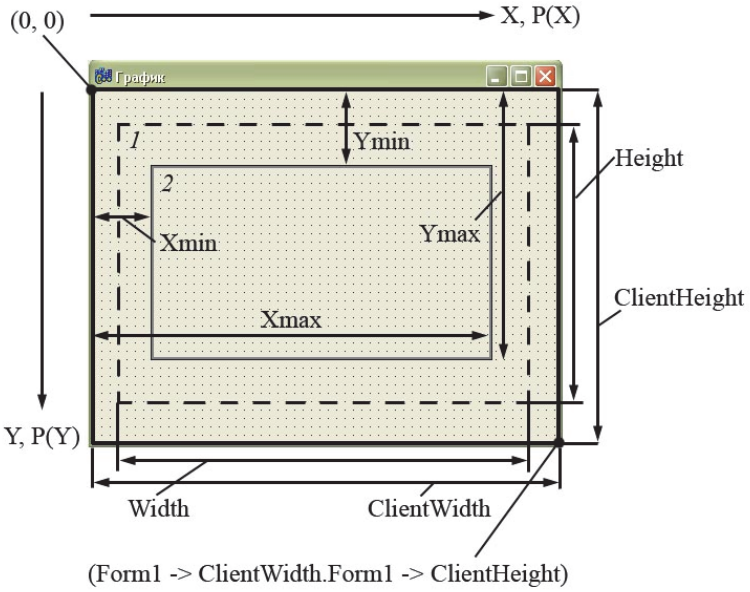


Рис. 2.5. К пояснению использованных обозначений: *1* – Image; *2* – область вывода графика.

Листинг 2.3.

```
0    //---------------------------------------------------------------------------
1    void __fastcall TForm1::Button1Click(TObject *Sender)
2    {
3    # define Pi 3.14159
4    float X, Y;
5    int PX, PY = 1;
6    float Y1[6990] = {1.16019};
7    for (PX=0; PX <= Image1 -> Width; PX++)
8    {
9    //X = PX*4*Pi/Image1 -> Width;
10   X = 680 + PX*122/Image1 -> Width;
11   //Y = (sin(X)*sin(X*X*X));
12   Y1[PX+1] = Y1[PX]*(1 - Y1[PX] + X/305);
13   Y = Y1[PX + 1];
14   //PY = Image1 -> Height - (Y + 1)*Image1 -> Height/2;
15   PY = Form1 -> Image1 -> Height - (Y + 1)*Image1 -> Height/4.169231;
```

```
16    //Form1 -> Image1 -> Canvas -> Pixels[PX][PY] = clBlack;
17    Form1 -> Image1 -> Canvas -> LineTo(PX, PY);
18    }
19    }
20    //---------------------------------------
```

В данном листинге закомментированы строчки, необходимые для построения бифуркационной диаграммы и вывода графиков с помощью пикселей.

На рис. 2.6 представлены результаты выполнения кода для функции и отображения при использовании обоих методов рисования.

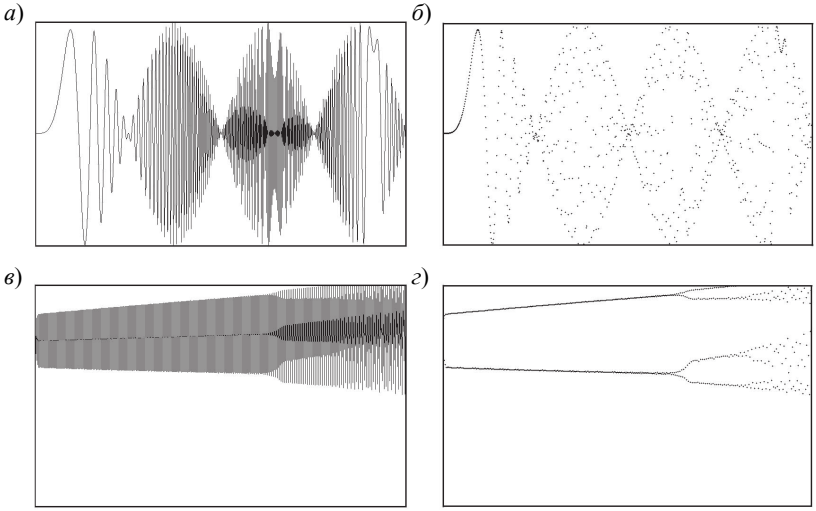


Рис. 2.6. Графики, построенные карандашом (*а*, *в*) и методом LineTo (,) (*б*, *г*).

Как правило, с точки зрения хорошего прочерчивания графика, предпочтительно использовать карандаш, но не всегда. Например, рисунок 2.6, *г* наглядно показывает, как бифуркационная диаграмма «рассыпается в пыль» при значительных гидрологических потенциалах $\dot{X}/Q_0$.

2.3. Простая мультипликация (с использованием метода базовой точки)

Мультипликация – это «заставить» рисунок «дышать» (двигаться или как-то меняться). Технология мультипликации сводится к следующему: 1. Вывести рисунок; 2. Стереть его; 3. Снова вывести рисунок на некотором расстоянии от места первого вывода на экран; 4. Подбирая промежутки времени и расстояния на экране между выводами рисунков, можно создать иллюзию перемещения рисунка по экрану.

Ключевым звеном в создании этой иллюзии является компонент Timer (вкладка System). Этот компонент невизуальный (он не отображается в диалоговом окне во время работы программы). Его основные свойства: Interval (период возникновения события OnTimer – смена положения рисунка); Enabled (разрешение или запрещение работы Timer).

Ниже будет программа, в которой функция Ship ( ) рисует на форме кораблик. В качестве ее параметра выступает «базовая точка» (x_0, y_0), от которой отсчитываются координаты графических примитивов, образующих корабль.

В листинге 2.4 есть функция Ship ( ), рисующая корабль, обработчик события OnTimer и функция FormCreate ( ), настраивающая таймер.

На рис. 2.7 показана базовая точка и относительные координаты корпуса корабля.

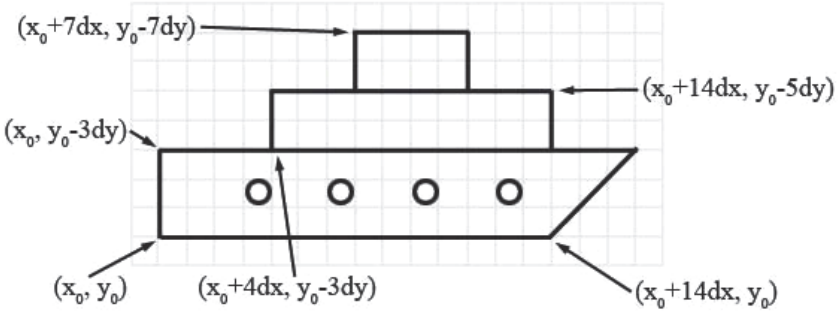


Рис. 2.7. Координаты корабля.

Листинг 2.4.

```
0    #include <vcl.h>
1    #pragma hdrstop
2
3    #include "Ship_.h"
4
5    #pragma package(smart_init)
6    #pragma resource "*.dfm"
7
8    TForm1 *Form1;
9    int x = -50, y = 50; // координаты базовой точки в начальный мо-
     мент времени
10
11   __fastcall TForm1::TForm1(TComponent* Owner)
12       : TForm(Owner)
13   {
14   }
15
16
17   // функция рисования корабля на форме
18   void __fastcall TForm1::Ship(int x, int y)
19   {
20    int dx=4,dy=4; // шаг
21    TPoint p1[5];
22   // переменные для хранения цвета карандаша и кисти
23    TColor pen_cl,brush_cl;
24    Canvas->Pen->Color = clBlack;
25    Canvas->Brush->Color = clWhite;
26    pen_cl = Canvas->Pen->Color;
27    brush_cl = Canvas->Brush->Color;
28    //нижняя часть корабля
29    p1[0].x = x;      p1[0].y = y;
30    p1[1].x = x;      p1[1].y = y-3*dy;
31    p1[2].x = x+17*dx; p1[2].y = y-3*dy;
32    p1[3].x = x+14*dx; p1[3].y = y;
33    p1[4].x = x;      p1[4].y =y;
34    Canvas->Polygon(p1,4);
35   //средняя часть корабля
36    Canvas->Rectangle(x+4*dx,y-3*dy,x+14*dx,y-5*dy);
37   //верхняя часть корабля
38    Canvas->MoveTo(x+7*dx,y-5*dy);
39    Canvas->LineTo(x+7*dx,y-7*dy);
```

```
40      Canvas->LineTo(x+11*dx,y-7*dy);
41      Canvas->LineTo(x+11*dx,y-5*dy);
42      // иллюминаторы
43      Canvas->Brush->Color = clYellow;
44      Canvas->Ellipse(x+3*dx,y-2*dy,x+4*dx,y-1*dy);
45      Canvas->Ellipse(x+6*dx,y-2*dy,x+7*dx,y-1*dy);
46      Canvas->Ellipse(x+9*dx,y-2*dy,x+10*dx,y-1*dy);
47      Canvas->Ellipse(x+12*dx,y-2*dy,x+13*dx,y-1*dy);
48      // восстановление цвета карандаша и кисти
49      Canvas->Pen->Color = pen_cl;
50      Canvas->Brush->Color = brush_cl;
51    }
52
53    // обработка события OnTimer
54    void __fastcall TForm1::Timer1Timer(TObject *Sender)
55    {
56      // корабль закрашивается цветом фона
57      Canvas->Brush->Color = Form1->Color;
58      Canvas->FillRect(Rect(x-1,y+1,x+68,y-40));
59
60      // новые координаты базовой точки
61      x+=3;
62      if (x > ClientWidth)
63      {
64        x= -50;
65        y=random(Form1->ClientHeight);
66      }
67     Ship(x,y);
68    }
69
70
71    // обработка события OnCreate для формы
72    void __fastcall TForm1::FormCreate(TObject *Sender)
73    {
74      // настройка и запуск таймера
75      Timer1->Interval = 100; // период возникновения события OnTimer
      - 0.1 сек.
76      Timer1->Enabled = true; // пуск таймера
77    }
```

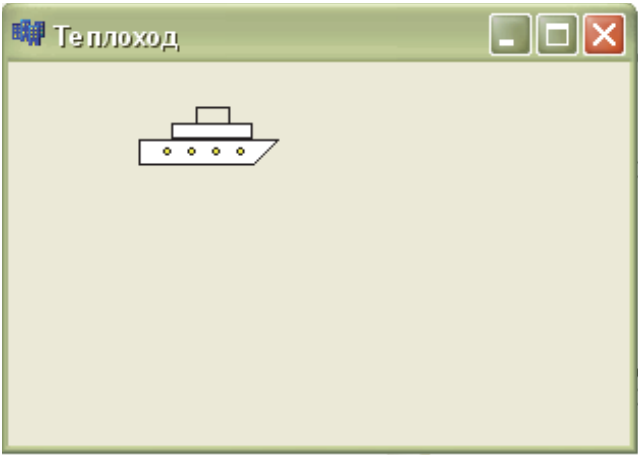


Рис. 2.8. Результат работы программы из листинга 2.4.

3. Текстовый редактор

3.1. Необходимые компоненты

MainMenu. С помощью компонента Main Menu (вкладка Standard) в программу вводится «главное меню», которое обычно присутствует в приложениях, работающих в Windows. Для настройки пунктов главного меню существует **дизайнер меню** (вызов: два щелчка по компоненту MainMenu на форме), рис. 3.1.

Рис. 3.1. Дизайнер меню.

Наращивание меню по пунктам (по горизонтали) и по командам каждого из пунктов (по вертикали) производится щелчком по пунктам меню, введением в свойство Caption соответствующих названий и нажатием клавиши Enter. При щелчке по командной кнопке открывается окно инспектора кодов с заготовкой функции, в тело которой надо записать исполняемый код (допустимые коды предлагает сама среда разработки с помощью системы подсказок).

Memo. С помощью компонента Memo (вкладка Standard) на форме создается область для ввода текста. С помощью свойств этого компонента можно менять цвет фона (Color), тип курсора (Cursor), характер отображаемого текста (Lines), добавлять полосы прокрутки (ScrollBars) и др. (рис. 3.2).

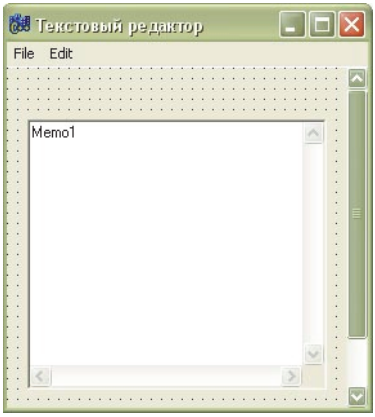


Рис. 3.2. Компонент Memo.

ImageList. Для хранения изображений, используемых при создании программ (в том числе и текстового редактора), применяется компонент ImageList (вкладка Win32). При двойном щелчке на этом компоненте, помещенном на форму, появляется окно редактора ресурсов компонента, рис. 3.3.

Рис. 3.3. Компонент ImageList.

При нажатии на кнопке Add открывается диалоговое окно, позволяющее выбирать файл с рисунком и загружать его в контейнер ImageList с автоматической индексацией.

BitBtn. При создании текстового редактора мы применим компонент BitBtn (вкладка Addition). Этот компонент позволяет создавать (с помощью свойства Kind) кнопки, широко используемые в приложениях, рис. 3.4.

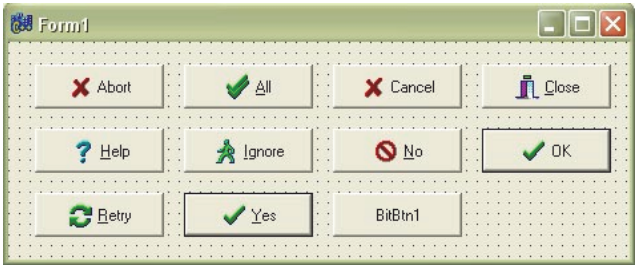


Рис. 3.4. Форма с компонентом BitBtn с различным свойством Kind.

(В нашем примере этот компонент применен в основном для ознакомительных целей.)

3.2. Создание визуальной оболочки текстового редактора

Окно формы имеет вид, показанный на рис. 3.5.

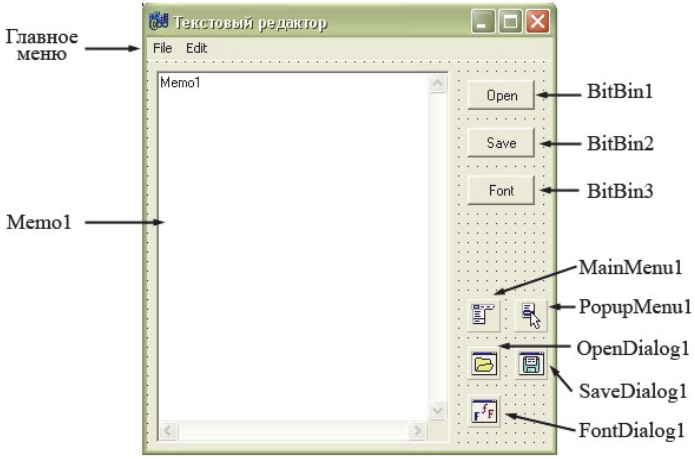


Рис. 3.5. Окно текстового редактора.

В приложении задействованы следующие компоненты: Memo; MainMenu; три кнопки компонента BitBtn; два диалога (OpenDialog и SaveDialog) для обращения к дисковым накопителям при открытии и сохранении файлов; диалог Font для настройки шрифтов; PopupMenu для создания контекстного меню (для того чтобы оно работало необходимо в его свойстве PopupMenu выбрать значение PopupMenu1).

Добьемся, чтобы при попадании курсора на кнопки Open, Save, Font менялся его тип. Для этого выделим мышью одновременно три кнопки и в появившемся объекте 3itemsSelected в инспекторе объектов изменим свойство Cursor на значение crHandPoint («рука»).

Главное меню настроим с помощью дизайнера меню, записав пункты и команды меню в соответствии с рис. 3.6.

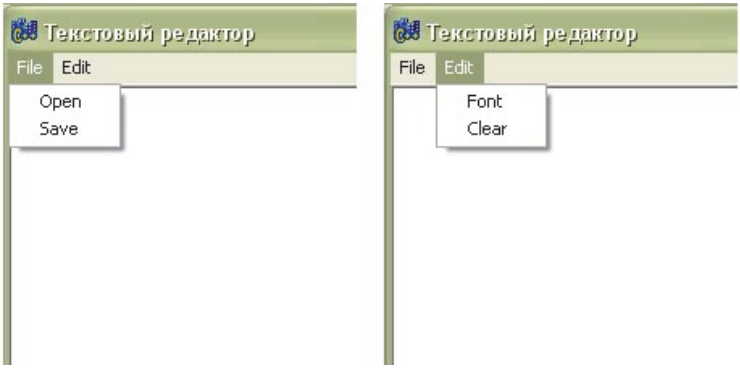


Рис. 3.6. Пункты и команды в меню.

С помощью диалоговых окон OpenDialog1 и SaveDialog1 открываются и закрываются файлы для редактирования. Но типов файлов много, а использовать можно только определенные. Для создания фильтра, пропускающих именно определенный тип файлов, служит свойство Filter, имеющийся в обоих диалогах. При нажатии кнопки с тремя точками правее свойства Filter открывается окно Filter Editor (рис. 3.7).

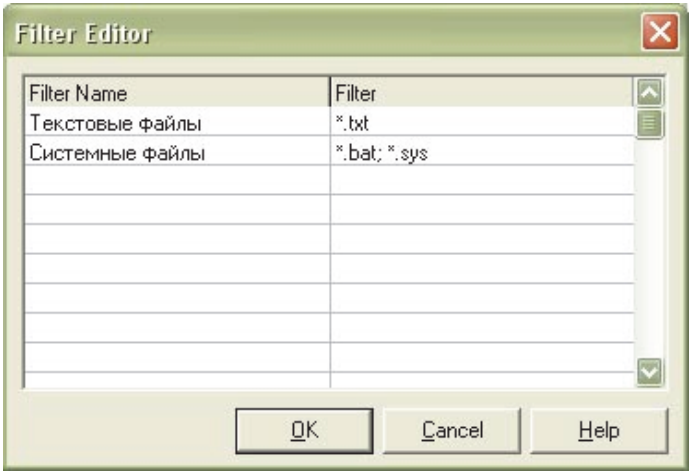


Рис. 3.7. Filter Editor.

В нем нужно задать нужные расширения файлов. При вызове диалоговых окон из выпадающего списка можно будет выбирать файлы только с заданным расширением (для корректной работы приложения оба диалога должны иметь одинаковую направленность).

Тип шрифта, его размер, цвет, стиль и т. д. выбираются с помощью свойства Font компонента FontDialog1.

Контекстное меню PopupMenu настраивается также как и главное меню MainMenu. Создадим две команды, рис. 3.8.

Рис. 3.8. Настройка контекстного меню PopupMenu.

Также в свойство PopupMenu главной формы добавим название PopupMenu1.

Настроив все компоненты, мы получим визуальную оболочку программы. Чтобы она заработала нужны обработчики событий.

3.3. Обработчики событий

В листинге 3.1 представлены обработчики события OnClick для четырех команд Open, Save, Font и Clear. Так как эти команды дублируются кнопками компонента BitBtn и контекстного меню, то обработчики должны быть заданы и для них. Делается это путем активизации нужной кнопки (например, Save в группе BitBtn) и выбора в раскрывающемся списке события OnClick строчки Save1Click.

Листинг 3.1.

```
0    #include <vcl.h>
1    #pragma hdrstop
2
3    #include "edit1.h"
4    //---------------------------------------------------------------------------
5    #pragma package(smart_init)
6    #pragma resource "*.dfm"
7    TForm1 *Form1;
8    //---------------------------------------------------------------------------
9    __fastcall TForm1::TForm1(TComponent* Owner)
10       : TForm(Owner)
11   {
12   }
13   //---------------------------------------------------------------------------
14   void __fastcall TForm1::Open1Click(TObject *Sender)
15   {
16   if (OpenDialog1->Execute())
17       Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
18   }
19   //---------------------------------------------------------------------------
20
21   void __fastcall TForm1::Save1Click(TObject *Sender)
22   {
```

```
23    if (SaveDialog1->Execute())
24          Memo1->Lines->SaveToFile(SaveDialog1->FileName);
25    }
26    //---------------------------------------------------------------------------
27
28    void __fastcall TForm1::Font1Click(TObject *Sender)
29    {
30    if (FontDialog1->Execute())
31       Memo1->Font=FontDialog1->Font;
32    }
33    //---------------------------------------------------------------------------
34
35    void __fastcall TForm1::Clear1Click(TObject *Sender)
36    {
37    Memo1->Clear();
38    }
39    //---------------------------------------------------------------------------
```

3.4. Создание панели инструментов и иконок пунктов главного меню

Для удобства работы с программой создадим панель инструментов, куда поместим кнопки для открытия и сохранения файлов, а также поместим на эти кнопки соответствующие иконки. Этими же иконками снабдим кнопки команд в пункте File главного меню. Для этого поместим на форму компонент ToolBar (вкладка Win32), а также – компонент ImageList (вкладка Win32) для загрузки и хранения рисунков иконок в панели инструментов и в пункте File главного меню.

Чтобы поместить кнопку на панели инструментов ToolBar, надо щелкнуть по ней правой кнопкой мыши и в открывшемся окне, рис. 3.9, надо нажать кнопку New Button (эту операцию надо повторить дважды для помещения кнопок Open и Save). Обработчики для этих кнопок надо задать по аналогии с изложенным в пункте 3.3.

Рис. 3.9. Размещение кнопки на панель инструментов.

Снабдим кнопки главного меню и панели инструментов рисунками иконок: двойной щелчок левой кнопкой мыши по компоненту ImageList и открывается окно, рис. 3.10.

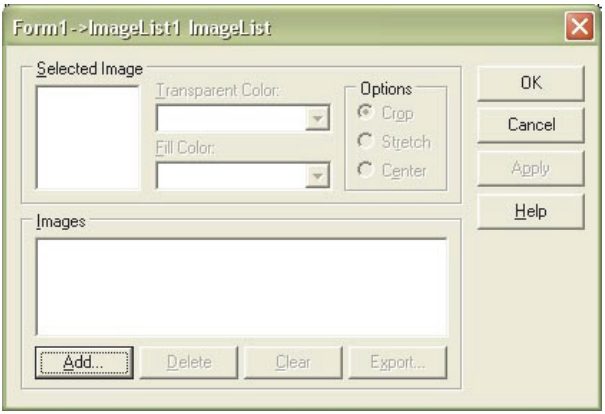


Рис. 3.10. Создание иконок кнопок.

При щелчке по кнопке Add открывается окно выбора файлов, рис. 3.11.

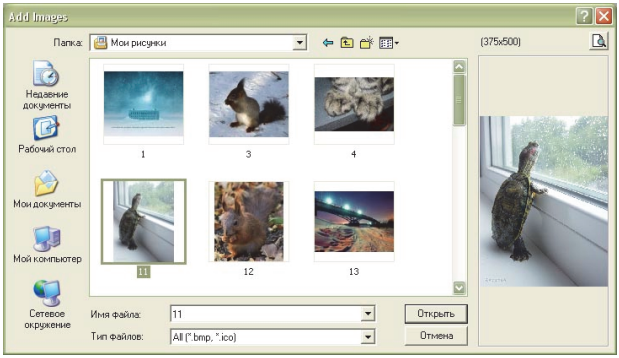


Рис. 3.11. Окно выборов файлов.

В каталоге C:\Program Files\Common Files\Borland Shared\Image\Buttons\ выберем файлы flilesopen ( ) и filessave ( ) и загрузим их в Form1 -> ImageList1. (Для обеспечения доступа к рисункам ImageList компоненты ToolBar и MainMenu должны иметь в свойстве Images имя компонента ImageList1, которое выбирается из списка в свойстве Images инспектора объектов.)

Окончательный вид окна программы (кнопка Open в группе BitBtn снабжена значком в «учебных целях») представлен на рис. 3.12.

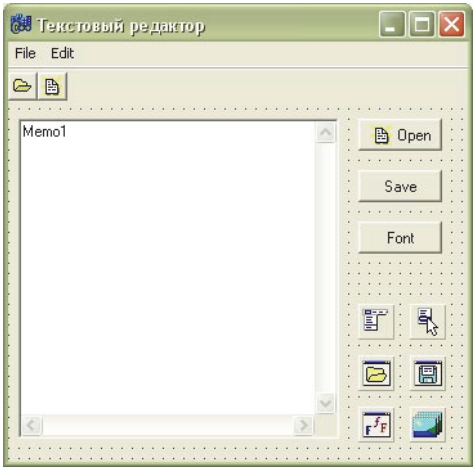


Рис. 3.12. Окно формы текстового редактора.

4. Оптимизация параметров прогностических моделей

4.1. Описание моделей

В качестве прогностических моделей используются дифференциальные уравнения первого и второго порядка, связывающее внешнее воздействие в виде ежедневных суммарных осадков на водосбор с расходом воды:

$$\frac{dQ}{dt} = -\frac{1}{k\tau}Q + \frac{\dot{X}}{\tau};$$

$$\tau_2 \frac{d^2Q}{dt^2} + \left(\frac{\tau_2}{k\tau_1} + 1\right)\frac{dQ}{dt} + \frac{1}{k\tau_1}Q = \frac{1}{\tau_1}\dot{X},$$

где Q – расход воды в замыкающем створе бассейна; $\dot{X}$ – интенсивность осадков; k – коэффициент стока; τ_1 – время добегания поверхностного стока; τ_2 – время добегания подземного стока; t – время (при $\tau_2 = 0$ $\tau_1 \equiv \tau$).

Этой моделью учитываются два основных фактора формирования стока: время релаксации (τ_1, τ_2) или запаздывания (добегания – в гидрологической терминологии) и коэффициент стока (k), который отвечает за «потери» выпавших осадков, например на испарение.

Самый простой численный метод, реализующий эти уравнения (метод Эйлера):

$$Q_{i+1} = Q_i - (Q_i / k - \dot{X}_i)/\tau;$$

$$\tau_2 \frac{Q_{i+1} - 2Q_i + Q_{i-1}}{\Delta t^2} + (\frac{\tau_2}{k\tau_1} + 1)(\frac{Q_{i+1} - Q_i}{\Delta t}) + \frac{1}{k\tau_1}Q_i = \frac{\dot{X}_i}{\tau_1},$$

где параметр i индексирует используемые интервалы времени (в нашем примере сутки $\Delta t = 1$).

4.2. Технология прогноза

В качестве внешнего воздействия задаются среднесуточные суммарные осадки на водосбор. Имеющиеся Интернет ресурсы (http://ready.arl.noaa.gov) позволяют использовать восьмисуточные прогнозы осадков практически для любой точки земного шара.

Для определения численных значений параметров τ_1, τ_2 и k будем использовать процесс оптимизации, применяя в качестве критериев отношение S/σ_Δ и относительное число оправдавшихся прогнозов P:

$$S = \sqrt{\sum_1^n \frac{(Q_i - Q_{Pi})^2}{n}};$$

$$\sigma_\Delta = \sqrt{\frac{\sum_1^n \left((Q_{i+1} - Q_i) - \frac{\sum (Q_{i+1} - Q_i)}{n} \right)^2}{n}}.$$

Оправдавшимся считаются прогнозы, такие что $(Q_i - Q_{Pi}) < \delta_{доп} = 0{,}674\sigma_\Delta$ (здесь Q_{Pi} – прогнозные значения расходов). Погрешности определения S и σ_Δ зависят и от числа членов ряда n, поэтому учитываются следующие условия применимости методик прогнозирования: при $n \le 15$ $S/\sigma_\Delta \le 0.70$; при $15 < n < 25$ $S/\sigma_\Delta \le 0.75$; при $n \ge 25$ $S/\sigma_\Delta \le 0.80$. Обеспеченность методики определяется из выражения $P = \frac{n'}{n} 100$, где n' – число оправдавшихся прогнозов.

Оптимальными считаются значения τ_1, τ_2 и k при которых отношение S/σ_Δ минимально, а значение P максимально.

Имея (например, за последние две–три недели) данные по осадкам и расходам, можно циклически «протащить» модель по параметрам τ_1, τ_2 и k (меняя τ_1, τ_2 и k в различных пределах) и выбрать их оптимальные значения. С использованием этих значений параметров, делаем прогноз хода расходов на ближайшие восемь суток (тут может быть много вариантов и нюансов, на которых мы не останавливаемся).

4.3. Программа на языке C++ в приложении C++ Builder

Форма приложения имеет вид, представленный на рис. 4.1.

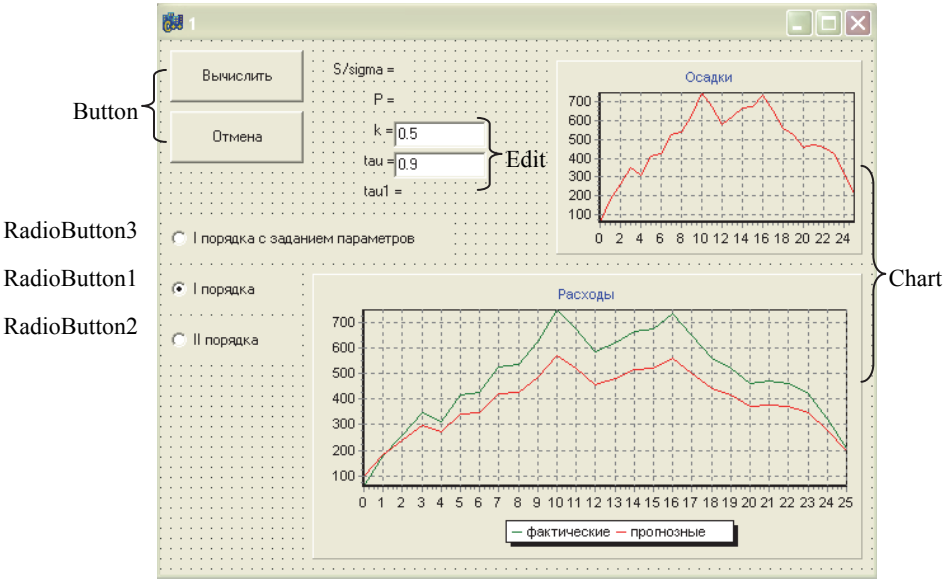


Рис. 4.1. Форма приложения
(при использовании модели второго порядка tau ≡ τ_1; tau1 ≡ τ_2).

Компоненты Edit становятся активными при выборе модели прогноза с заданием параметров (на форме указаны значения по умолчанию). Компоненты Chart содержат в режиме редактирования выдуманные графики, вид которых можно настроить в свойствах комопнента. Свойства редактируются в контекстном меню компонента (щелчок правой кнопки по компоненту).

Обработчики событий представлены в листинге (в головной файл подключить: # include <math.h>).

Листинг 4.1.

```
0     //---------------------------------------------------------------------------
1
2     #include <vcl.h>
3     #pragma hdrstop
4
```

```
5   #include "Kurs.h" //подключение заголовочного файла
6   #include <math.h>
7   //---------------------------------------------------------------------------
8   #pragma package(smart_init)
9   #pragma resource "*.dfm"
10  TForm1 *Form1;
11  //---------------------------------------------------------------------------
12  __fastcall TForm1::TForm1(TComponent* Owner)
13        : TForm(Owner)
14  {
15  }
16  //---------------------------------------------------------------------------
17  void __fastcall TForm1::Button1Click(TObject *Sender)
18  {
19  //с оптимизацией коэффициента стока и времени добегания
20
21  //объявление переменных
22  int t, i, j; // счетчики для циклов
23  int dt = 1; // шаг интегрирования
24  float tau, Kst, P, tau1; // время добегания, коэффициент стока, число
    оправдавшихся прогнозов
25  //массив интенсивности осадков
26  float X[15] = {5.10,15.5,33.6,39.2,28.6,10.1,1.00,1.00,4.50,18.2,
    20.3,8.90,3.50,2.70,15.8};
27  //массив фактических расходов
28  float Q[15] = {2.23,4.15,10.0,16.1,20.2,20.2,16.8,8.65,1.23,1.51,
    3.00,6.48,10.2,9.45,17.2};
29  float Qp[15]; // массив прогнозных расходов воды
30  float Sum, S, delta=0, deltasr, Sums=0, Sigma, Kr, dop; // переменные,
    необходимые для оценки оправдываемости прогноза и эффектив-
    ности методики
31  float Kr_o=100, P_o=0, Kst_o, tau_o, tau1_o; // оптимальные значе-
    ния параметров
32
33  Series1->Clear();
34  Series2->Clear();
35  Series3->Clear();
36  for (int j=0; j<15; j++)
37  {
38  Series1->AddXY(j, X[j],j,clBlue);
39  }
40
```

```
41  //расчет допустимой погрешности
42  for (t=0; t<15; t++)
43  { delta +=(Q[t]-Q[t+1]);}
44  deltasr = delta/14;
45  for (t=0; t<15; t++)
46  { Sums += ((Q[t]-Q[t+1])-deltasr)*((Q[t]-Q[t+1])-deltasr);}
47  Sigma = sqrt(Sums/14);
48  dop = 0.674*Sigma;
49
50
51
52  if  (RadioButton1->Checked)
53  {
54  Label9->Visible = false;
55  Label10->Visible = false;
56  Edit1->Visible = false;
57  Edit2->Visible = false;
58  //прогноз расходов воды с разными параметрами (коэффициентом
    стока и временем добегания)
59  for (tau=1; tau<6; tau+=0.1)
60  {
61  for (Kst=0.1; Kst<1.1; Kst+=0.1)
62  {
63   float opr=0;
64   for (t=1; t<15; t++)
65    {
66    if (t==1)
67      Qp[t]=Q[t-1]+dt*(X[t-1]/tau-Q[t-1]/(Kst*tau));
68    else
69    Qp[t]=Qp[t-1]+dt*(X[t-1]/tau-Qp[t-1] /(Kst*tau));
70    Sum += (Q[t]-Qp[t])*(Q[t]-Qp[t]);
71    if (fabs(Q[t]-Qp[t])<=dop)
72    opr+=1;
73    }
74   //критерий эффективности методики
75   S = sqrt(Sum/(14));
76   Sum = 0;
77   Kr = S/Sigma;
78   P = (opr/14)*100;
79   //оптимальные коэф-т стока и время добегания
80   if ((Kr<=Kr_o) && (P>=P_o))
81   { Kr_o=Kr; P_o=P; Kst_o=Kst; tau_o=tau; }
```

```
82    }
83    }
84    Label5->Caption =  FloatToStrF(Kr_o, ffGeneral, 3, 2);
85    Label6->Caption =  FloatToStrF(P_o, ffGeneral, 3, 2);
86    Label7->Caption =  FloatToStrF(Kst_o, ffGeneral, 3, 2);
87    Label8->Caption =  FloatToStrF(tau_o, ffGeneral, 3, 2);
88
89    for (t=1; t<15; t++)
90      {
91      if (t==1)
92        Qp[t]=Q[t-1]+dt*(X[t-1]/tau_o-Q[t-1]/(Kst_o*tau_o));
93      else
94       Qp[t]=Qp[t-1]+dt*(X[t-1]/tau_o-Qp[t-1]/(Kst_o*tau_o));
95    }
96    }
97
98    if  (RadioButton2->Checked)
99    {
100   Label9->Visible = true;
101   Edit1->Visible = false;
102   Edit2->Visible = false;
103   //прогноз расходов воды с разными параметрами
104   for (tau1=1; tau1<50; tau1+=1)
105   {
106   for (tau=1; tau<16; tau+=1)
107   {
108   for (Kst=0.1; Kst<1.1; Kst+=0.1)
109   {
110    float opr=0;
111    for (t=2; t<15; t++)
112     {
113     if (t==2)
114        Qp[t]=(X[t-1]/tau - Q[t-1]/(Kst*tau) + Q[t-1] *(2 *tau1 +(tau1
      +Kst*tau1)/(Kst*tau)) - tau1*Q[t-2]) / (tau1+(tau1+Kst*tau)/(Kst*tau));
115     if (t==3)
116      Qp[t]=(X[t-1]/tau - Qp[t-1]/(Kst*tau) + Qp[t-1] *(2 *tau1 +(tau1
      +Kst*tau1)/(Kst*tau))- tau1*Q[t-2])/(tau1+(tau1+Kst*tau)/(Kst*tau));
117     if (t>3)
118        Qp[t]=(X[t-1]/tau - Qp[t-1]/(Kst*tau) + Qp[t-
      1]*(2*tau1+(tau1+Kst*tau1)/(Kst*tau)) - tau1*Qp[t-
      2])/(tau1+(tau1+Kst*tau)/(Kst*tau));
119     Sum += (Q[t]-Qp[t])*(Q[t]-Qp[t]);
```

```
120     if (fabs(Q[t]-Qp[t])<=dop)
121     opr+=1;
122     }
123    //критерий эффективности методики
124    S = sqrt(Sum/(14));
125    Sum = 0;
126    Kr = S/Sigma;
127    P = (opr/14)*100;
128    //оптимальные коэф-т стока и время добегания
129    if ((Kr<=Kr_o) && (P>=P_o))
130    { Kr_o=Kr; P_o=P; Kst_o=Kst; tau_o=tau; tau1_o=tau1;}
131   }
132   }
133   }
134   Label5->Caption =  FloatToStrF(Kr_o, ffGeneral, 3, 2);
135   Label6->Caption =  FloatToStrF(P_o, ffGeneral, 3, 2);
136   Label7->Caption =  FloatToStrF(Kst_o, ffGeneral, 3, 2);
137   Label8->Caption =  FloatToStrF(tau_o, ffGeneral, 4, 2);
138   Label10->Visible = true;
139   Label10->Caption =  FloatToStrF(tau1_o, ffGeneral, 4, 2);
140
141   for (t=2; t<15; t++)
142     {
143     if (t==2)
144        Qp[t]=(X[t-1]/tau_o  -  Q[t-1]/(Kst_o*tau_o)  +  Q[t-1]*(2*tau1_o
      +(tau1_o + Kst_o*tau1_o) / (Kst_o * tau_o)) - tau1_o * Q[t-2]) /(tau1_o
      + (tau1_o + Kst_o*tau_o) / (Kst_o*tau_o));
145     if (t==3)
146        Qp[t]=(X[t-1]/tau_o - Qp[t-1]/(Kst_o*tau_o) + Qp[t-1] *(2*tau1_o+
      (tau1_o + Kst_o *tau1_o) / (Kst_o *tau_o)) - tau1_o * Q[t-2]) /(tau1_o
      +(tau1_o + Kst_o*tau_o)/(Kst_o*tau_o));
147     if (t>3)
148        Qp[t]=(X[t-1]/tau_o - Qp[t-1]/(Kst_o*tau_o) + Qp[t-1] *(2*tau1_o
      +(tau1_o + Kst_o * tau1_o) /(Kst_o *tau_o)) -tau1_o * Qp[t-2]) /(tau1_o
      +(tau1_o + Kst_o*tau_o)/(Kst_o*tau_o));
149     }
150   }
151
152
153   if  (RadioButton3->Checked)
154   {
155   float k, ta;
```

```
156  Label9->Visible = false;
157  Label10->Visible = false;
158  Edit1->Visible = true;
159  Edit2->Visible = true;
160
161  k =  StrToFloat(Edit1->Text);
162  ta =  StrToFloat(Edit2->Text);
163
164
165  //прогноз расходов воды с заданными параметрами
166   float opr=0;
167   for (t=1; t<15; t++)
168    {
169    if (t==1)
170      Qp[t]=Q[t-1]+dt*(X[t-1]/ta-Q[t-1]/(k*ta));
171    else
172    Qp[t]=Qp[t-1]+dt*(X[t-1]/ta-Qp[t-1] /(k*ta));
173    Sum += (Q[t]-Qp[t])*(Q[t]-Qp[t]);
174    if (fabs(Q[t]-Qp[t])<=dop)
175    opr+=1;
176    }
177   //критерий эффективности методики
178   S = sqrt(Sum/(14));
179   Sum = 0;
180   Kr = S/Sigma;
181   P = (opr/14)*100;
182
183  Label5->Caption =  FloatToStrF(Kr, ffGeneral, 3, 2);
184  Label6->Caption =  FloatToStrF(P, ffGeneral, 3, 2);
185
186  }
187
188  //изображение гидрографа фактического и спрогнозированного
189  Series3->Clear();
190  for (int j=0; j<15; j++)
191  {
192  Series2->AddXY(j, Q[j],j,clGreen);
193  Series3->AddXY(j, Qp[j],j,clRed);
194  }
195
196  }
197  //---------------------------------------------------------------------------
```

```
198  void __fastcall TForm1::Button2Click(TObject *Sender)
199  {
200  Form1->Close();
201  }
202  //---------------------------------------------------------------------------
```

В основном действия программы понятны из комментариев, поясним только некоторые строки.

Для построения графиков используется компонент Chart и указатели Series. Алгоритм обращения к компоненту представлен в строках 37–39: вызывается метод AddXY, который в качестве первого параметра принимает значения по оси *x*, второй параметр – значения по оси *y*, третий параметр – подписи оси *x*, четвертый параметр – цвет графика. В представленных строках отображается график хода осадков. В строках 190–194 показано построение фактического и прогнозного гидрографов. Следует отметить, что для отображения графиков только под выбранный вариант прогноза необходимо очищать содержимое компонента Chart по указателям Series, это проделано в строках 33–35, 189. Выбор варианта прогноза организован с помощью компонента RadioButton, в листинге – это строки 52, 98, 153.

С 38-й строки начинается блок программы, в котором происходит определение оптимальных значений параметров модели первого порядка. Для этого используется три цикла, два из которых являются вложенными друг в друга. Первый цикл перебирает значения времени добегания (tau) с шагом 0.1, второй – коэффициента стока (Kst) с тем же шагом, третий цикл организован для расчета прогнозных значений на каждый момент времени. В строках 67 и 69 происходит вычисление прогнозных расходов воды на каждом витке циклов. Отличие этих строк заключается в том, что в строке 67 рассчитывается прогнозный расход в момент времени, равный единице; при этом берется начальное (элемент массива с индексом ноль) значение расхода из фактического ряда. В строке 69 при расчетах используются прогнозные расходы воды за предыдущий момент времени. В строке 70 рассчитывается сумма погрешностей прогноза. В условном операторе в строках 71, 72 подсчитывается число оправдавшихся прогнозов. С 75-й по 78-ю вычисляются критерии эффективности применяемой методики. И в строках 80, 81 выбирается наилучший результат, т. е. оптимальные значения

параметров, при которых S/σ_Δ стремиться к минимальному, а P – к максимальному значениям. В строке 31 эти критерии были инициализированы фиксированными значениями. С 89-й по 95-ю строку дается прогноз с оптимальными значениями параметров.

С 98-й строки начинается блок программы, в котором происходит прогнозирование расходов воды по модели второго порядка. Производится оптимизация трех параметров – времени добегания поверхностного стока, времени добегания подземного стока и коэффициента стока (строка 104): добавлен внешний цикл по tau1 с шагом 1. Так как рассматривается модель второго порядка, то в качестве начального условия берутся фактические расходы в нулевой и в первый моменты времени (условные операторы в строках 113, 115).

С 153-й строки начинается часть программы, в которой происходит прогнозирование расходов воды с использованием конкретных значений коэффициента стока и времени добегания, которые задаются в появляющихся компонентах Edit (строки 158, 159).

Окна программы представлены на рис. 4.2 – 4.4.

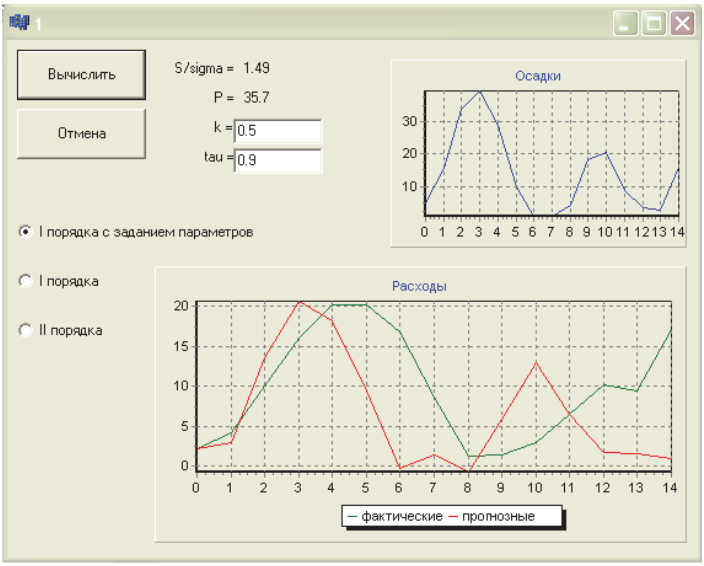


Рис. 4.2. Результат работы программы при выборе прогнозной модели I порядка с заданием параметров.

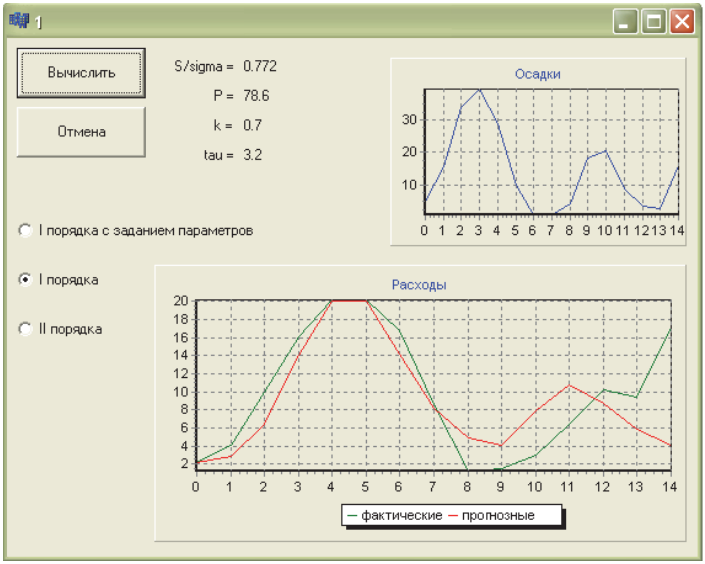


Рис. 4.3. Результат работы программы при выборе прогнозной модели I порядка с оптимизацией параметров.

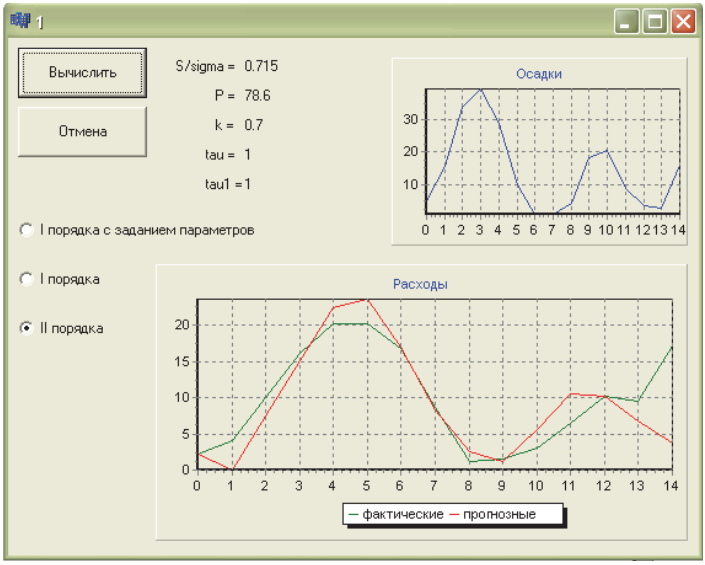


Рис. 4.4. Результат работы программы при выборе прогнозной модели II порядка с оптимизацией параметров.

5. Решение уравнения Фоккера–Планка–Колмогорова (ФПК)

5.1. Описание модели

Уравнение Фоккера–Планка–Колмогорова (ФПК) имеет следующий вид:

$$\frac{\partial p(Q,t)}{\partial t} = -\frac{\partial (A(Q,t)p(Q,t))}{\partial Q} + 0.5\frac{\partial^2 (B(Q,t)p(Q,t))}{\partial Q^2}, \quad (5.1)$$

где $p(Q, t)$ – плотность вероятности; Q – расход воды (слой стока, модуль); t – время.

Коэффициенты сноса $A(Q, t)$ и диффузии $B(Q, t)$ выражаются формулами

$$A(Q,t) = -(\overline{c} - 0.5G_{\widetilde{c}})Q - 0.5G_{\widetilde{c}\widetilde{N}} + \overline{N}; \quad (5.2)$$

$$B(Q,t) = G_{\widetilde{c}}Q^2 - G_{\widetilde{c}\widetilde{N}}Q + G_{\widetilde{N}}, \quad (5.3)$$

где $G_{\widetilde{c}}$, $G_{\widetilde{N}}$ и $G_{\widetilde{c}\widetilde{N}}$ – интенсивности и взаимная интенсивность шумов, а $\overline{c}$ и $\overline{N}$ – математические ожидания (они могут быть функциями времени).

Для численного решения уравнения (5.1) используем следующую конечно-разностную аппроксимацию (она имеет очевидные недостатки, но для учебного примера на них можно «закрыть глаза»):

$$\frac{(p_j^{i+1} - p_j^i)}{\Delta t} = -\frac{(A_j^i p_j^i - A_{j-1}^i p_{j-1}^i)}{\Delta Q} + \\ + 0.5\frac{(B_{j+1}^i p_{j+1}^i - 2B_j^i p_j^i + B_{j-1}^i p_{j-1}^i)}{(\Delta Q)^2}, \quad (5.3)$$

где i и j – номера расчетных шагов по времени и расходу соответственно. Расчетная сетка и логика вычисления показаны на рис. 5.1.

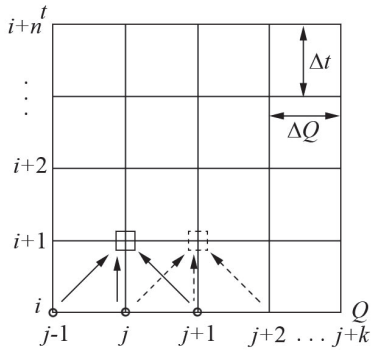


Рис. 5.1. Иллюстрация алгоритма вычисления конечно-разностной аппроксимации (5.3).

На каждом временном слое (i+1, i+2 и т. д.) происходит вычисление значений плотности вероятности в узлах сетки с координатами (i+1, j; i+1, j+1; …). На границах (для рис. 5.1 это j–1 и j+k) должны быть заданы граничные условия $F(p, \partial p / \partial Q) = 0$, а в начальный момент времени t_0 (для рис. 5.1 в слое i) – начальные условия $p(Q, t_0)$.

5.2. Задание внешнего воздействия

Информация о внешнем воздействии $\overline{\dot{X}}(t)$ приведена в табл. 5.1. Значения параметров $G_{\tilde{c}}$, $G_{\tilde{N}}$, $G_{\tilde{c}\tilde{N}}$, k, τ должны находиться в некотором интервале значений, при которых сохраняется устойчивость расчетов: $G_{\tilde{c}}$ – $(1...4)\cdot 10^{-4}$; $G_{\tilde{N}}$ – 4…7; $G_{\tilde{c}\tilde{N}}$ – –0.005 …0.005; k – 0.49…0.51; τ – 1…2. Граничными условиями будут нулевые значения плотности вероятности на концах интервала: $p(Q = 97) = 0$; $p(Q = 107) = 0$.

Таблица 5.1

Ход осадков, поступающих на водосбор

Время (сут)	0	1	2	3	4	5	6	7	8	9	10
Осадки, $м^3/с$	201	203	205	207	208	207	205	203	201	200	200

В начальный момент времени (t = 0) значения плотности вероятности приведены в табл. 5.2.

Таблица 5.2

Значения плотности вероятности в начальный момент времени

Q, м^3/с	97/98	98/99	99/100	100/101	101/102	102/103	103/104	104/105	105/106	106/107
p, %/(м^3/с)	0	8	20	18	16	14	12	8	4	0

Шаги по времени составляют Δt = 0.1 сут, а по расходу $\Delta Q = 1$ м^3/с (приведенные выше значения характеристик шумов $G_{\tilde{c}}$, $G_{\tilde{N}}$ и $G_{\tilde{c}\tilde{N}}$ соответствуют внесистемной размерности времени в сутках, т. е. при вычислении $\bar{c} = 1/k\tau$ и подстановке в алгоритм Δt значения τ и Δt переводить в секунды не нужно).

5.3. Программа на языке C++ в приложении C++ Builder

Форма приложения представлена на рис. 5.2.

Рис. 5.2. Форма приложения.

На форму помещено 5 компонентов Edit для ввода параметров модели. Компоненты Chart служат для вывода гистограмм плотности вероятности в каждый момент времени.

Обработчики событий представлены в листинге 5.1.

Листинг 5.1.

```
0   //---------------------------------------------------------------------------
1
2   #include <vcl.h>
3   # include <math.h>
4   #pragma hdrstop
5
6   #include "FPK.h"
7   //---------------------------------------------------------------------------
8   #pragma package(smart_init)
9   #pragma resource "*.dfm"
10  TForm1 *Form1;
11  //---------------------------------------------------------------------------
12  __fastcall TForm1::TForm1(TComponent* Owner)
13        : TForm(Owner)
14  {
15  }
16  //---------------------------------------------------------------------------
17
18  void __fastcall TForm1::Button1Click(TObject *Sender)
19  {
20  double Gc, GN, GcN; //интенсивности шумов
21  float k, t; // коэффициент стока и время добегания
22  float p[100][10]; // массив для плотности вероятности
23  const float dt = 0.1; // шаг по времени
24  const float dQ = 1; // шаг по расходу воды
25  // интенсивность осадков
26  float X[10] = {201, 203, 205, 207, 208, 207, 205, 203, 201, 200};
27  //значения расходов воды
28  float Q[10] = {97, 98, 99, 100, 101, 102, 103, 104, 105, 106};
29  float A[100][10]; //массив для коэффициента сноса
30  float B[100][10]; // массив для коэффициента диффузии
31
32  Gc = StrToFloat(Edit1->Text);
33  GN = StrToFloat(Edit2->Text);
34  GcN = StrToFloat(Edit3->Text);
35  k = StrToFloat(Edit4->Text);
```

```
36  t = StrToFloat(Edit5->Text);
37
38  Series1->Clear();
39  Series2->Clear();
40  Series3->Clear();
41  Series4->Clear();
42  Series5->Clear();
43  Series6->Clear();
44  Series7->Clear();
45  Series8->Clear();
46  Series9->Clear();
47
48  //проверка правильности ввода данных
49  if (((Gc<0.0001)||(Gc>0.0004)) || ((GN<4)||(GN>7)) || ((GcN<-
    0.005)||(GcN>0.005)) ||((k<0.49)||(k>0.51)) || ((t<1)||(t>2)))
50    ShowMessage("Проверь правильность ввода данных");
51
52    float c;
53  c = 1/(k*t);
54  for (int i=0; i<100; i++)
55  p[i][0]=0;
56  // плотности вероятности в начальный момент времени
57  p[0][0] = 0; p[0][1] = 8; p[0][2] = 20; p[0][3] = 18;
58  p[0][4] = 16; p[0][5] = 14; p[0][6] = 12; p[0][7] = 8;
59  p[0][8] = 4; p[0][9] = 0;
60  for (int i=0; i<99; i++)
61  {
62  for (int j=1; j<10; j++)
63  {
64  A[i][j] = (-1)*(c-0.5*Gc)*Q[j] - 0.5*GcN + X[int(i/24)]/t;
65  A[i][j-1] = (-1)*(c-0.5*Gc)*Q[j-1] - 0.5*GcN + X[int(i/24)]/t;
66  B[i][j-1] = (Gc*pow(Q[j-1], 2) - GcN*Q[j-1] + GN);
67  B[i][j] = (Gc*pow(Q[j], 2) - GcN*Q[j] + GN);
68  B[i][j+1] = (Gc*pow(Q[j+1], 2) - GcN*Q[j+1] + GN);
69  //вычисление плотности вероятности
70  p[i+1][j]=p[i][j]+dt*(((-1)*(A[i][j]*p[i][j]-A[i][j-1]*p[i][j-
    1]))/dQ+0.5*((B[i][j+1]*p[i][j+1]-2*B[i][j]*p[i][j]+B[i][j-1]*
    p[i][j-1])/dQ*dQ));
71  if (p[i][j] < 0) p[i][j] = 0;
72   }
73   }
74  //построение плотности вероятности через сутки
```

```
75    for (int j=0; j<10; j++)
76    {
77    Series1->AddXY(j, p[0][j],j,clBlue); //нулевые сутки
78    Series2->AddXY(j, p[10][j],j,clBlue); //первые сутки
79    Series3->AddXY(j, p[20][j],j,clBlue); //вторые сутки и т.д.
80    Series4->AddXY(j, p[30][j],j,clBlue);
81    Series5->AddXY(j, p[40][j],j,clBlue);
82    Series6->AddXY(j, p[50][j],j,clBlue);
83    Series7->AddXY(j, p[60][j],j,clBlue);
84    Series8->AddXY(j, p[70][j],j,clBlue);
85    Series9->AddXY(j, p[80][j],j,clBlue); //восьмые сутки
86    }
87    }
88    //---------------------------------------------------------------------------
89
90    void __fastcall TForm1::Button2Click(TObject *Sender)
91    {
92    Form1->Close();
93    }
```

Сделаем некоторые пояснения. В строке 3 объявлено, что будет подключена стандартная библиотека math.h, отвечающая за реализацию математических функций. В строках 66–68 вызывается функция возведения в степень pow с параметрами: первый параметр возводится в степень, указанную во втором параметре. Данная функция работает со значениями вещественного типа, поэтому элементы массивов осадков и расходов воды, объявленных в строках 26 и 28 имеют вещественный тип **float**. В строках 29, 30 объявлены двумерные массивы под значения коэффициентов сноса и диффузии. Эти коэффициенты рассчитываются для каждого временного слоя. Алгоритм расчета плотности вероятности представлен в строках 60–73.

В строках 75–86 происходит построение плотности вероятности для временных слоев кратных десяти (т. е. через сутки) в компонентах Chart путем применения метода AddXY через указатели Series. Для того чтобы была возможность многократно работать с приложением, в строках 38–46 происходит очистка графических компонент.

Окно программы представлено на рис. 5.3.

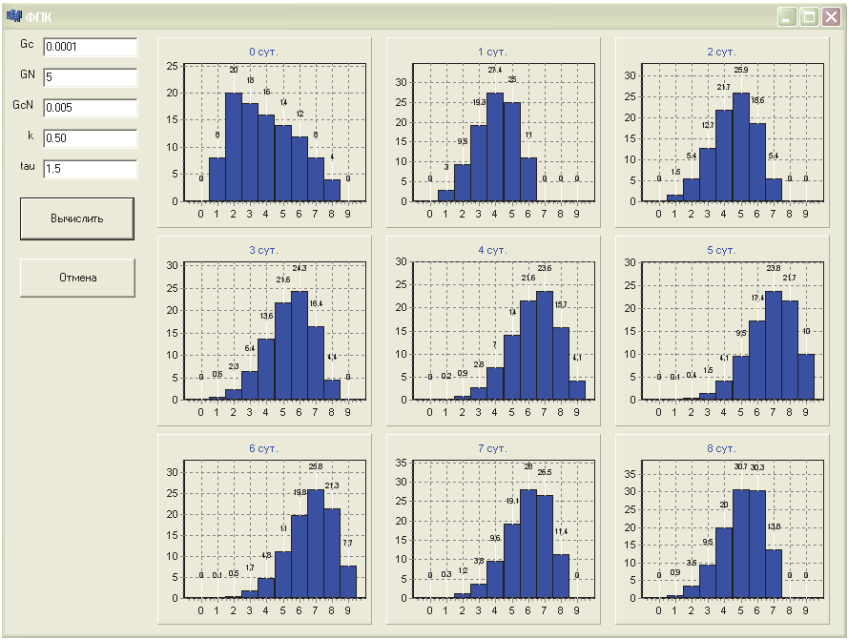


Рис. 5.3. Результат работы программы из листинга 5.1.

6. Неустойчивость решений системы дифференциальных уравнений для моментов вероятностных распределений

6.1. Описание модели

Уравнение ФПК можно аппроксимировать системой дифференциальных уравнений. Рассмотрим ее решение для двух первых начальных моментов:

$$\begin{aligned} dm_1/dt &= -(\bar{c} - 0{,}5G_{\tilde{c}})m_1 - 0{,}5G_{\tilde{c}\tilde{N}} + \bar{N}; \\ dm_2/dt &= -2(\bar{c} - G_{\tilde{c}})m_2 + 2\bar{N}m_1 - 3G_{\tilde{c}\tilde{N}}m_1 + G_{\tilde{N}}, \end{aligned} \tag{6.1}$$

где $c = 1/k\tau = \bar{c} + \tilde{c}$; $N = \dot{X}/\tau = \bar{N} + \tilde{N}$ (здесь $\bar{c}$, $\bar{N}$ – математические ожидания; $\tilde{c}$, $\tilde{N}$ – коррелированные друг с другом белые шумы с интенсивностями $G_{\tilde{c}}$, $G_{\tilde{N}}$ и взаимной интенсивностью $G_{\tilde{c}\tilde{N}}$); k – коэффициент стока; $\dot{X}$ – интенсивность осадков; τ – время релаксации речного бассейна, соответствующее радиусу автокорреляции стока – как правило, один год.

Заметим, что в системе (6.1) моменты k-го порядка не зависят от моментов порядка $k+1$ и выше (см. также (7.1)). Поэтому сначала можно решить уравнения для m_1. Затем, используя знание значений m_1 для каждого временного шага, решить уравнение для m_2. Для решения подобных уравнений обычно используют метод Рунге–Кутты. Рассмотрим алгоритм этого метода для первого уравнения системы (6.1). Запишем его в виде ($m_1 \equiv m$)

$$dm/dt = f(m,t) \tag{6.2}$$

с начальным условием

$$m(t_0) = m_0. \tag{6.3}$$

Пусть m_i приближенное значение искомого решения в точке t_i. Дальнейшие вычисления значения момента в точке $t_{i+1} = t_i + \Delta t$

производятся по формулам:

$$m_{i+1} = m_i + \Delta m_i;$$
$$\Delta m_i = \frac{1}{6}(k_1^{(i)} + 2k_2^{(i)} + 2k_3^{(i)} + k_4^{(i)}), \tag{6.4}$$

где

$$k_1^{(i)} = \Delta t f(t_i, m_i),$$
$$k_2^{(i)} = \Delta t f(t_i + \frac{\Delta t}{2}, m_i + \frac{k_1^{(i)}}{2}),$$
$$k_3^{(i)} = \Delta t f(t_i + \frac{\Delta t}{2}, m_i + \frac{k_2^{(i)}}{2}), \tag{6.5}$$
$$k_4^{(i)} = \Delta t f(t_i + \Delta t, m_i + k_3^{(i)}).$$

Для решения задачи необходимо задать начальные условия в виде: $m_1\big|_{t=0} = m_1(0)$, $m_2\big|_{t=0} = m_2(0)$, а также значения параметров k, τ, $G_{\tilde{c}}$, $G_{\tilde{c}\tilde{N}}$, $\overline{N}$, $G_{\tilde{N}}$ и шага по времени Δt.

6.2. Программа на языке C++ в приложении C++ Builder

Форма приложения имеет вид, представленный на рис. 6.1.

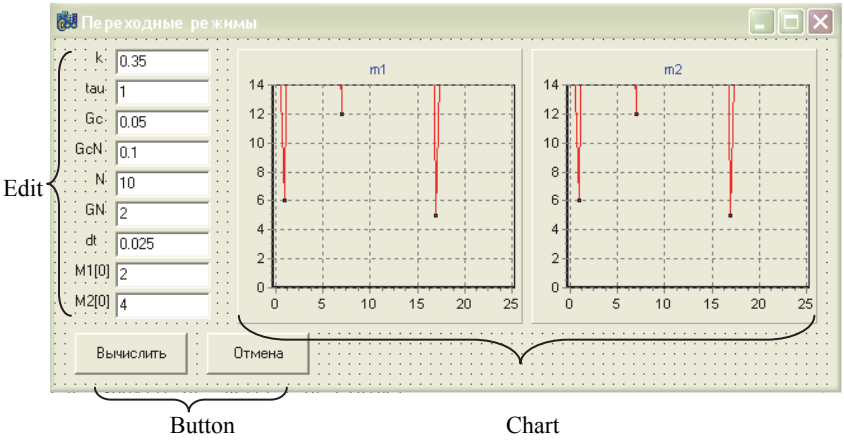


Рис. 6.1. Форма приложения.

Параметры и значения первого и второго моментов в начальный момент времени на форме вводятся в компоненты Edit. В компоненты Chart будут размещаться временны*е* графики для моментов.

Возможный вариант программы представлен в листинге 6.1.

Листинг 6.1.

```
0    //---------------------------------------------------------------------------
1
2    #include <vcl.h>
3    #pragma hdrstop
4
5    #include "resh_dif_ur.h"
6    //---------------------------------------------------------------------------
7    #pragma package(smart_init)
8    #pragma resource "*.dfm"
9    TForm1 *Form1;
10   //---------------------------------------------------------------------------
11   __fastcall TForm1::TForm1(TComponent* Owner)
12         : TForm(Owner)
13   {
14   }
15
16   float M1[21];
17   float M2[21];
18   float k, tau,Gc, Gcn,N, Gn,dt, c, m, m1, m2;
19
20   float calc_m1 (float );
21   float calc_m2 (float );
22   float runge1 (float, float);
23   float runge2 (float, float, float);
24   //---------------------------------------------------------------------------
25   void __fastcall TForm1::Button1Click(TObject *Sender)
26   {
27   k = StrToFloat(Edit1->Text);
28   tau = StrToFloat(Edit2->Text);
29   Gc = StrToFloat(Edit3->Text);
30   Gcn = StrToFloat(Edit4->Text);
31   N = StrToFloat(Edit5->Text);
32   Gn = StrToFloat(Edit6->Text);
33   dt = StrToFloat(Edit7->Text);
34   M1[0] = StrToFloat(Edit8->Text);
35   M2[0] = StrToFloat(Edit9->Text);
```

```
36   c=1/(k*tau);
37
38   Series1->Clear();
39   Series2->Clear();
40
41    for (int t=0; t<0.5/dt+1; t++)
42    {
43     float m=M1[t], m2=M2[t];
44     M1[t+1]=runge1(dt,m);
45     m1=M1[t+1];
46     M2[t+1]=runge2(dt,m1,m2);
47    }
48   //построение графиков
49   for (int t=0; t<0.5/dt+1; t++)
50   {
51   Series1->AddXY(t*dt, M1[t],t*dt,clBlue);
52   Series2->AddXY(t*dt, M2[t],t*dt,clBlue);
53   }
54
55
56   }
57   //---------------------------------------------------------------------------
58   float calc_m1 (float m1)
59   {return -1*(c-0.5*Gc)*m1-0.5*Gcn+N;}
60   //---------------------------------------------------------------------------
61   float calc_m2 (float m1, float m2)
62   {return -2*(c-Gc)*m2+2*N*m1-3*Gcn*m1+Gn;}
63   //---------------------------------------------------------------------------
64   float runge1 (float dt, float m)
65   {
66    float k1, k2, k3, k4, dm;
67    k1=dt*calc_m1(m);
68    k2=dt*calc_m1(m+k1/2);
69    k3=dt*calc_m1(m+k2/2);
70    k4=dt*calc_m1(m+k3);
71    dm=(k1+2*k2+2*k3+k4)/6;
72    return m=m+dm;
73   }
74   //---------------------------------------------------------------------------
75   float runge2 (float dt, float m1, float m2)
76   {
77    float k1, k2, k3, k4, dm;
```

```
78    k1=dt*calc_m2(m1, m2);
79    k2=dt*calc_m2(m1, m2+k1/2);
80    k3=dt*calc_m2(m1, m2+k2/2);
81    k4=dt*calc_m2(m1, m2+k3);
82    dm=(k1+2*k2+2*k3+k4)/6;
83    return m2=m2+dm;
84    }
85
86    //---------------------------------------------------------------------------
87
88    void __fastcall TForm1::Button2Click(TObject *Sender)
89    {
90    Form1->Close();
91    }
```

В строках 16–23 производится объявление необходимых переменных и четырех функций.

Функции calc_m1() и calc_m2(), определение которых располагается в строках 58, 59 и 61, 62, реализуют правую часть уравнений для 1-го и 2-го моментов и принимают в качестве параметров значения моментов на предшествующем временном шаге.

Функции runge1() и runge2(), определение которых располагается в строках 64–73 и 75–84, реализуют алгоритм метода Рунге–Кутты. Их отличие заключается в том, что функция runge2() принимает в качестве параметров значения первого и второго моментов на предшествующем временном шаге. Обе функции при расчете коэффициентов k1, k2, k3, k4 вызывают соответственно функции calc_m1() и calc_m2(). Цикл, реализующийся в строках с 41 по 47, осуществляет последовательный расчет первого и второго моментов для каждого временного шага. В цикле с 49 по 53 строку стоится графики зависимости первого и второго моментов от времени.

Окно программы имеет вид, представленный на рис. 6.2.

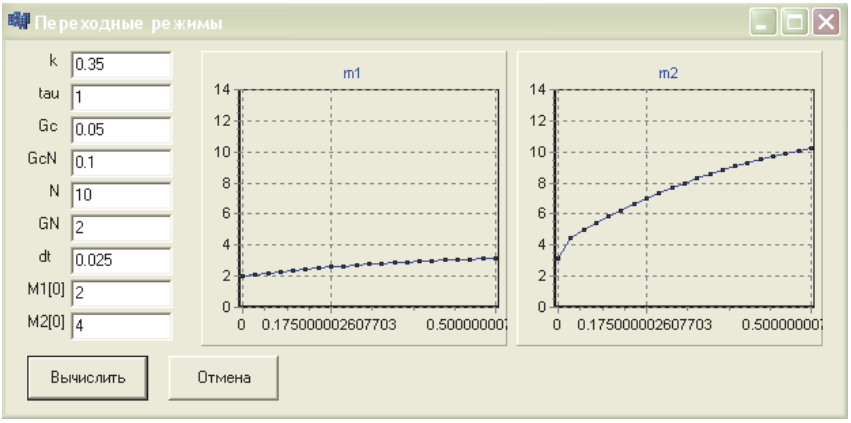


Рис. 6.2. Результат работы программы из листинга 6.1.

Если нарушить условие устойчивости для второго момента (взять значение $G_{\tilde{c}}$ больше $\bar{c}$), но обеспечить устойчивость по первому моменту ($\bar{c}$ >0.5 $G_{\tilde{c}}$), то результат работы программы будет таким, как показано на рис. 6.3.

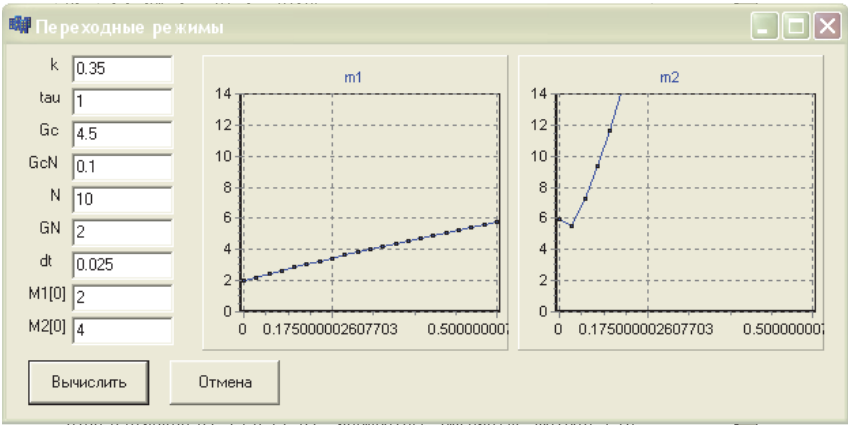


Рис. 6.3. Результат работы программы при неустойчивости по второму моменту.

7. Оценка устойчивости решений модели формирования многолетнего речного стока

7.1. Критерий устойчивости

Уравнение ФПК аппроксимируется системой уравнений для моментов. В теории случайных процессов подобная процедура известна, и в нашем случае она приводит к следующей системе дифференциальных уравнений:

$$\begin{aligned}
dm_1/dt &= -(\bar{c} - 0.5G_{\tilde{c}})m_1 - 0{,}5G_{\tilde{c}\tilde{N}} + \bar{N};\\
dm_2/dt &= -2(\bar{c} - G_{\tilde{c}})m_2 + 2\bar{N}m_1 - 3G_{\tilde{c}\tilde{N}}m_1 + G_{\tilde{N}};\\
dm_3/dt &= -3(\bar{c} - 1.5G_{\tilde{c}})m_3 + 3\bar{N}m_2 - 7.5G_{\tilde{c}\tilde{N}}m_2 + 3G_{\tilde{N}}m_1;\\
dm_4/dt &= -4(\bar{c} - 2G_{\tilde{c}})m_4 + 4\bar{N}m_3 - 14G_{\tilde{c}\tilde{N}}m_2 + 6G_{\tilde{N}}m_1.
\end{aligned} \qquad (7.1)$$

Из системы уравнений (7.1) видно, что при $\beta = G_{\tilde{c}}/\bar{c} = 2/i$ (здесь i – порядок момента) происходит потеря устойчивости решения для соответствующего момента (чем старше момент, тем при меньшей относительной интенсивности шума β он теряет устойчивость). Ранее [4, 5] было получено выражение, позволяющее вычислять значения β по информации, измеряемой на стандартной гидрометеорологической сети наблюдений:

$$\beta = 2\,k\ln r + 2, \qquad (7.2)$$

где r – коэффициент автокорреляции ряда расходов; k – коэффициент стока. Формула (7.2) является следствием экспоненциального решения

$$r = \exp[-(\bar{c} - 0.5G_{\tilde{c}})\Delta t] \qquad (7.3)$$

уравнения ФПК, определяющего изменение автокорреляционных функций для простых марковских процессов (время релаксации бассейнов для многолетнего стока τ = 1 год), к которым приводит

использование линейного формирующего фильтра. При годовой сдвижке $\Delta t = 1$ из данного выражения следует формула (7.2).

По формуле (7.2) были продиагностированы речные бассейны России. Выясняется, что многолетний годовой сток может формироваться неустойчиво по начальным моментам, в основном, в регионах с недостаточным увлажнением.

7.2. Программа на языке C++ в приложении C++ Builder

Основная форма приложения имеет вид, представленный на рис. 7.1.

Рис. 7.1. Форма приложения.

При щелчке по кнопке «Карта k» появляется форма Form2 с компонентом Image, связанным с графическим файлом, содержащим карту коэффициента стока (рис. 7.2, *б*). Кнопка «Таблица r(1)» вызывает форму Form3 с таблицей коэффициентов автокорреляции (см. рис. 7.2, *в*).

Обработчики событий представлены в листингах 7.1–7.3.

Листинг 7.1 (см. рис. 7.2, *а*).

```
0    #include <vcl.h>
1    #include <math.h>
2    #pragma hdrstop
```

```
3   #include "betta1.h" //подключение заголовочного файла для Form2
4   #include "betta3.h" //подключение заголовочного файла для Form3
5   #include "betta1.cpp" //подключение файла реализации для Form2
6   #include "betta3.cpp" //подключение файла реализации для Form3
7
8   #include "betta.h"
9   //---------------------------------------------------------------------------
10  #pragma package(smart_init)
11  #pragma resource "*.dfm"
12  TForm1 *Form1;
13  //---------------------------------------------------------------------------
14  __fastcall TForm1::TForm1(TComponent* Owner)
15        : TForm(Owner)
16  {
17  }
18  //---------------------------------------------------------------------------
19
20  void __fastcall TForm1::Button1Click(TObject *Sender)
21  {
22  Form2-> Visible = true; //появляется Form2 с картой k
23  }
24  //---------------------------------------------------------------------------
25
26  void __fastcall TForm1::Button2Click(TObject *Sender)
27  {
28  Form3-> Visible = true; //появляется Form3 с таблицей r(1)
29  }
30  //---------------------------------------------------------------------------
31  void __fastcall TForm1::Button4Click(TObject *Sender)
32  {
33  Form1->Close();
34  }
35  //---------------------------------------------------------------------------
36
37  void __fastcall TForm1::Button3Click(TObject *Sender)
38  {
39  float k; //коэффициент стока
40  float r; // коэффициент автокорреляции
41  float b; //критерий устойчивости
42
43  k =  StrToFloat(Edit1->Text);
44  r =  StrToFloat(Edit2->Text);
```

```
45
46    b = 2*k*log(r)+2;
47
48    Label4->Caption =  FloatToStrF(b, ffGeneral, 3, 2);
49
50    }
```

Листинг 7.2 (для формы с картой коэффициента стока).

```
0     #include <vcl.h>
1     #pragma hdrstop
2
3     #include "betta1.h"
4     //---------------------------------------------------------------------
5     #pragma package(smart_init)
6     #pragma resource "*.dfm"
7     TForm2 *Form2;
8     //---------------------------------------------------------------------
9     __fastcall TForm2::TForm2(TComponent* Owner)
10          : TForm(Owner)
11    {
12    }
13    //---------------------------------------------------------------------
14    void __fastcall TForm2::Button1Click(TObject *Sender)
15    {
16    Form2->Close();
17    }
```

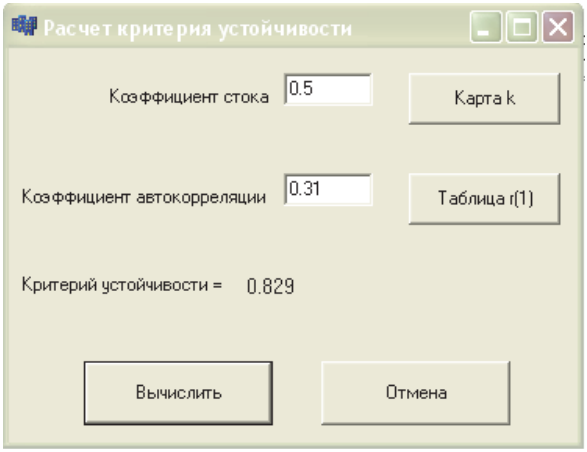
Листинг 7.3 (для формы с таблицей коэффициентов автокорреляции).

```
0     #include <vcl.h>
1     #pragma hdrstop
2
3     #include "betta3.h"
4     //---------------------------------------------------------------------
5     #pragma package(smart_init)
6     #pragma resource "*.dfm"
7     TForm3 *Form3;
8     //---------------------------------------------------------------------
9     __fastcall TForm3::TForm3(TComponent* Owner)
10          : TForm(Owner)
11    {
12    }
76
```

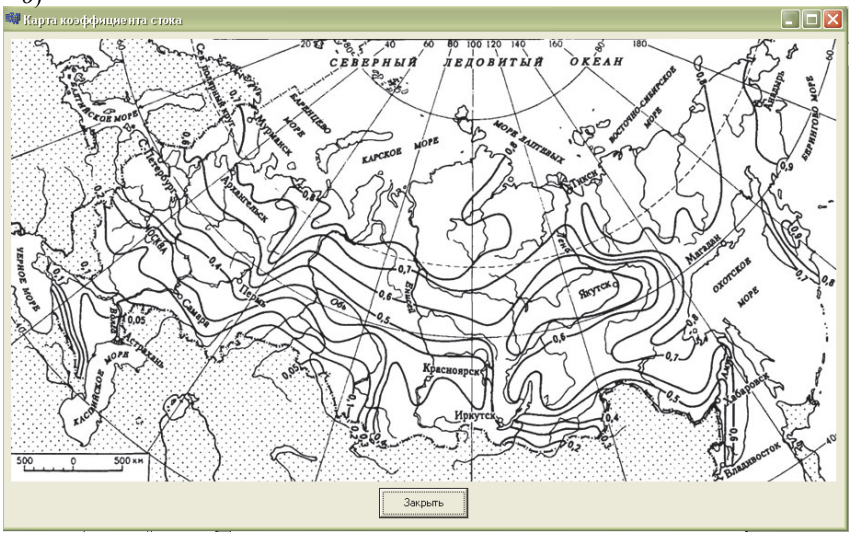
```
13  //---------------------------------------------------------------------------
14  void __fastcall TForm3::Button1Click(TObject *Sender)
15  {
16  Form3->Close();
17  }
```

Окно программы имеет вид, представленный на рис. 7.2.

а)



б)



в)

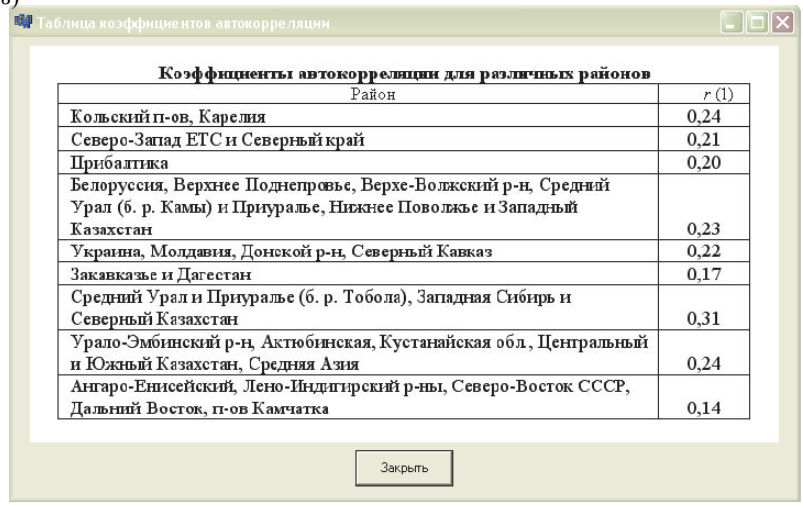


Рис. 7.2. Результат работы программы листинга 7.1 (*а*), листинга 7.2 (*б*) и листинга 7.3 (*в*).

8. Фрактальная диагностика гидрологических рядов

8.1. Описание методики фрактального диагностирования

Фрактальная размерность характеризует степень заполненности изучаемым объектом своего пространства вложения. Этим объектом в нашем случае служит временной ряд расходов воды. Фрактальность ряда связывается с порождающими его факторами. Если этих факторов много и они равновероятны, то приходим к белому шуму, который заполняет пространство вложения наподобие ничем не связанных молекул газа, находящихся в объеме. Если же во временном ряде существуют корреляции, то они образуют продолжительные группировки членов ряда. Это приводит к тому, что у ряда появляется своя собственная размерность (фрактальная, дробная).

По Б. Мандельброту, множество X называется фрактальным, если его размерность Хаусдорфа $D(X)$ не является целым числом. В свою очередь эта размерность, по определению, показывает, как растет число n шаров («шар» может иметь любую размерность) диаметром ε, необходимых для покрытия X: $n(\varepsilon) \approx 1/\varepsilon^D$ при $\varepsilon \to 0$

$$D = \lim_{\varepsilon \to 0} (\log n(\varepsilon) / \log(1/\varepsilon)) . \qquad (8.1)$$

В случае не фрактального множества формула (8.1) дает обычную размерность. Появление фрактальности связано со степенью фиксации предметной области. При ее слабой фиксации появляется неустойчивость (например, по моментам распределения) и, как следствие, – возникновение дробной размерности, т. е. индикатора начала переходных процессов появления новой фазовой переменной.

Знание фрактальных размерностей предельных множеств (аттракторов) позволяет оценить минимальное число фазовых переменных, необходимых для описания изучаемых процессов (размерность пространства вложения). Одновременное измерение всех составляющих вектора $\vec{x}(t)$, описывающего систему, не всегда возможно (мы можем не знать ни размерности $\vec{x}(t)$, ни что из себя

представляют его составляющие; возможно для наблюдения доступен только один из компонентов вектора $\vec{x}(t)$). Однако Такенс обосновал возможность восстановления размерности аттракторов d в фазовом пространстве размерности n по временной последовательности одного компонента, который несет информацию обо всех переменных, формирующих аттрактор. Почти всегда $d < n$; математически этот факт говорит о некомпактности («рыхлости») фрактала.

Следовательно, восстановив по временному ряду фрактальную размерность, можно целое число, непосредственно следующее за нею, считать минимальным числом переменных, необходимых для построения модели. Оно определяет наименьшее число дифференциальных уравнений первого порядка, описывающих динамику изучаемой системы.

Таким образом, зная значения одной физической переменной, доступной измерению (в нашем случае это расход воды), взятые со сдвижкой во времени τ ($Q(t), Q(t+\tau), Q(t+2\tau), \ldots$), мы строим псевдофазовое пространство (пространство вложения). Для этого фиксируется малое ε, которое используется как «метр» для зондирования структуры аттрактора. Далее подсчитывается число точек, расстояние между которыми не превосходит r. Для d-мерного многообразия число таких точек $(r/\varepsilon)^d$. Поэтому корреляционный интеграл будет меняться как $C(r) = r^d$, т. е. размерность аттрактора может быть определена как коэффициент наклона зависимости $\ln C(r) = d \cdot \ln r$. Такие же зависимости надо построить для проекций возрастающей размерности n, и размерностью аттрактора считать стабилизировавшееся значение.

8.2. Программа на языке C++ в приложении C++ Builder

Рассматриваемый вариант программы разрабатывался для дежурств в учебном Бюро гидрологических прогнозов при прогнозировании суточных стоковых характеристик. В связи с этим сдвижка ряда во времени, число диапазонов и длина ряда задаются литеральными и символьными константами, что делает программу не универсальной для любого вида стока.

Форма приложения имеет вид, представленный на рис. 8.1.

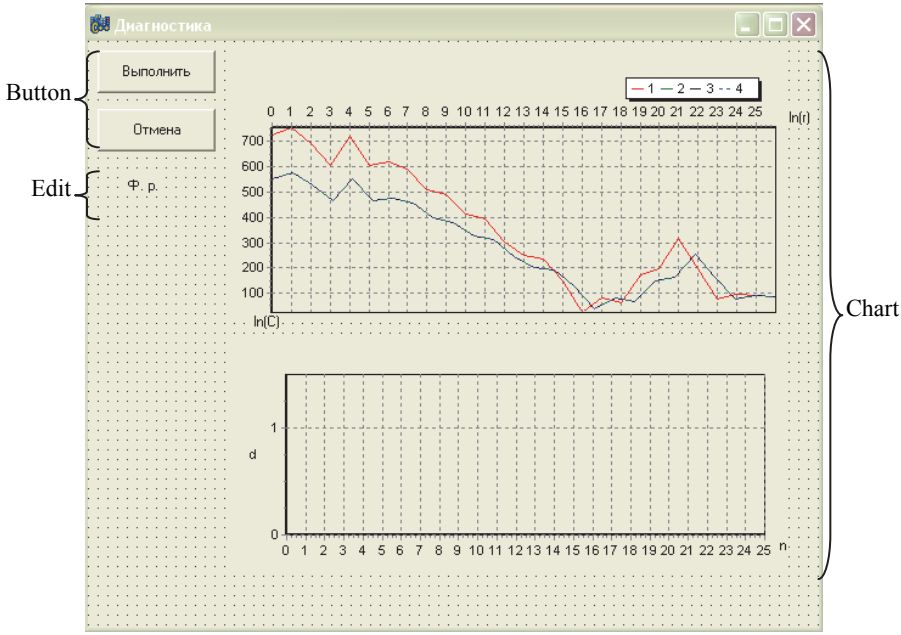


Рис. 8.1. Форма приложения.

На форме размещены два компонента Chart для построения зависимости $\ln C(r) = f(\ln r)$ и кривой насыщения. Некоторые оси графиков подписаны с помощью компонента Label. Еще два компонента Label, слева на форме, используются для вывода результата работы программы – фрактальной размерности ряда. Алгоритм расчета в виде строк кода закреплен за событием OnClick кнопки Button1 («Выполнить»).

Возможный вариант программы представлен в листинге 8.1.

Листинг 8.1.

```
0    #include <vcl.h>
1    #include <math.h>
2    #include <fstream.h> //библиотека для файлового ввода и вывода
3    #pragma hdrstop
4
```

```
5   #include "DiagnBC.h"
6   //---------------------------------------------------------------------------
7   #pragma package(smart_init)
8   #pragma resource "*.dfm"
9   TForm1 *Form1;
10  //---------------------------------------------------------------------------
11  __fastcall TForm1::TForm1(TComponent* Owner)
12        : TForm(Owner)
13  {
14  }
15  //---------------------------------------------------------------------------
16
17  void __fastcall TForm1::Button2Click(TObject *Sender)
18  {
19  Form1->Close(); //кнопка «Отмена»
20  }
21  //---------------------------------------------------------------------------
22
23  void __fastcall TForm1::Button1Click(TObject *Sender)
24  {
25
26  const int N = 30;
27  double Q[N];
28   //открытие файла для чтения
29   //для примера р. Пяльма, д. Пяльма, июль (31) 1980 г.
30  ifstream fin("Q.txt");
31  for (int i=0; i<N; i++)
32     fin >> Q[i];
33  fin.close();
34
35  double maxx;
36  double r1;
37
38  //число точек на кривой насыщения
39  const int f =  4;
40
41  //число диапазонов
42  const int r =  4;
43  //очистка графического компонента
44  Series1->Clear();
45  Series2->Clear();
46  Series3->Clear();
```

```
47   Series4->Clear();
48   Series5->Clear();
49
50   double d[f]; //массив для фрактальных размерностей
51
52   // «построение» псевдофазового портрета (1f)
53   double F1 [N-f+1][ N-f+1];
54   for (int j=0; j<N-f+1; j++)
55   {
56      F1[0][j] = Q[j];
57   }
58   maxx = 0;  //для нахождения максимального значения
59   for (int i=1; i<N-f+1; i++)
60   {
61      for (int j=i-1; j<N-f+1; j++)
62      {
63         F1[i][j]= (F1[i-1][i-1] - F1[i-1][j]);
64         if (fabs(F1[i][j]) > maxx) maxx = F1[i][j];
65      }
66   }
67   double n[r];
68   r1 = maxx/r;
69    double lnc[r], lnr[r];
70    //вычисление корреляционного интеграла для 1f
71    for (int c=1; c<=r; c++)
72   {
73      n[c]=0;
74      for (int i=1; i<N-f+1; i++)
75      {
76         for (int j=i; j<N-f+1; j++)
77         {
78               if (fabs(F1[i][j]) < r1*c)
79               { n[c]= n[c]+1;}
80         }
81      }
82      lnc[c] = log(n[c]/(((N-f+1)*(N-f+1)-(N-f+1))/2-1));
83      lnr[c] = log(r1*c);
84   }
85   d[0] = (lnc[2]-lnc[1])/(lnr[2]-lnr[1]);
86   //построение зависимости ln(c) от ln(r) для 1f
87   for (int i=1; i<=r; i++)
88   {
```

```
89      Series1->AddXY(lnr[i], lnc[i], i,clRed);
90    }
91    //сдвижка ряда tau1=1
92    double F2 [N-f+1][ N-f+1];
93    for (int j=0; j<N-f+1; j++)
94    {
95      F2[0][j] = Q[j+1];
96    }
97    for (int i=1; i<N-f+1; i++)
98    {
99      for (int j=i-1; j<N-f+1; j++)
100     {
101       F2[i][j]=(F2[i-1][i-1]-F2[i-1][j]);
102     }
103   }
104   // «построение» псевдофазового портрета (2f=1f+tau1)
105   double F3 [N-f+1][N-f+1];
106   maxx = 0;
107   for (int i=1; i<N-f+1; i++)
108   {
109     for (int j=i-1; j<N-f+1; j++)
110     {
111       F3[i][j]=fabs(F1[i][j])+fabs(F2[i][j]);
112       if (F3[i][j]> maxx) maxx = F3[i][j];
113     }
114   }
115   r1 = maxx/r;
116   // вычисление корреляционного интеграла для 2f
117   for (int c=1; c<=r; c++)
118   {
119     n[c]=0;
120     for (int i=1; i<N-f+1; i++)
121     {
122       for (int j=i; j<N-f+1; j++)
123       {
124             if (fabs(F3[i][j]) < r1*c)
125             { n[c]= n[c]+1;}
126       }
127     }
128     lnc[c] = log(n[c]/(((N-f+1)*(N-f+1)-(N-f+1))/2-1));
129     lnr[c] = log(r1*c);
130   }
```

```
d[1] = (lnc[1]-lnc[2])/(lnr[1]-lnr[2]);
// построение зависимости ln(c) от ln(r) для 2f
for (int i=1; i<=r; i++)
{
   Series2->AddXY(lnr[i], lnc[i],i,clGreen);
}
/ /сдвижка ряда tau2=2
double F4 [N-f+1][N-f+1];
for (int j=0; j<N-f+1; j++)
{
   F4[0][j] = Q[j+2];
}
for (int i=1; i<N-f+1; i++)
{
   for (int j=i-1; j<N-f+1; j++)
   {
      F4[i][j]=(F4[i-1][i-1]-F4[i-1][j]);
   }
}
// «"построение» псевдофазового портрета (3f=2f+tau2)
double F5 [N-f+1][N-f+1];
maxx = 0;
for (int i=1; i<N-f+1; i++)
{
   for (int j=i-1; j<N-f+1; j++)
   {
      F5[i][j]=fabs(F3[i][j])+fabs(F4[i][j]);
      if (F5[i][j]> maxx) maxx = F5[i][j];
   }
}
r1 = maxx/r;
//вычисление корреляционного интеграла для 3f
for (int c=1; c<=r; c++)
{
   n[c]=0;
   for (int i=1; i<N-f+1; i++)
   {
      for (int j=i; j<N-f+1; j++)
      {
            if (fabs(F5[i][j]) < r1*c)
            { n[c]= n[c]+1;}
      }
```

```
173     }
174     lnc[c] = log(n[c]/(((N-f+1)*(N-f+1)-(N-f+1))/2-1));
175     lnr[c] = log(r1*c);
176   }
177   d[2] = (lnc[1]-lnc[2])/(lnr[1]-lnr[2]);
178   //построение зависимости ln(c) от ln(r) для 3f
179   for (int i=1; i<=r; i++)
180   {
181     Series3->AddXY(lnr[i], lnc[i],i,clYellow);
182   }
183
184   // сдвижка ряда tau3=3
185   double F6 [N-f+1][N-f+1];
186   for (int j=0; j<N-f+1; j++)
187   {
188     F6[0][j] = Q[j+3];
189   }
190   for (int i=1; i<N-f+1; i++)
191   {
192     for (int j=i-1; j<N-f+1; j++)
193     {
194       F6[i][j]=(F6[i-1][i-1]-F6[i-1][j]);
195      }
196   }
197   // «построение» псевдофазового портрета (4f=3f+tau3)
198   double F7 [N-f+1][N-f+1];
199   maxx = 0;
200   for (int i=1; i<N-f+1; i++)
201   {
202     for (int j=i-1; j<N-f+1; j++)
203     {
204       F7[i][j]=fabs(F5[i][j])+fabs(F6[i][j]);
205       if (F7[i][j]> maxx) maxx = F7[i][j];
206     }
207   }
208   r1 = maxx/r;
209   //вычисление корреляционного интеграла для 4f
210   for (int c=1; c<=r; c++)
211   {
212     n[c]=0;
213     for (int i=1; i<N-f+1; i++)
214     {
```

```
215         for (int j=i; j<N-f+1; j++)
216         {
217               if (fabs(F7[i][j]) < r1*c)
218               { n[c]= n[c]+1;}
219         }
220      }
221      lnc[c] = log(n[c]/(((N-f+1)*(N-f+1)-(N-f+1))/2-1));
222      lnr[c] = log(r1*c);
223   }
224   d[3] = (lnc[1]-lnc[2])/(lnr[1]-lnr[2]);
225   //построение зависимости ln(c) от ln(r) для 4f
226   for (int i=1; i<=r; i++)
227   {
228      Series4->AddXY(lnr[i], lnc[i],i,clBlue);
229   }
230   double dd[5]; //вспомогательный массив для кривой насыщения
231   for (int i=0; i<r+1; i++)
232   {
233      dd[0]= 0;
234      dd[i+1] = d[i];
235      Series5->AddXY(i, dd[i],i,clRed);
236   }
237   float fr;   //фрактальная размерность
238   for (int i=1; i<r; i++)
239   {
240      if ((dd[i+1] - dd[i])/ dd[i] <= 0.1)
241      fr = dd[i+1]; break;
242   }
243   Label4->Caption = FloatToStrF(fr, ffGeneral, 2, 2);
244   }
```

Программа снабжена комментариями, поэтому поясним только некоторые строки.

В строках 30–33 происходит считывание расходов воды из текстового файла Q.txt. В качестве примера взят ряд суточных расходов воды на р. Пчевжа – д. Белая за месяц июль 1980 года. В 35-й строке объявлена переменная maxx вещественного типа для максимальных значений; слово max – входит в расширенный список ключевых слов, использовать его в качестве идентификатора нельзя. В строке 64 используется математическая функция fabs, возвращающая абсолютное значение числа. Функция находится в

подключаемой библиотеке math.h. В строках 231–236 происходит построение кривой насыщения, начинающейся с нулевого значения (строка 233). Считается, что кривая выходит на прямую, когда последующее значение фрактальной размерности отличается от предыдущего не более чем на 10 % (строки 238–242).

Окно программы имеет вид, представленный на рис. 8.2.

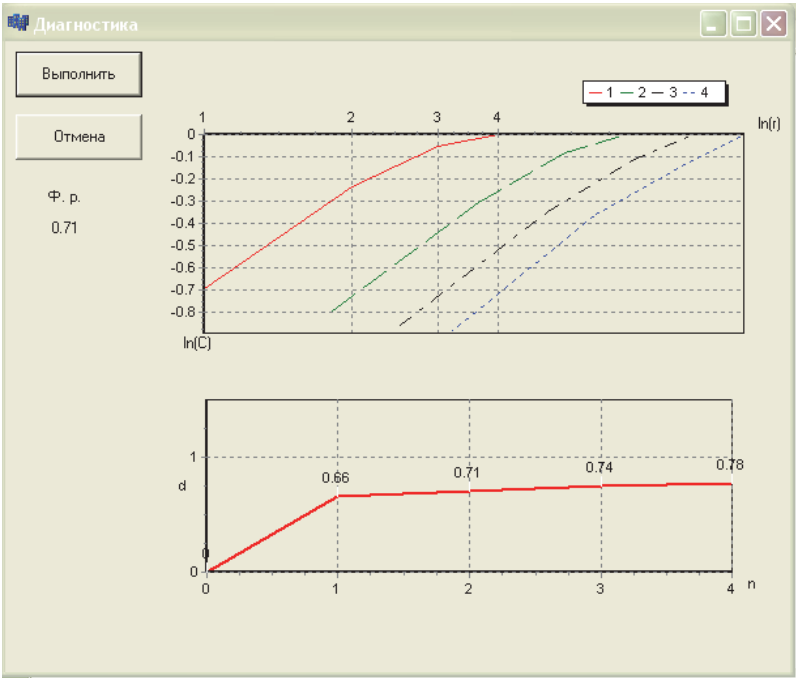


Рис. 8.2. Результат работы программы листинга 8.1.

9. Системная модель, включающая взаимодействие различных звеньев, участвующих в гидрологическом цикле

9.1. Текст задания для самостоятельного решения

Водохозяйственный объект (задается искусственная ситуация, не соответствующая по численным значениям расчетных величин какому-либо реальному объекту), включает водоем с двумя втекающими и двумя вытекающими реками (рис. 9.1). На нижней реке на 20-м километре имеется железнодорожный мост с подмостовым отверстием, способным пропускать волны дождевых паводков уровнем не более 3 м над условной плоскостью сравнения. При б*о*льших уровнях вода начинает переливаться через железнодорожную насыпь, что приводит к аварии.

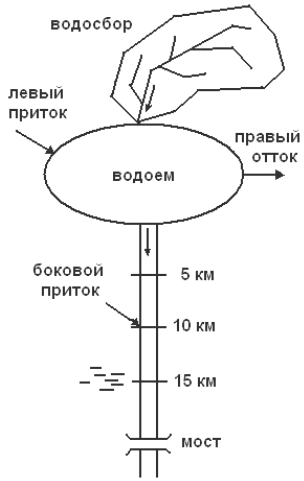


Рис. 9.1. Схема водохозяйственного объекта.

Известно также, что на 10-м километре есть боковой приток, вызывающий изменение уровня в главной реке 0.5 м/ч в точке слияния; на 15-м километре имеет место взаимодействие речных и грунтовых вод, с коэффициентом взаимодействия (или фильтрации) $k_ф = 0.1$ 1/ч.

Из непосредственных измерений на гидрологических постах следует, что расходы левого притока и правого оттока не зависят от уровня воды в водоеме и равны $Q_л = 19$ м^3/ч; $Q_п = 21$ м^3/ч.

Кроме того известна следующая дополнительная информация:

– для бассейна верхнего притока – время добегания $T_{доб} = 2$ ч; коэффициент стока $k = 0.16$; $Q|_{t=0} = 10$ м^3/ч;

– для водоема – морфометрический коэффициент $k_{морф} = 0.46$ 1/ч, $H|_{t=0} = 1$ м, площадь $F = 19.2$ м^2, которая не меняется при изменении уровня;

– для руслового стока на нижней реке – скорость распространения волны по руслу a = const = 1760 м/ч, начальное наполнение русла 1 м вдоль всей реки, движение воды в русле управляется (создается) изменением уровня воды в водоеме;

– для насыщенной – грунтовые воды имеют горизонтальную поверхность по обеим координатам x и y вне зависимости от того, какие процессы происходят в реке; внешнее воздействие на зону насыщения отсутствует, т. е. равно нулю; происходит только взаимодействие с речными водами, причем так, что $H_{гр}$ = const = 1 м на всем промежутке времени, на котором мы интересуемся процессами в данной системе.

В качестве внешнего воздействия на всю рассматриваемую систему задается метеопрогноз по осадкам на бассейн верхнего притока в водоем (табл. 9.1).

Таблица 9.1.

Внешнее воздействие на систему

t ч	0	1	2	3	4	5
X м^3/ч	200	500	500	300	150	0

Необходимо рассчитать процесс формирования волны по всем элементам системы и определить максимальную высоту (амплитуду) волны на 20-м километре в районе моста, а также время прохождения максимума под мостом. Выяснить, произойдет ли авария.

Полезные указания, упрощающие расчеты:

– При рассмотрении процесса формирования стока на верхнем бассейне не учитывать влияния на этот процесс поднятия уровня в водоеме.

– При расчете подъема уровня в водоеме не учитывать изъятия части стока нижней рекой.

– Шаги по времени брать $\Delta t = 1$ ч, а по координате $x = 5000$ м.

– При переходе к 15-му километру надо использовать условие стыковки модели руслового стока и насыщенной зоны.

9.2. Программа на языке C++ в приложении C++ Builder

Форма приложения имеет вид, представленный на рис. 9.2.

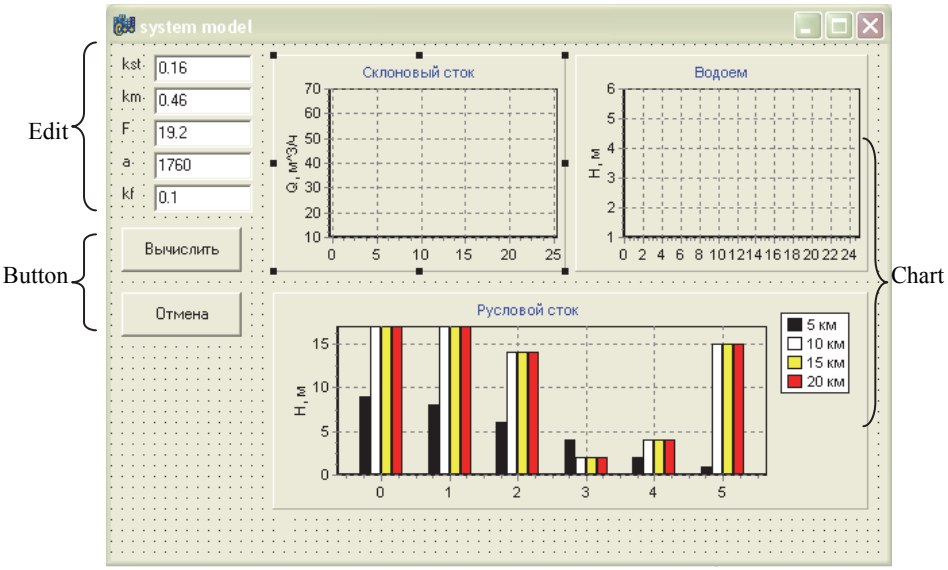


Рис. 9.2. Форма приложения.

Возможный вариант программы представлен в листинге 9.1. Действия программы понятны из комментариев.

Листинг 9.1.

```
0    #include <vcl.h>
1    #pragma hdrstop
2
3    #include "k_r.h"
4    //---------------------------------------------------------------------
5    #pragma package(smart_init)
```

```
6   #pragma resource "*.dfm"
7   TForm1 *Form1;
8   //---------------------------------------------------------------------------
9   __fastcall TForm1::TForm1(TComponent* Owner)
10        : TForm(Owner)
11  {
12  }
13  //---------------------------------------------------------------------------
14  void __fastcall TForm1::Button1Click(TObject *Sender)
15  {
16  int l; // счетчик цикла по расстоянию
17  int t; // счетчик цикла по времени
18  int dt= 1; //шаг интегрирования по времени
19  int tau= 2; //время добегания
20  float Kst; //коэффициент стока
21  float Km; //морфометрический коэффициент
22  float F; // площадь водоема
23  int Q1= 19; // расход левого оттока
24  int Q2= 21 ; // расход правого притока
25  int a; // скорость волны
26  int dx=5000; // шаг по расстоянию для модели руслового стока
27  int dL=20000; //длина реки до моста
28  float Kf; //коэффициент фильтрации
29  float Hg=1.0; //уровень грунтовых вод
30  int NaUr=1; // начальное наполнение русла
31
32  float X[10] = {200, 500, 500, 300, 150, 0, 0, 0, 0, 0}; //интенсивность
    осадков
33  float Q[10] = {10}; //массив расходов воды
34  float H[10] = {1}; //массив уровней водоема
35
36  float Hr[5][10]; //массив уровней реки
37  float Hx; // приток к реке
38
39  Kst = StrToFloat(Edit1->Text);
40  Km = StrToFloat(Edit2->Text);
41  F = StrToFloat(Edit3->Text);
42  a = StrToFloat(Edit4->Text);
43  Kf = StrToFloat(Edit5->Text);
44
45  Series1->Clear();
46  Series2->Clear();
```

```
47  Series3->Clear();
48  Series4->Clear();
49  Series5->Clear();
50  Series6->Clear();
51
52  //модель водосбора
53  for (t=1; t<10; t++)
54  {
55   Q[t]=Q[t-1]+dt*(Kst*X[t-1]-Q[t-1])/tau;
56  if (Q[t]<Q[0]) Q[t]=Q[0];
57  Series1->AddXY(t, Q[t],t,clBlue);
58  }
59  //модель водоёма
60  for (t=1; t<10; t++)
61  {
62   H[t]=H[t-1]+dt*((Q[t-1]+Q1-Q2)/F-Km*H[t-1]);
63  if (H[t]<H[0]) H[t]=H[0];
64  Series2->AddXY(t, H[t],t,clBlue);
65  }
66  //модель руслового стока
67  if (a*dt/dx<1) // проверка условия устойчивости
68  {
69  for (l=0; l<5; l++)
70    Hr[l][0]=NaUr;
71  for (t=0; t<10; t++)
72    Hr[0][t]=H[t]; // граничные условия
73  for (t=1; t<10; t++)
74  {
75     for (l=1; l<5; l++)
76      {
77      if (l==2)
78        Hx=0.5;
79      if (l==3)
80        Hx=-Kf *(Hr[3][t-l] - Hg);
81      if (l==4)
82        Hx=0;
83      Hr[l][t]= Hr[l][t-1]+dt*(Hx-a*(Hr[l][t-1]-Hr[l-1][t-1])/dx);
84     }
85  }
86   }
87
88  for (t=1; t<10; t++)
```

```
89    {
90    Series3->AddXY(t, Hr[1][t],t,clBlue);
91    Series4->AddXY(t, Hr[2][t],t,clGreen);
92    Series5->AddXY(t, Hr[3][t],t,clYellow);
93    Series6->AddXY(t, Hr[4][t],t,clRed);
94    }
95    for (t = 1; t<10; t++)
96     {
97    if (Hr[4][t]>3) //если уровень больше 3 на 20-м км,
98      Label6->Caption ="Failure!!!"; // то - авария!
99    else
100   Label6->Caption = "The condition of stability is not carried out!\n";
101   }
102   }
103
104   //---------------------------------------------------------------------------
105   void __fastcall TForm1::Button2Click(TObject *Sender)
106   {
107   Form1->Close();
108   }
```

Окно программы имеет вид, представленный на рис. 9.3.

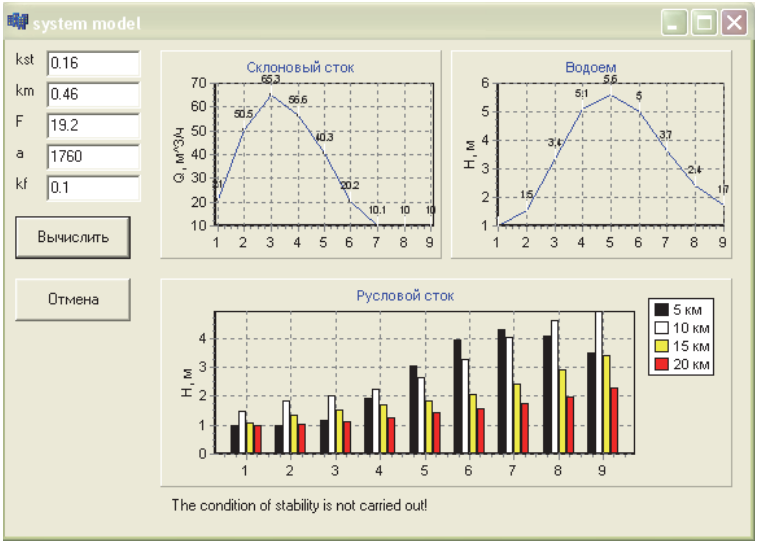


Рис. 9.3. Результат работы программы из листинга 9.1.

Заключение

В части III Практикума (заключительной) не удалось в полной мере проиллюстрировать решение гидрологических задач, связанных с расширением размерности неустойчивых (развивающихся) гидрологических объектов. Однако показано как решаются задачи по выявлению степени статистической неустойчивости и вычислению фрактальной размерности рядов, доступных непосредственному измерению. Это связано с тем, что визуализация фазовых пространств требует привлечения трехмерной графики, которая отсутствует в имеющихся вариантах C++ Builder. В рекомендуемой литературе (см. например, [4]) трехмерные иллюстрации присутствуют, но они получены с помощью других (коммерческих) приложений. Этот недостаток предполагается устранить в последующих изданиях учебных пособий по рассматриваемой тематике.

В целом во всех трех частях практикума удалось сделать главное: проиллюстрировать возможности применения языка C++ (и его расширения C++ Builder) для решения задач, связанных с моделированием гидрологических процессов разными типами моделей. Получилось не только пособие по моделированию, но и самому языку C++ («два в одном»). Появление подобного пособия во многом способствовали гранты Министерства образования и науки РФ (№ 196/08, № 2.1.1/3355, № П740, № 2.1.1/9596, № 5.3400.2011, № 14.B37.21.0678).

Список литературы

1. *Архангельский А. Я., Тагин М. А.* Программирование в C++ Builder 6 и 2006. – М.: ООО «Бином-Пресс», 2007. – 1184 с.
2. *Боровский А. Н.* C++ и Borland C++ Builder. Самоучитель. – СПб.: Питер, 2005. – 256 с.
3. *Вальпа О. Д.* Borland C++ Builder. Экспресс-курс. – СПб.: БХВ-Петербург, 2006. – 224 с.
4. *Коваленко В. В.* Частично инфинитная гидрология. – СПб.: изд. РГГМУ, 2007. – 230 с.
5. *Коваленко В. В.* Моделирование гидрологических процессов. – СПб.: Гидрометеоиздат, 1993. – 256 с.
6. *Коваленко В. В., Викторова Н. В., Гайдукова Е. В.* Моделирование гидрологических процессов. Изд. 2-е исправ. и доп. Учебник. – СПб., изд. РГГМУ, 2006. – 559 с.
7. *Коваленко В. В., Гайдукова Е. В., Викторова Н. В.* Практикум по дисциплине «Моделирование гидрологических процессов. Часть I. Динамическое модели» (на базе языка C++). Учебное пособие. – СПб.: изд. РГГМУ, 2010. – 152 с.
8. *Коваленко В. В., Гайдукова Е. В., Викторова Н. В.* Практикум по дисциплине «Моделирование гидрологических процессов. Часть II. Стохастическое модели» (на базе языка C++). Учебное пособие. – СПб.: изд. РГГМУ, 2012. – 247 с.
9. *Культин Н. Б.* Самоучитель C++ Builder. – СПб.: БХВ-Петербург, 2005. – 320 с.

Содержание

The contents

Учебное издание

Коваленко Виктор Васильевич,
Гайдукова Екатерина Владимировна,
Викторова Наталья Владимировна

ПРАКТИКУМ ПО ДИСЦИПЛИНЕ
«МОДЕЛИРОВАНИЕ ГИДРОЛОГИЧЕСКИХ ПРОЦЕССОВ.
ЧАСТЬ III.
ЧАСТИЧНО ИНФИНИТНОЕ МОДЕЛИРОВАНИЕ»
(на базе языка C++ Builder)

Редактор: *И.Г. Максимова*

ЛР № 020309 от 30.12.96

Подписано в печать 01.02.13. Формат 60×90 1/16. Гарнитура Times New Roman.
Печать цифровая. Усл.печ.л. 6,4. Тираж 200 экз. Заказ № 162.
РГГМУ, 195196, Санкт-Петербург, Малоохтинский пр., 98.
Отпечатано в ЦОП РГГМУ