Министерство образования и науки Украины

Харьковский национальный университет имени В. Н. Каразина

М. А. Волков, А. С. Сорочкин, В. М. Лазурик, В. Т. Лазурик

Разработка программ генераторов псевдослучайных чисел

Учебно-методическое пособие по дисциплинам

«Языки прикладного программирования»

«Компьютерное моделирование стохастических процессов»

Харьков - 2009

УДК 004.43 (075.8)
ББК 32. 973. 26 – 018. 1я73 DELPHI
Р17

Утверждено ученым советом факультета компьютерных наук
Харьковского национального университета имени В. Н. Каразина
(протокол № 9 от 23 апреля 2009 г.)

Рецензенты: Зав. кафедрой искусственного интеллекта и программного обеспечения Харьковского национального университета имени В.Н. Каразина профессор **Куклин В. М.**; председатель учебно-методической комиссии Института высоких технологий, доцент **Юнаков Н. Н.**.

Волков М. А., Сорочкин А. С., Лазурик В. М., Лазурик В. Т.
Р 17 Разработка программ генераторов псевдослучайных чисел: Методическое пособие по курсовому проектированию для студентов компьютерных специальностей. – Х.: ХНУ им. В. Н. Каразина, 2009. – 55 с.

В данном методическом пособии по курсовому проектированию рассматриваются отдельные моменты разработки программ генераторов случайных чисел в визуальной среде разработки Delphi. Пособие содержит примеры, разработанные авторами, с подробным описанием и комментариями.

УДК 004.43 (075.08)
ББК 32. 973. 26 – 018. 1я73 DELPHI

Оглавление

Предисловие

В методическом пособии описаны приемы разработки программного обеспечения моделирования стохастических процессов с заданными законами распределения вероятности методом Монте-Карло. Примеры разработки приведены с использованием визуальной среды программирования Delphi 7. **в пособии выделен ряд моментов, важных, с точки зрения авторов, в процессе разработки программного обеспечения. Приведены коды программных реализаций, даны пояснения и комментарии.**

Методическое пособие может быть рекомендовано студентам факультета компьютерных наук, обучающимся по специальности «Информационные управляющие системы и технологии» при выполнении курсовых проектов по дисциплинам «Языки прикладного программирования» и **«Компьютерное моделирование стохастических процессов».**

В связи с тем, что в контингент студентов, обучающихся на факультете компьютерных наук, входят иностранные студенты, для которых в контракте определен русский язык как язык обучения, данное методическое пособие издается на русском языке.

1. Метод Монте-Карло

Численный метод решения практических задач при помощи моделирования случайных величин и статистической оценки их характеристик называется методом Монте-Карло. В основе метода лежит моделирование статистического эксперимента с помощью средств вычислительной техники и регистрация числовых характеристик, получаемых в этом эксперименте. Важнейший прием построения метода Монте-Карло – сведение задачи к расчету математических ожиданий (**M**). Для того, чтобы приближенно вычислить некоторую скалярную величину **a**, надо придумать такую случайную величину **ξ**, что **Mξ=a**; тогда, вычислив **N** независимых значений $\xi_1,\ldots,\xi_N$ величины **ξ**, можно считать, что $a \sim (1/N)(\xi_1+\ldots+\xi_N)$ [1], [2]. Для реализации метода Монте-Карло, как правило, используют «псевдослучайную» последовательность

чисел, генераторы которых предусмотрены в различных языках программирования. В приведенных программных кодах используется стандартный генератор псевдослучайных чисел, имеющийся в Delphi, – функция random. Функция используется для получения случайных чисел, равномерно распределенных на отрезке [0,1]. Для разработки генератора псевдослучайных чисел с заданной плотностью вероятности необходимо разработать алгоритм преобразования равномерно распределенных случайных величин в случайные величины с заданным распределением.

Целью курсовой работы является разработка в визуальной среде программирования Delphi [3]-[9] программы – генератора случайных чисел, подчиняющихся заданному закону распределения. Из множества различных методов преобразования случайных величин в курсовых работах рекомендовано использовать в случае непрерывного распределения метод обратных функций и метод Неймана, в случае дискретного распределения – метод интервалов.

2. Разработка интерфейса программы

Одним из основных моментов при разработке приложений является грамотное проектирование интерфейса пользователя. Сложный в применении интерфейс, как минимум, приводит к ошибкам пользователя. Все современные персональные компьютеры поддерживают графический интерфейс (graphical user interface – GUI), который подразумевает использование цветного графического экрана с высоким разрешением и позволяет работать с мышью и клавиатурой. Из пяти основных стилей взаимодействия [10] при разработке приложения по курсовой работе наиболее приемлемыми являются *Выбор из меню* и *Заполнение форм*. При *Выборе из меню* пользователь выбирает команду из списка пунктов меню, а при *Заполнении форм* заполняет поля экранной формы. На форме могут быть другие элементы, в том числе и командные кнопки, при щелчке мышью на которых инициируется некоторое действие. К взаимодействию с пользователем следует отнести сообщения об ошибках и справочную систему. Сообщения об ошибках должны быть на том языке, на

каком программа общается с пользователем и в тех терминах, которые используются в данной задаче; сообщения должны быть вежливыми, краткими, не содержать оскорблений, четко описывать ситуацию, приведшую к ошибке, и давать варианты ее исправления. Файл справочной информации может быть реализован любым способом. Но в любом случае, каким бы способом ни была реализована справочная система, она должна содержать, по крайней мере, три основных раздела: *User guide* (учебник пользователя с описанием решаемой задачи и методов ее решения), *How to get result* (инструкция пользователя по работе с программой), *About* (сведения об авторах, контакты).

Первое впечатление при работе с программой пользователь получает от ее внешнего вида. Неряшливо оформленный, многоцветный, немасштабируемый интерфейс так же раздражает, как и нечетко сформулированные сообщения об ошибках. Цветовая гамма, используемая в разработке, должна быть спокойной. Формы ярких цветов раздражают глаз, с ними невозможно долго работать. Все формы приложения должны быть выполнены в одном стиле. Значение цвета лучше всего задавать в 16-тиричном виде. Каждый компонент, расположенный на форме, должен иметь четкое назначение, перенасыщенный компонентами интерфейс затрудняет работу с программой. Для выполнения курсовой работы может быть предложен минимальный набор элементов формы: поля для ввода параметров и значения числа экспериментов; главное меню; кнопки для управления ходом вычислений, причем при реализации расчетов методом Монте-Карло наличие кнопки СТОП, позволяющей прервать вычисления, обязательно; компоненты для когнитивной визуализации результатов вычислений, такие как TStringGrid (сетка),Tchart (график), TEdit (поля вывода); TProgressBar (компонент для визуализации процесса вычислений). Вид интерфейса программы показан на рис. 1 и рис. 2.

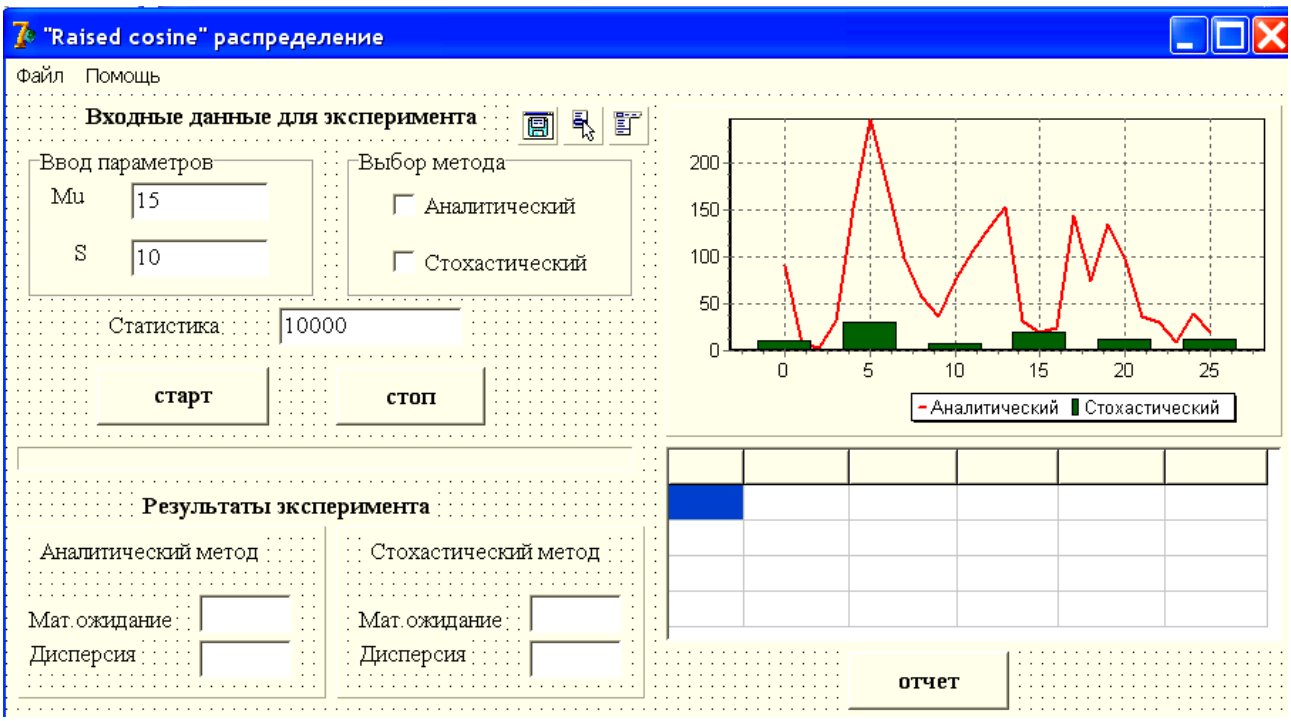


Рис. 1. Интерфейс программы в режиме конструктора

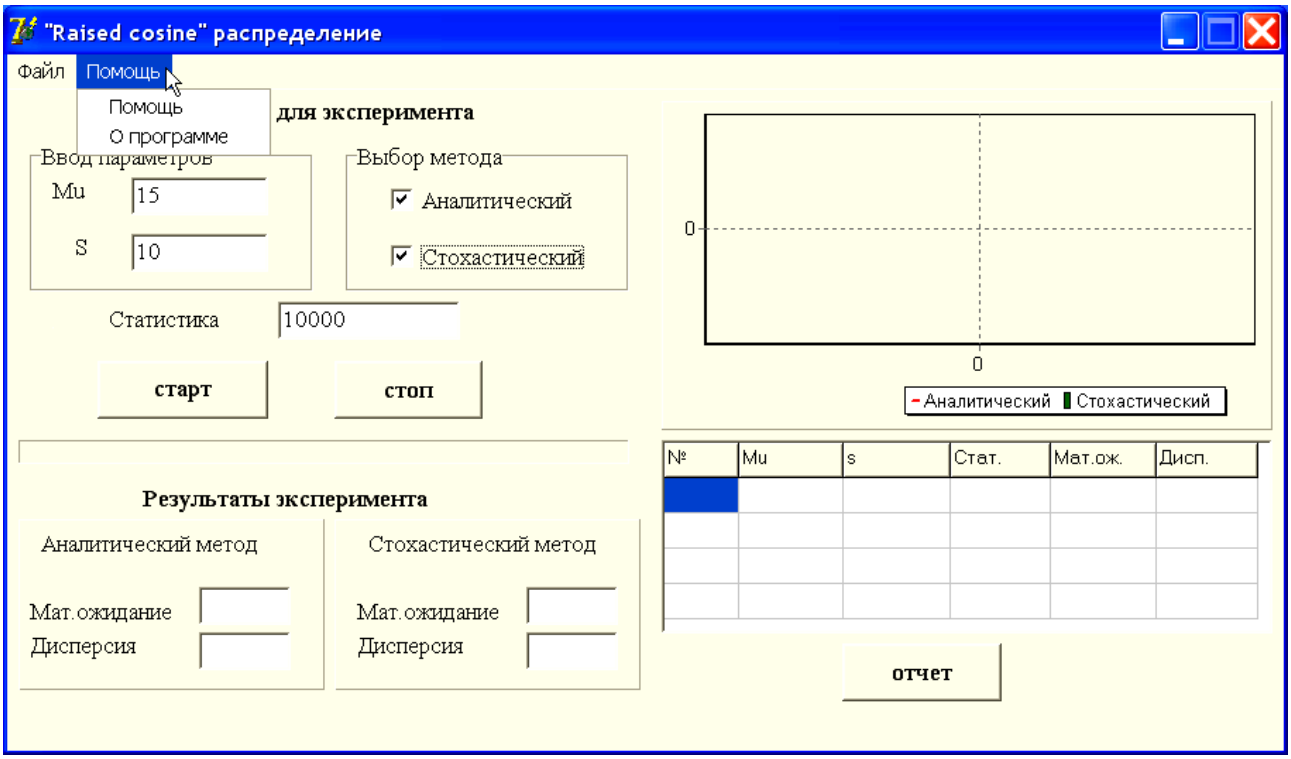


Рис. 2. Интерфейс программы, запущенной на выполнение

2.1. Защита полей ввода

Для ввода параметров расчета могут быть использованы поля ввода Edit. Введенные пользователем данные должны быть преобразованы к Integer или

Float с использованием функций StrToInt и StrToFloat соответственно. Нет гарантии, что пользователь правильно введет числа в поля ввода, поэтому при преобразовании может возникнуть исключительная ситуация «Edit1 is not valid floating point value» в случае применения функции StrToFloat или «Edit1 is not valid integer value» в случае использования функции StrToInt. Давать возможность видеть пользователю сообщения такого рода – это плохо разработанная программа, дурной стиль программирования. Все сообщения должны быть понятны пользователю и должны относиться к решаемой задаче, а не к внутренним проблемам программы. Избежать этой ситуации можно несколькими способами. Один их них, когда можно запретить ввод нечисловых данных в поля ввода. Для защиты полей от ввода нечисловых данных могут быть предложены два варианта обработчика события OnKeyPress для полей ввода, реализованных компонентом Edit. Оба из них предполагают, что числа вводят не в научной нотации. Первый вариант разрешает ввод только цифр, десятичного разделителя и допускает использование клавиши «BackSpace». Этот вариант обработчика события «Нажатие одиночной клавиши» (OnKeyPress) может быть использован при работе с положительными целыми и вещественными числами.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if Sender is TEdit then begin
  if not((key in ['0'..'9']) or (key = chr(vk_back)) or
          (key=DecimalSeparator)) then
      key := #0;
  if (key = DecimalSeparator) then
   if (pos(key, (sender as TEdit).Text)>0) then
      key := #0;
 end;   end;
```

Второй вариант обработчика события OnKeyPress помимо того, что разрешает ввод только цифр, десятичного разделителя, допускает использование клавиши «BackSpace», еще разрешает вводить знак «-», т. е. он может быть использован для ввода как положительных, так и отрицательных чисел. Обработчик отслеживает появление десятичного разделителя. Если он находится в первой позиции, перед ним подставляется символ нуля «0». Если в набранном тексте присутствует знак «-», он устанавливается в первую позицию. Не допускается дублирование десятичного разделителя и знака «-».

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if Sender is TEdit then
  if not( (key in ['0'..'9']) or (key = chr(vk_back)) or
          (key=DecimalSeparator) or (key = '-')) then
          // ожидание ввода числа от 0 до 9, символа десятичного
          // разделителя, знака минус или нажатия клавиши BackSpace
            key := #0
  else begin
    if (key = DecimalSeparator) then begin
      if (pos(key, (sender as TEdit).Text)>0) then begin
          //если это уже не первое вхождение
        key := #0;  exit;
      end;
      if  ((Sender as TEdit).SelStart=0) then begin
          //вставка 0, если десятичный разделитель помещен
          //в первую позицию
        (Sender as TEdit).Text:='0'+DecimalSeparator +
                                  (Sender as TEdit).Text;
```

```
        (Sender as TEdit).SelStart:=(Sender as TEdit).SelStart+2;
        key:=#0;
      end;
    end;
    if (key = '-') then begin
      if (pos(key, (sender as TEdit).Text)>0) then  begin
        //если повторное нажатие знака минус
        key := #0;
        (Sender as TEdit).Text:= copy((Sender as TEdit).Text, 2,
          length((Sender as TEdit).Text)-1);
          //удаление предыдущего знака минус
        exit;
      end;
      if (copy((Sender as TEdit).Text, 1, 1) <> '-') then begin
        //если знака минус не было
        (Sender as TEdit).Text := '-' + (Sender as TEdit).Text;
        (Sender as TEdit).SelStart :=
                                length((Sender as TEdit).Text);
          //вставка знака минус на первую позицию
        key := #0;
      end;
    end; //if (key = '-')
  end; //else
end;
```

2.2. Блокировка вставки текста в поля ввода

Для защиты полей ввода Edit от вставки текстовых значений при помощи системного всплывающего меню необходимо запретить появление

стандартного меню при щелчке правой кнопкой мыши над полем ввода Edit. Для этого необходимо разместить на форме компонент PopupMenu и назначить его полям ввода, выбрав в Инспекторе Объектов на закладке Events или Properties пункт PopupMenu. В этом случае назначение пустого всплывающего меню перекроет возможность появления системного контекстного меню.

Для предотвращения вставки текста в поля ввода с использованием сочетания клавиш Ctrl+V или Shift+Insert нужно создать обработчик для события OnKeyDown или OnKeyUp:

```
procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if (key=vk_insert) then key:=0;
end;
```

2.3. Использование блока try…except для проверки ввода

Одним из вариантов повышения безопасности программы является использование блоков try…except для обработки исключительных ситуаций (ИС). В разделах 2.1 и 2.2 был предложен механизм создания обработчиков событий OnKeyPress и OnKeyDown/OnKeyUp для коррекции вводимых пользователем данных в поля ввода. Но можно не ограничивать возможности пользователя при вводе числовых значений. Для этого нужно использовать в коде программы блок try…except в том месте, где введенные пользователем данные конвертируются в целые (StrToInt) или вещественные (StrToFloat) числа. Обработчики событий OnKeyPress и OnKeyDown/OnKeyUp в этом случае не нужны. Ниже приведен фрагмент кода, в котором отслеживается и обрабатывается ИС, связанная с ошибкой преобразования.

```
try
r := StrToFloat(Form1.Edit1.Text);//здесь может возникнуть ИС
```

```
except
on EConvertError do
   begin
        //Здесь блок обработки ИС
        //Например, вывод сообщения об ошибке
      ShowMessage('В поле введено не числовое значение!');
      exit;//Выходим из подпрограммы
   end;
end;
```

2.4. Использование функций TryStrToFloat и TryStrToInt для проверки ввода

В старших версиях Delphi введены две полезные функции: `TryStrToInt` и `TryStrToFloat`. Эти функции в качестве аргументов используют строку, которую нужно конвертировать в целое или вещественное значение и переменную, куда будет записан результат в случае успешного преобразования. Если преобразование неуспешно – функция возвращает false.

```
function TryStrToInt(const S: string; out Value: Integer): Boolean;
function TryStrToFloat(const S: string; out Value: Double):
Boolean; overload;
```

Фрагмент кода с использованием функции `TryStrToFloat` приведен ниже. Следует заметить, что в качестве аргумента функция принимает вещественное число с десятичным разделителем, установленным в данной операционной системе.

```
Var Mu:double;
begin
if(TryStrToFloat(EditMu.Text, Mu)=false) then
 begin
  ShowMessage('Правильно введите параметр Mu.');
  exit;
 end;  end;
```

2.5. Кнопки для запуска/остановки расчета

Часто программисты при разработке расчетных программ блокируют поля ввода от изменения пользователем значений в них на время проведения расчета. Для этого при запуске расчета поля делают недоступными (Edit1.Enabled:= false;) или устанавливают для них свойство ReadOnly (Edit1.ReadOnly:= true;). Оставляем этот момент на рассмотрение разработчика. Это возможная, но не обязательная мера, поскольку, пока программа считает, она не проверяет поля ввода. А вот на работу кнопок управления обязательно нужно обратить внимание. Если расчет методом Монте-Карло достаточно хорошо обеспечен, т.е. проводится с большой статистикой, программа может считать несколько секунд. В этом случае разработчик должен предусмотреть наличие кнопки, позволяющей прервать расчет. Закономерно предположить, что перед запуском программы на расчет кнопка СТАРТ активна, а кнопка СТОП недоступна для использования. Когда идет расчет, кнопку СТАРТ делают недоступной, а кнопку СТОП доступной к использованию. По окончании расчета или в случае, когда расчет был прерван, кнопки возвращаются в первоначальное состояние. Один из вариантов работы с кнопками приведен в виде фрагмента кода ниже. В этом случае используется одна кнопка, которая переименовывается в зависимости от действий программы.

```
if Button1.Caption = 'Старт' then begin
    Button1.Caption := 'Стоп';
     ProcRun; //Запуск расчета, проверка не нажата ли кнопка СТОП
  end
  else if Button1.Caption = 'Стоп' then begin
    Button1.Caption := 'Старт';
      ProcStop; //Остановка расчета, запоминание статистики
  end;
```

2.6. Проверка введенных данных на корректность

При нажатии кнопки СТАРТ, прежде чем запустить расчет, необходимо проверить введенные данные на корректность. На рис. 3 приведен фрагмент формы с входными параметрами для моделирования случайных чисел с Бета распределением.

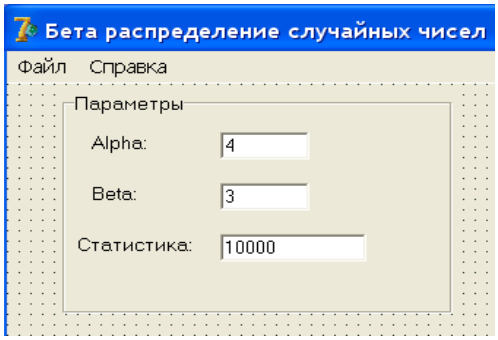


Рис. 3. Поля ввода параметров расчета

В этом случае для проверки введенных числовых данных на корректность может быть использован следующий код:

```
if(StrToFloat(Alpha.Text)<=0) or (StrToFloat(Alpha.Text)>500) then
 begin
  ShowMessage('Ограничение: 0< Alpha <= 500');
  exit;
 end;
if(StrToFloat(Beta.Text)<=0 ) or (StrToFloat(Beta.Text)>500)  then
 begin
  ShowMessage('Ограничение: 0<  Beta <= 500');
  exit;
 end;
```

3. Разработка расчетного блока программы

При разработке расчетного блока программы следует выделить несколько важных моментов. К ним относится работа со статистикой, взаимодействие

пользователя с программой в момент расчета, работа с памятью, организация многопоточности, динамические библиотеки.

3.1. Число экспериментов

Если программа выполняет расчеты методом Монте-Карло, то известно, что чем больше количество экспериментов (статистика расчета), тем точнее результат моделирования. Пользователь может задать значение статистики в поле ввода, после чего это значение используется как количество итераций при расчете. Можно в программе объявить переменную, ответственную за статистику, как `var Stat : integer;` В этом случае максимальное значение статистики = 2147483647. Это легко проверить, используя следующий код:

```
var stat:integer;
begin
ShowMessage(IntToStr(High(stat)));
end;
```

В этом случае при разработке кода можно использовать оператор цикла:

```
for counter := initialValue to finalValue do statement
```

Оператор цикла-счетчика `for` использует в качестве переменной цикла `counter` переменную `Ordinal type` (порядковый тип), к которому относится `integer`. Но в этом случае перед использованием значения статистики в программе необходимо проверить введенное пользователем значение, чтобы оно было меньше максимально допустимого для типа `integer`.

Можно использовать другой подход: объявить переменную статистики как

```
var stat:int64;
```

В этом случае максимальное значение переменной 9223372036854775807 удовлетворит даже самые смелые надежды пользователя, хотя проверка введенного значения, видимо, тоже необходима, поскольку никто не запрещает пользователю ввести в поле ввода что угодно. При работе со статистикой, объявленной как `int64`, нужно иметь в виду несколько моментов. Функция

конвертирования строкового значения одинаково хорошо работает как с integer, так и с int64, но для обратного преобразования существуют две разные функции: StrToInt (const S: string): Integer; и StrToInt64 (const S: string): Int64;. В случае использования int64 невозможно использование цикла-счетчика for. Применение этого цикла приведет к ошибке. В старших версиях компилятор Delphi указывает на ошибку. Чтобы этого не произошло, при разработке кода расчетного блока программы цикл for целесообразно заменить на цикл while с инкрементированием счетчика в коде программы. Например:

```
var
i,Stat : int64;//i – счетчик, Stat -  статистика
Begin
  i := 0;    //обнуляем счетчик цикла
  Stat := StrToInt64(Form1.Edit1.Text);//Значение статистики
  while(i < Stat)
  Begin
    . . .  {тело цикла} . . .
    inc(i);
  End;    End;
```

3.2. Взаимодействие программы с пользователем во время расчета

Расчеты, использующие методы Монте-Карло, могут занимать от нескольких секунд до значительно большего времени в зависимости от быстродействия компьютера и заданной для расчета статистики. Пока программа считает, пользователя необходимо чем-то занять. Хорошим стилем программирования является подсчет времени, которое требуется программе на расчет одной итерации, и экстраполяция на заданное количество итераций. В этом случае в интерфейсе следует предусмотреть поля для вывода времени начала расчета, времени, требуемого для расчета, которое может обновляться через определенное количество просчитанных итераций, и время окончания расчета. В качестве другого варианта можно использовать компонент Delphi,

который называется ProgressBar и находится на вкладке Win32 палитры компонентов среды разработки Delphi. При работе с ProgressBar задается начальное и максимальное значение позиции:

```
ProgressBar1.Position := 0;
ProgressBar1.Max := Stat;
```

а в расчетном цикле формируется текущее состояние

```
i := 0;
  while i <= Stat-1 do
  begin
     if i mod 100 = 0 then
      begin
        Application.ProcessMessages;
        ProgressBar1.Position:=ProgressBar1.Position + 100;
        if FlagStop = true then
        begin
          Stat := i+1;
          Edit3.Text := IntToStr(Stat);
          break;
        end;
       end;
           . . .  {тело цикла} . . .
           inc(i);
    end;
```

При работе с ProgressBar следует иметь в виду, что свойства `Position` и `Max` имеют тип `integer`. Приведенный выше фрагмент кода требует разъяснений. Переменная `FlagStop` имеет тип `boolean` и получает значение `true` в тот момент, когда пользователем нажата кнопка СТОП, прерывающая расчет. Проверка, нажата ли кнопка СТОП, осуществляется на каждой сотой итерации. В случае завершения расчета или его прерывания результаты расчета нормируются на значение статистики. Если нажата кнопка СТОП, цикл расчета

прерывается, значение статистики в этом случае равно количеству посчитанных итераций. Это значение присваивается переменной `Stat` и визуализируется в поле `Edit3.Text`. Когда идет расчет – процессор компьютера занят, программа ничего «не видит» и «не слышит». Для того, чтобы прервать вычисления и передать операционной системе управление для просмотра очередей сообщений (нужно увидеть, что нажата кнопка СТОП), используется оператор `Application.ProcessMessages` на каждой сотой итерации.

3.3. Использование потоков

Использование потоков имеет смысл для организации приложения, в котором можно производить одновременно вычисления и какую-то другую полезную работу, например, работать с результатами предыдущих вычислений. Допустим, имеется основная форма для работы с результатами Form1 и немодальная форма Form2, которая вызывается из основной формы и содержит поля ввода параметров и статистики, кнопки СТАРТ и СТОП. При нажатии кнопки СТАРТ осуществляется проверка введенных пользователем данных и запускается расчет, для чего создается новый поток. В этом потоке производятся все вычисления и проверяется нажатие кнопки СТОП, чтобы прервать расчет до его завершения. При такой организации вычислений в момент расчета у пользователя есть возможность работать с Form1, просматривая и анализируя результаты предыдущих вычислений. Пошаговый алгоритм реализации расчетов с использованием потока может выглядеть так.

1. На главной форме приложения Form1 (Unit1) размещаем главное меню, одной из функций которого будет вызов формы Form2 (Unit2) для заполнения полей ввода и запуска расчета, и компоненты, предназначенные для когнитивной визуализации результатов расчетов: поля, сетки, графики.

2. Добавляем в проект новую форму Form2, размещаем поля для ввода данных, кнопки СТАРТ и СТОП.

3. File -> New -> Other -> вкладка New -> Thread object. В проект будет добавлен новый файл (назовем его Unit3.pas). В этом модуле Delphi сформирует

заготовку для потока: класс потока (назовем его TThread) и Execute – основной метод класса TThread. В метод Execute помещаем код, реализующий расчетный блок с использованием метода Монте-Карло.

4. В обработчике события OnClick кнопки СТАРТ модуля формы Form2 размещаем программный код для считывания введенных данных, их валидацию и запуск расчета, если данные корректны. При запуске расчета создаем новый поток. Булевская переменная в конструкторе указывает на то, как создавать поток – приостановленным или запускать сразу.

```
//в разделе глобальных переменных объявляем
//переменную-экземпляр класса потока
Var th : TThread;
//после нажатия СТАРТ и успешной валидации
th := TThread.Create(false);//создаем приостановленный поток
th.FreeOnTerminate := true; // деструктор потока вызывается
  // автоматически по завершении потока
th.Priority := tpLower; //низкий приоритет для потка
th.Resume;//явный запуск потока
```

Поток останавливается по окончании расчета или принудительно при нажатии пользователем кнопки СТОП (флаг FlagStop получает значение true). Отличительной чертой специального класса Tthread, предоставляемого Delphi для реализации потоков, является гарантия безопасной работы с библиотекой визуальных компонентов VCL. Delphi предоставляет программисту метод Synchronize для безопасного вызова методов VCL внутри потоков. Во избежание конфликтных ситуаций, метод Synchronize дает гарантию, что к каждому объекту VCL одновременно имеет доступ только один поток. Допустим, мы создали поток вычислений и хотим в нем в конце каждой сотой итерации цикла выводить статус выполнения расчета в ProgressBar, расположенный на форме Form2. Значения начальной и максимальной позиции ProgressBar заданы до запуска расчетного блока в потоке.

```
Procedure TThread.Execute;
Begin
. . .
While(i < Size) //Основной цикл внутри потока
 Begin
  . . .
  If i mod 100 = 0 then
   Synchronize(Form2.ShowStatus);
  . . .
 End;

Procedure TForm2.ShowStatus;
Begin
 Application.ProcessMessages;
 ProgressBar1.Position:=ProgressBar1.Position + 100;
 if FlagStop = true then
  begin
     Stat := i+1;
     th.Terminate;
  End;
End;
```

И последний штрих: в интерфейсной части модуля формы Form2 нужно добавить ссылку на модуль Unit3, содержащий описание и методы класса потока, а в исполнимой части модуля Unit3 потока добавить оператор `Uses Unit2;` Этот оператор нужен, поскольку метод синхронизации принадлежит модулю Unit2, а в модуле потока есть на него ссылка.

3.4. Работа с памятью

При разработке расчетных программ всегда нужно очень аккуратно использовать память. Для динамического распределения памяти используются динамические и разреженные массивы. Динамические массивы не имеют фиксированной размерности и длины. Память под массив распределяется, когда

массиву присваивается значение или вызывается процедура SetLength**.** Объявление динамического массива отслеживает конструктор формы. В программе хранится указатель на массив. Память под динамический массив автоматически освобождается, когда длина массива устанавливается к 0, динамические массивы нулевой длины имеют значение **NIL.**

Еще одним способом динамического распределения памяти является создание разреженных массивов. В разреженных массивах фактически имеются не все элементы. Такой массив может потребоваться, когда размеры массива превышают размер оперативной памяти машины или тогда, когда одновременно в памяти не нужно хранить все элементы массива. Примером является электронная таблица. Она может быть как угодно велика. В электронных таблицах одновременно используются не все ячейки таблицы, поэтому она реализована в виде разреженного массива. Память под каждую ячейку выделяется по мере необходимости. Имеется три способа создания разреженных массивов: связанный список, двоичное дерево и массив указателей.

3.4.1. Динамический массив структур

Рассмотрим пример решения практической задачи с использованием динамического массива структур. Задача формулируется следующим образом. Есть набор текстовых файлов с дискретными целыми значениями X и вещественными значениями Y для каждого X. Необходимо просмотреть любое количество файлов и построить график результатов, в котором для каждой точки X вывести среднее арифметическое значение Y по всем просмотренным файлам. В качестве десятичного разделителя при записи вещественного значения используется точка.

Разместим на форме 3 кнопки : кнопка Input – «Ввод файлов», Show – «График результатов», Clear – «Очистка»; компонент Tchart – для визуализации результатов работы программы; компонент Tlabel – для того, чтобы вывести

число уникальных значений X (число точек графика); компонент TopenDialog для выбора файла. Вид интерфейса программы показан на рис. 4.

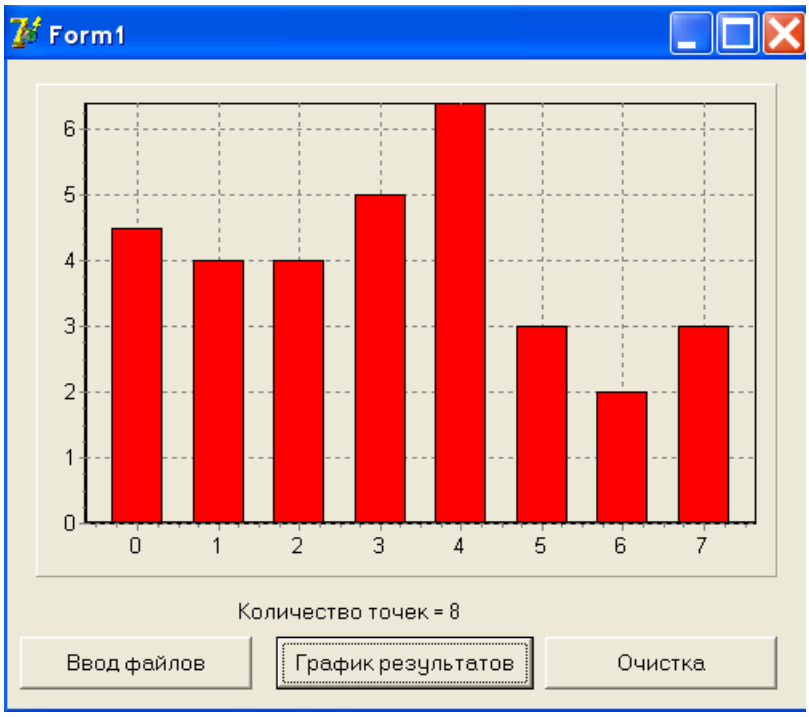


Рис. 4. Вид программы для расчета среднего значения

Пользователь нажимает на кнопку «Ввод файлов», чтобы прочитать файл, и делает это столько раз, сколько файлов должно быть прочитано. После этого нажатие кнопки «График результатов» инициирует отрисовку графика результатов. Перед обработкой следующей серии файлов пользователю необходимо нажать кнопку «Очистка». Последовательность разработки.

1. Объявим структуру для хранения значения X, суммы всех Y для этого X и количество точек, у которых было найдено это значение X.

```
point = record
     X:integer;        Y:double;        Count:Cardinal;
     end;
```

2. Объявим глобально динамический массив типа описанной выше структуры и переменную для хранения количества точек с уникальным значением X.

```
var
  Form1: TForm1;
  Mas: array of Point;
  TotalPoints:integer;
```

3. Разработаем процедуру обработки события – нажатие кнопки «Очистка».

```
procedure TForm1.ClearClick(Sender: TObject);
begin
 TotalPoints:=0;
 Label1.Caption:='Количество точек = 0';
 Chart1.Series[0].Clear;
 If Assigned(Mas) then SetLength(Mas,0);
end;
```

В этой процедуре обнуляем переменную для хранения количества точек, чистим график и проверяем, если память под динамический массив распределена, то ее освобождаем. Последняя проверка нужна обязательно. Если пользователь начнет работу с программой с того, что нажмет на кнопку «Очистка», то без такой проверки попытка освободить нераспределенную память приведет к возникновению ИС – ошибки адресации.

4. Обработчик события «Создание формы».

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 ClearClick(Nil);// вызов процедуры очистки как метода формы
end;
```

5. Обработчик события «Закрытие формы».

```
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
If Assigned(Mas) then SetLength(Mas,0);
end;
```

Освобождаем память, предварительно убедившись, что она была распределена под массив.

6. Обработчик события – нажатие кнопки «Ввод файлов».

```
procedure TForm1.InputClick(Sender: TObject);
 var F:TextFile; i,x:integer; y:double; flag:boolean;
 begin
  if not OpenDialog1.Execute then exit; // отмена выбора файла
  AssignFile(F,OpenDialog1.FileName); // связывание файла
                                      с  //файловой переменной
  reset(F); // открытие текстового файла для чтения
  while not EOF(F) do // цикл пока не закончится файл
   begin
    Read(F, x);  Readln(F, y); // чтение X и Y с дес. разд. точка
    if(TotalPoints=0) then // если точек еще нет
     begin
      SetLength(Mas,TotalPoints+1);// первая точка,распределить
                                   //память под один элемент
      Mas[0].x:=x; // занести считанное из файла x
      Mas[0]. y:=y; // занести считанное из файла y
      Mas[0].count:=1; // с этим x – одна точка
      inc(TotalPoints); // инкременировать счетчик точек
     end else //  какие-то точки уже есть
     begin
      flag:=false;
      for i:=0 to High(Mas) Do // цикл по всему массиву точек
        if Mas[i].x = x then // есть уже точка с таким x
         begin
          Mas[i].y:=Mas[i].y + y; //добавить y в сумму всех y
          inc(Mas[i].Count); // увеличить счетчик точек с этим x
          flag:=true; // перебросить флаг – точка была найдена
         end;
       if flag=false then // новая точка, такого x  в массиве нет
        begin
```

```
          SetLength(Mas,TotalPoints+1);//запросить память еще
                                        под //один элемент массива
          Mas[TotalPoints].x:=x; //занести x
          Mas[TotalPoints].y:=y; //занести y
          Mas[TotalPoints].count:=1; //такая точка пока одна
          inc(TotalPoints); // инкременировать счетчик точек
        end;
      end;
    end;
    CloseFile(F); // закрыть текстовый файл
end;
```

Некоторые фрагменты кода процедуры `InputClick` следует объяснить. Первый оператор `SetLength(Mas,TotalPoints+1)`, который исполняется в случае, когда `TotalPoints=0`, динамически запрашивает выделение 20 байтов памяти под структуру `point`. Оператор `SetLength(Mas,TotalPoints+1)` выполняется, когда массив уже существует и прочитано из файла новое уникальное значение X. В этом случае программа просит «довыделить» еще 20 байтов памяти под размещение новых данных. Компилятор поступает следующим образом: выделяется новый непрерывный кусок памяти, увеличенный по сравнению с прежним на 20 байтов, данные из предыдущего размещения переписываются в отведенную новую память, после чего память предыдущего размещения массива освобождается. Таким образом, при работе с динамическими массивами следует помнить, что под него выделяется непрерывный блок памяти. Несколько иначе осуществляется работа с двумерными динамическими массивами. Они реализуются как массивы указателей на массивы. Например, вызов `SetLength(A, 2, 4)` создает три массива: A – массив указателей размерностью 2 на блоки памяти, выделенные под массивы A[0] и A[1] соответственно и A[0], A[1] – два массива размерностью по 4 для размещения данных. Под A[0] и A[1] выделяются два непрерывных блока памяти. Такая организация позволяет создавать непрямоугольные массивы.

7. Обработчик события – нажатие кнопки «График результатов».

```
procedure TForm1.ShowClick(Sender: TObject);
 var i:integer;
 begin
  Chart1.Series[0].Clear;
  if not Assigned(Mas) then exit;
  for i:=0 to High(Mas) do //цикл по всему динамич. массиву
  Chart1.Series[0].AddXY(i,Mas[i].y/Mas[i].Count);
  Label1.Caption:= 'Количество точек = '+ IntToStr(High(Mas)+1);
 end;
```

3.4.2. Двухсвязный список

Решим предыдущую задачу с использованием «разреженного массива», реализованного в виде двухсвязного списка.

Здесь следует рассматривать «разреженный массив» как название определенного способа хранения информации в памяти, а не как языковую конструкцию Delphi. Двухсвязный список может быть реализован в виде стуктуры, которая в своем составе помимо переменных имеет два указателя: на предыдущий элемент и на следующий (рис. 5). Наличие указателей на предыдущий и следующий элементы позволяет просматривать весь информационный массив (список) в двух направлениях. Каждый элемент списка произвольно расположен в памяти, которая выделяется ему при создании.

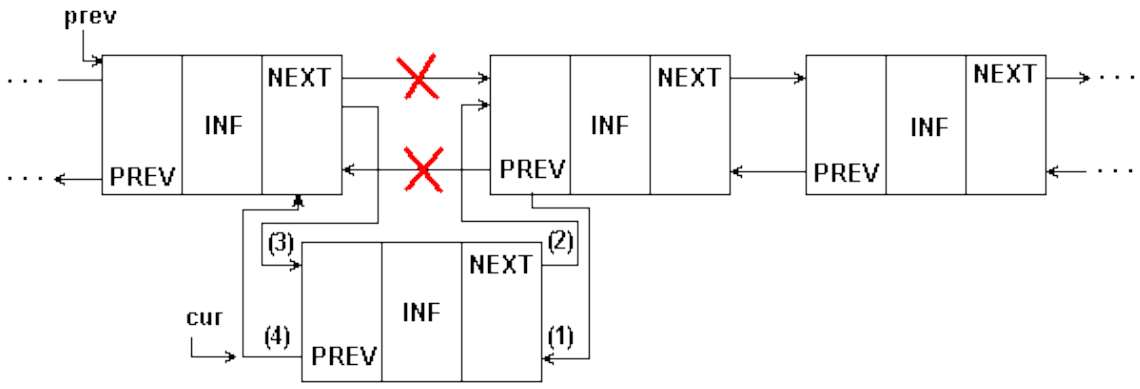


Рис. 5. Структура двухсвязного списка

На рисунке изображен механизм вставки нового элемента. В месте вставки существующая связь разрывается. Вставляемый элемент начинает указывать своим предыдущим концом на следующий места разрыва, а своим следующим элементом – на предыдущий места разрыва. Такая организация списка позволяет существенно ускорить алгоритмы поиска и вставки за счет возможности двустороннего просмотра массива.

Структура, реализующая двухсвязный список:

```
type CellPointer = ^cell;
  cell = record
    index: double;      // уникальное значение X
    value: double;      //сумма Y значений для X
    count: integer;     //количество точек со значением X
    next: CellPointer; //указатель на следующий элемент
    prev: CellPointer; //указатель на предыдущий элемент
  end;
```

При реализации поставленной задачи целесообразно создать класс, в котором хранить указатели на первый и последний элементы всего списка. Для этого класса можно разработать метод, осуществляющий проверку считанного из файла текущего значения X и вставку нового элемента в список, если считанное значение X встретилось впервые, или увеличение значений value и count в случае, если считанное значение X уже встречалось. В последствии класс может быть расширен добавлением функций удаления, сортировки или поиска.

```
type
  TLinkedList = class(TObject)
    First, Last: CellPointer; //указатели на первый и последний
                              //элементы списка
```

```
    constructor TLinkedList;
    function Store(info: CellPointer): CellPointer;
  end;
```

Конструктор класса `TlinkedList`:

```
constructor TLinkedList.TLinkedList;
begin
  new(First);
  new(Last);
end;
```

Метод `Store` класса `TLinkedList` для проверки X и сохранения информации реализован в виде функции:

```
function TLinkedList.Store(info: CellPointer): CellPointer;
var old, top: CellPointer;
  iter: CellPointer;
begin
  old:= nil;
  done:= false;
  new(iter);
  if First = nil then begin //первый элемент списка
    info^.next:= nil;
    new(Last);
    new(First);
    First^.index:= info^.index;
    First^.value:= info^.value;
    First^.count:= 1;
    First^.next:= nil;
    First^.prev:= nil;
    Store:= info;
  end
  else begin //First <> nil
```

```
if First^.next = nil then begin
  if First^.index = info^.index then begin //при
     //совпадении элемент не добавляется, а изменяется
    First^.count := First^.count + 1;
    First^.value := First^.value + info^.value;
  end
  else begin       //вставка нового
    First^.next := info;
    info^.prev  := First;
    info^.next  := nil;
    Store:= info;
  end;
end //First^.next = nil
else begin
  iter^.index:= First^.index;
  iter^.value:= First^.value;
  iter^.count:= First^.count;
  iter^.next := First^.next;
  iter^.prev := First^.prev;
  while(iter<>nil) do begin
    if iter^.index = info^.index then begin
      iter^.count:= iter^.count + 1;
      iter^.value:= info^.value + iter^.value;
      Store:= info;
      break;
    end;
    if iter^.next=nil then begin
      iter^.next := info;
      info^.prev := iter;
      info^.next := nil;
      Store:= info;
      break;
    end;
    iter := iter^.next;
  end;//while
```

```
    end; //else of First^.next = nil
  end; //else of start <> nil
end; //function  Store
```

Функция проверяет наличие в списке элемента со значением `index` = X. Если элемент присутствует, изменяются его параметры. Если элемент с таким индексом отсутствует, под него выделяется память и он добавляется в конец списка. Функция возвращает указатель на элемент списка.

Заполнение структуры `cell` и вызов метода `Store` класса `TLinkedList`:

```
      LList: TLinkedList;  // объявление глобальной переменной
     tempL: CellPointer;
      . . . // часть кода, отвечающая за чтение из текстового файла
      new(tempL);           //создание и формирование структуры
      tempL^.index:= X; // индекс (X)
      tempL^.value:= Y; // значение Y
      tempL^.count:= 1;   // количество точек с индексом X
      tempL^.next:= nil;  // указатель на следующий элемент
      tempL^.prev:= nil;  // указатель на предыдущий элемент
      LList.Store(tempL);  // сохранение элемента в памяти
```

Обращение к сформированному списку для добавления значений точки на график:

```
var start: CellPointer;
      new(start);
      start:= LList.First; //получение указателя на первый элемент
      while start<>nil do begin //перебор всех элементов
       Chart1.SeriesList[0].AddXY(start^.index,
                                  start^.value/start^.count);
```

```
    if start.next <> nil then
     start:= start.next
     else break;
  end;
```

3.4.3. Массив указателей

Создание «разреженного массива» в виде массива указателей покажем на примере решения конкретной задачи. Рассматривается задача моделирования непрерывной случайной величины ξ с распределением вероятности: $\rho(x) = 1/x^2$ на интервале (1, ∞). Методом обратных функций получаем формулу для розыгрыша случайной величины ξ = 1/random. Для визуализации результатов моделирования получаемых значений случайной величины ξ по умолчанию выбирается шаг группировки равный 0.1, это значит, что все значения случайной величины, отличающиеся друг от друга меньше, чем на 0.1, заносятся в одну ячейку гистограммы. Интерфейс программы показан на рис. 6.

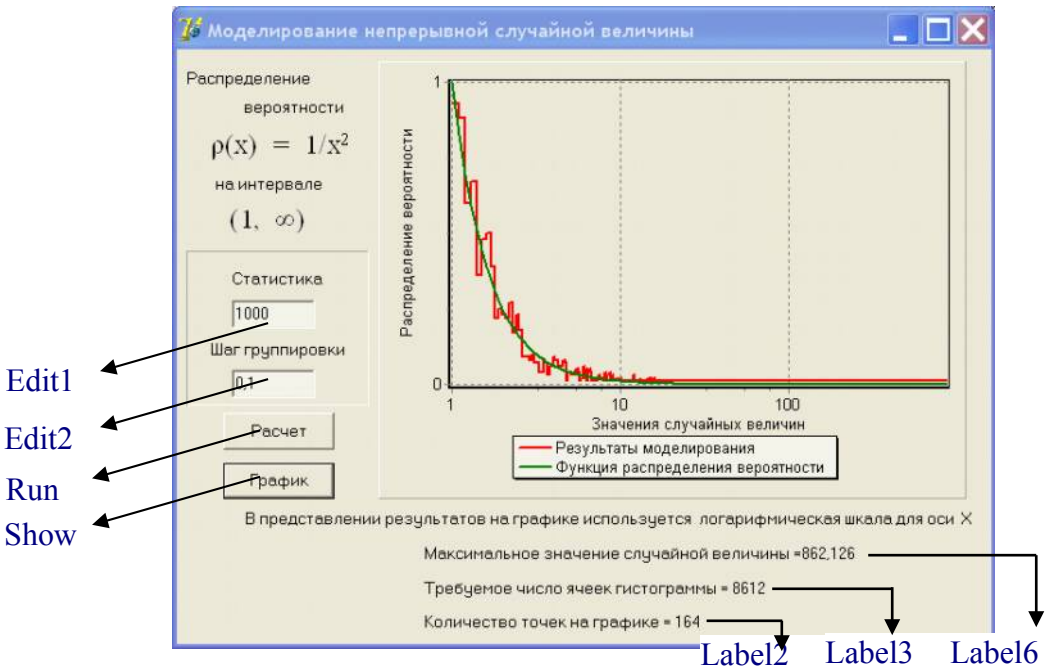


Рис. 6. Интерфейс программы моделирования случайной величины

На рисунке показан результат выполнения программы. При увеличении статистики разница между требуемым количеством ячеек гистограммы и количеством ячеек, которые используются в «разреженном массиве» («количество точек на графике»), еще более разительна.

Создадим класс для того, чтобы для каждой уникальной точки хранить значения индекса и количество точек, попавших в этот индекс.

```
type
  TOne= class (TObject)
   Find:integer;
   Fvalue:integer;
   Constructor Create(Fi,Fv:integer);
   end;
```

Что здесь имеется в виду под индексом `Find`? Пусть есть интервал от **a** до **b .**

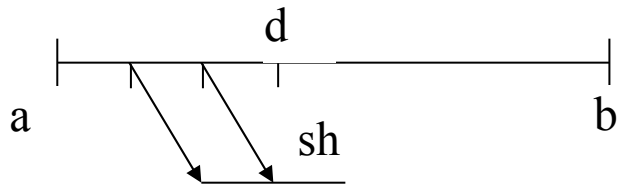


Рис. 7. Индекс точки

Чтобы узнать, в какой интервал попадает случайная точка $a \le d \le b$, необходимо вычислить (d – a) / sh. Для распределения, выбранного для этого примера, интервал начинается с 1, поэтому индекс гистограммы равен trunc ((ξ -1)/sh). Описание глобальных переменных и конструктор класса Tone:

```
var
  Form1: TForm1;
  MyList:TList; // Объявление объектной переменной, списка
                                    //указателей
  flag:boolean;  // Флажок
  stat:integer;  // Статистика (берется из Edit1)
  sh,ma:single; //  sh - Шаг гистограммы (берется из Edit2)
                //  ma – максимальное значение случ. в-ны ξ
Constructor TOne.Create(Fi,Fv:integer);
```

```
begin
    inherited Create;
   FInd:=Fi; FValue:=Fv;
   end;
```

Обработчик события OnCreate формы Form1:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
MyList:=TList.Create; // создание списка указателей
Label2.Visible:=false; Label3.Visible:=false;
Label6.Visible:=false;
end;
```

Обработчик события нажатие кнопки Run («Расчет»):

```
procedure TForm1.RunClick(Sender: TObject);
var Temp:TOne; //Переменная,чтобы использовать свойства класса TOne
    d,t:single;
    i,j,indArr:integer;
begin
 Label2.Visible:=false; Label3.Visible:=false;
 Label6.Visible:=false;
 MyList.Clear; // очистка списка
 stat:=StrToInt(Edit1.Text);  sh:=StrToFloat(Edit2.Text);
 ma:=1; // нач. значение для поиска максимума
 Randomize;
 for i:=1 to stat do begin // очень большая статистика не нужна
      t:=random;
      if not (t=0) then d:=1/t else d:=1;
      indArr:=trunc((d-1)/sh); // индекс
      if d>ma then ma:=d; // максимальное значение случ. величины
      if MyList.Count=0 then  // если список пуст
       MyList.Add(TOne.Create(indArr,1)) //добавить в список
      else begin // не пустой список
       flag:=false;
       for j:=0 to MyList.Count-1 do begin  // цикл по
        всему     //списку, есть ли точка с инд indArr
```

```
          Temp:=MyList.Items[j]; // указатель из MyList на Temp
          if Temp.Find=IndArr then begin // indArr совпал
            inc(Temp.Fvalue); // добавить к-во точек
              flag:=true; // флаг - искали индекс и нашли
                    break;
             end;  // совпали
        end;  // цикл по всему списку
       if flag=false then  // индекс не нашли,пришло новое значение
        MyList.Add(TOne.Create(IndArr,1)); //добавить в список
     end;  // не пустой список
 end;  // цикл по статистике
 Label2.Visible:=true;  Label2.Caption:='Расчет закончен';
end;
```

Обработчик события нажатие кнопки Show («График»):

```
procedure TForm1.ShowClick(Sender: TObject);
var i:integer;
 Temp:TOne;
 x,y,y1:single;
begin
 Chart1.Series[0].Clear;   Chart1.Series[1].Clear;
 if MyList.Count>0 then begin
  Label2.Visible:=true;
  Label2.Caption:='Количество точек на графике ='+
    IntToStr(MyList.Count);
   for i:=0 to MyList.Count-1 do begin // просмотреть весь список
     Temp:=MyList.Items[i]; // очередной элемент списка
     x:=Temp.Find*Sh+1;
     y:=Temp.Fvalue/stat/sh; // моделируемое значение в точке x
     y1:=1/(x*x); // аналитическое значение функции в точке x
     Chart1.Series[0].AddXY(x,y,'',clTeeColor);
     Chart1.Series[1].AddXY(x,y1,'',clTeeColor);
  end;
  Label3.Visible:=true;
  Label3.Caption:='Требуемое число ячеек гистограммы = '+
   IntToStr(Trunc((ma-1)/sh)+1);
```

```
  Label6.Visible:=true;
  Label6.Caption:='Максимальное значение случайной величины ='+
   FloatToStrF(ma,ffFixed,6,3);
 end;  end;
```

Обработчик события OnClose формы Form1:

```
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
 MyList.Free; // освободить память, отведенную для списка
 Chart1.Series[0].Clear; // освободить память, отведенную под серию
 Chart1.Series[1].Clear;
end;
```

3.5. Динамические библиотеки

При разработке программы можно использовать динамические библиотеки (dll). При использовании dll следует учитывать следующие моменты:

- динамические библиотеки хранятся в памяти в единственном экземпляре;
- они используют адресное пространство вызывающей программы;
- не работают с глобальными переменными вызывающей программы;
- подключаются к основной программе во время исполнения (runtime);
- должны обрабатывать все исключительные ситуации внутри себя;
- учитывают регистр экспортируемых функций.

Чтобы создать dll-библиотеку в Delphi, необходимо последовательно выбрать File –> New –> Other –> DLL wizard. Delphi создаст заготовку для разработки библиотеки и откроет окно редактора кода. После ключевого слова library необходимо указать имя будущей библиотеки. В теле library реализовать необходимые функции и после ключевого слова exports перечислить подпрограммы, которые библиотека должна поставлять вызывающей программе. При реализации функций необходимо учитывать соглашение о

вызовах. Если прототип реализуемой функции выглядит как `function SomeFunc(X: SomeParam): SomeParam; stdcall;`, то соглашение `stdcall` (стандарт Windows) определяет: направление передачи параметров подпрограмме вызывающей программой – справа налево; передачу параметров через стек; очистку стека подпрограммой.

Для отладки dll необходимо задать host приложение, которое будет осуществлять вызов dll: Run –> Parameters, и, воспользовавшись кнопкой «Browse», загрузить в поле «Host Application» полное имя вызывающей программы. После компиляции проекта библиотеки Delphi создаст файл с расширением dll – это и есть динамическая библиотека. Вызвать подпрограмму из библиотеки можно явным или неявным способом. Механизм неявного вызова наиболее прост, т. к. выполняется автоматически и основан на имеющейся в приложении информации о вызываемых функциях и динамических библиотеках. Однако разработчик не имеет возможности влиять на ход загрузки dll. Если операционная система не смогла загрузить библиотеку, просто выдается сообщение об ошибке. Единственный способ повлиять на процесс загрузки – использовать секцию инициализации библиотеки. При неявном вызове Delphi откомпилирует код так, что библиотека будет загружаться во время запуска программы. Для повышения скорости загрузки и для экономии памяти библиотеку необходимо подключать динамически, т.е. явно. Это позволяет производить загрузку библиотеки только в тот момент, когда необходимо, после чего библиотеку можно выгрузить и освободить память.

Разработаем библиотеку Neuman.dll. Разместим в ней процедуру CalcNeuman для генерации методом Неймана псевдослучайного числа из экспоненциального распределения. Эту процедуру библиотека будет экспортировать.

```
library Neuman;
uses
  SysUtils, Classes, Math;
{$R *.res}
function ExpDistr(l, x: Double): double;
begin
  Result:= l * exp(-l*x);
end;

procedure CalcNeuman(l: double; var x: double; var y: double);
stdcall
begin
  while true do begin
    x:= random;
    y:= random;
    if y < ExpDistr(l, x) then break;
  end;
end;

exports   CalcNeuman;
begin
end.
```

В приведенном примере реализована также функция ExpDistr, которая используется внутри библиотеки для расчета экспоненциального распределения, и не может быть вызвана за ее пределами. При неявном вызове поцедуры CalcNeuman из библиотеки Neuman.dll описание ее в основной программе будет иметь вид:

```
procedure CalcNeuman(l: double; var x: double; var y: double);
stdcall; external 'Neuman.dll';
```

А вызов:

```
for i:= 1 to stat do begin
  CalcNeuman(0.76, x, y);
  ...
```

```
end;
```

В случае использования явного вызова объявление прототипа выглядит так:

```
type
  TFuncType = function (x : SomeType):SomeType; stdcall;
var
  dllhandle: THandle;
  MyFunc: TFuncType;
  error: DWORD;
```

И вызов:

```
  dllhandle:=LoadLibrary('Neuman.dll');
  if dllhandle <> 0 then begin //есть ли библиотека?
    @MyFunc:= GetProcAddress(dllhandle,
              'CalcNewman'); //получить // адрес экспортируемой
              процедуры
    if Addr(MyFunc) = nil then begin
      ShowMessage('CalcNewman в Neuman.dll не найдена.');
      FreeLibrary(dllhandle);
      Close;
    end;
  end
  else begin // Neuman.dll не найдена
    error:= GetLastError;
    if  ( (error= ERROR_DLL_NOT_FOUND) or
           (error = ERROR_MOD_NOT_FOUND) ) then
      ShowMessage(' Neuman.dll не найдена.');
  end;
```

Если по какой-либо причине библиотека отсутствует или в ней не найдена вызываемая функция, то разработчику следует предусмотреть какие-либо варианты дальнейшего функционирования программы, как, например, блокирование определенных возможностей или завершение работы.

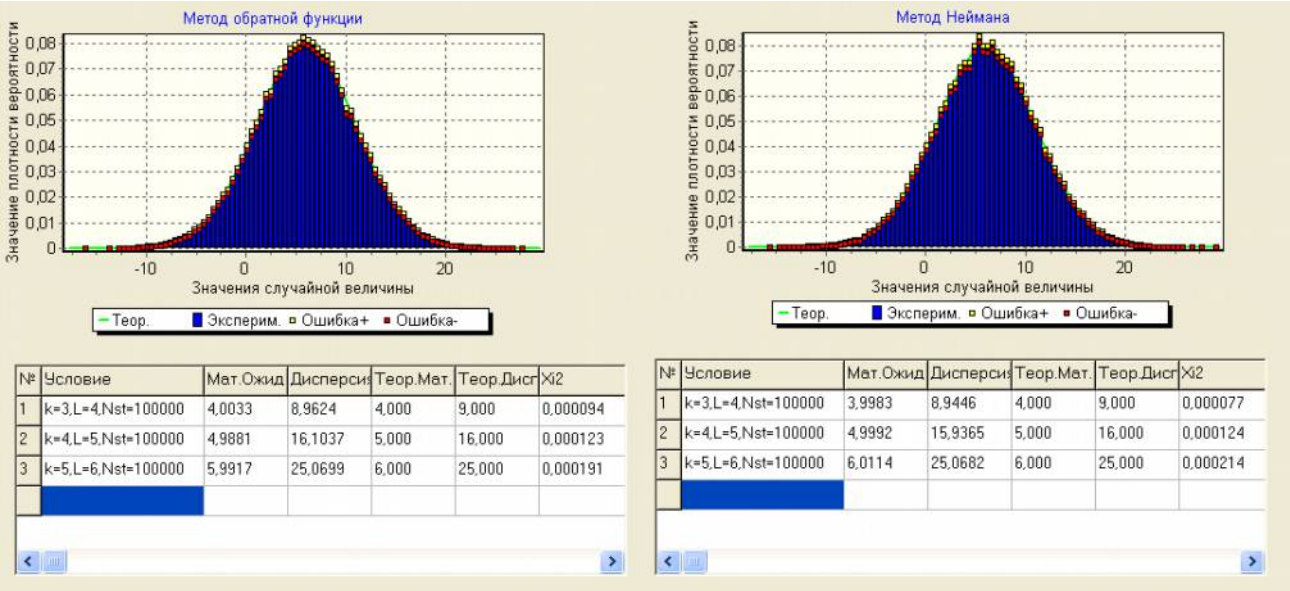
Для освобождения занимаемой библиотекой памяти необходимо по окончании работы программы выполнить проверку на наличие открытых библиотек и при

необходимости отключить их. Удобно назначить данную функцию обработчику события «Закрытие формы»:

```
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  if dllhandle <> 0 then FreeLibrary(dllhandle);
end;
```

4. Представление результатов моделирования

Вариаций на тему представления результатов моделирования может быть бесконечное множество. Но сразу нужно выделить необходимый минимум, который должен присутствовать при когнитивной визуализации результатов расчета. Целесообразно разместить на форме два компонента-графика (TChart), на каждом из которых отобразить рузультат моделирования методом Монте-Карло и результат аналитических вычислений. На первом графике можно показать результаты расчета с использованием метода Неймана, на втором с использованием метода обратной функции. Поскольку на графиках отображаются результаты последнего эксперимента, желательно иметь таблицу, в которую для каждого эксперимента выводить интегральные характеристики, такие как математическое ожидание, дисперсия, согласие по критерию χ^2 с аналитическими расчетами. Возможный вариант интерфейса для просмотра результатов вычислений представлен на рис. 8. Нет необходимости приводить код для заполнения таблиц интегральными значениями, полученными в результате расчета, и для формирования точек на графике. Никаких трудностей в реализации простейший вариант визуализации не представляет. Хотелось бы остановиться на некоторых моментах нестандартного представления результатов. К ним можно отнести механизм формирования отчетов и механизм, позволяющий анализировать и сравнивать серию полученных результатов расчетов.



№	Условие	Мат.Ожид	Дисперсия	Теор.Мат.	Теор.Дисп	Xi2
1	k=3,L=4,Nst=100000	4,0033	8,9624	4,000	9,000	0,000094
2	k=4,L=5,Nst=100000	4,9881	16,1037	5,000	16,000	0,000123
3	k=5,L=6,Nst=100000	5,9917	25,0699	6,000	25,000	0,000191

№	Условие	Мат.Ожид	Дисперсия	Теор.Мат.	Теор.Дисп	Xi2
1	k=3,L=4,Nst=100000	3,9983	8,9446	4,000	9,000	0,000077
2	k=4,L=5,Nst=100000	4,9992	15,9365	5,000	16,000	0,000124
3	k=5,L=6,Nst=100000	6,0114	25,0682	6,000	25,000	0,000214

Рис. 8. Когнитивная визуализация результатов

4.1. Разработка отчета

Графические и табличные результаты расчетов могут быть сохранены в виде отчета в Microsoft Word (вид отчета для дзета распределения показан на рис. 9). MS Word может быть использован в этом случае как COM (Component Object Model) сервер, поскольку как COM-объект он экспортирует группу функций, позволяющих с ним работать. Наша задача из Delphi обратиться к

Метод Неймана:

Параметр S	2
Ширина интервала	10
Статистика	1000

	Мат.Ожидание	Дисперсия	Критерий Хи-2
Теоретически	1,78061000681406	2,90873597974992	
Экспериментально	1,913	2,647431	0,999999998696396

График функции распределения

Рис. 9. Вид отчета в MS Word

интерфейсам, предоставляемым сервером. Для разработки отчета на форме, предназначенной для когнитивной визуализации результатов расчета, необходимо разместить кнопку «Отчет» и два компонента TWordApplication и TWordDocument, которые находятся на палитре компонентов Servers. При размещении на форме компонента TWordApplication в секцию uses интерфейсной части модуля Delphi добавляет системные модули WordXP и OleServer. Эти модули содержат Vtable таблицы, обеспечивающие интерфейс взаимодействия с сервером Word. Компонент TWordApplication обеспечивает соединение с сервером.

```
WordApplication1.Connect;
```

Установленное свойство Visible компонента TWordApplication позволяет сделать видимым процесс запуска сервера и формирования документа:

```
WordApplication1.Visible:= true;
```

После подключения к серверу автоматизации уже можно работать с документами: добавить новый документ и подключиться к нему, воспользовавшись компонентом TWordDocument, который обеспечивает доступ к документу:

```
WordApplication1.Documents.Add(EmptyParam, EmptyParam,
                               EmptyParam, EmptyParam);
WordDocument1.ConnectTo(WordApplication1.ActiveDocument);
```

Для формирования документа необходимо объявить переменные:

```
var     a, b : OleVariant;
   WordTable : OleVariant;
```

Переменные a, b используются для позиционирования в документе, WordTable представляет собой объект таблицы. Код, который будет приведен ниже, использовался для разработки отчета по моделированию распределения χ^2. Заголовок документа с пропуском строки до вывода текста и после:

```
a:= WordDocument1.Content.End_-1; // позиционирование в конец
b:= WordDocument1.Content.End_;
WordDocument1.Range(a, b).InsertAfter(#13+'Отчет по моделированию
случайной величины'+#13+#13);
```

Свойство End_ всегда указывает на конец документа. Создание таблицы:

```
a:= WordDocument1.Content.End_-1;
b:= WordDocument1.Content.End_;
WordTable := WordDocument1.Tables.Add(WordDocument1.Range(a,b),
                2, // количество строк таблицы
                2, // количество столбцов таблицы
                EmptyParam,
                EmptyParam);
WordTable.Cell(1,1).Range.Text := 'Интегральные результаты';
WordTable.Cell(1,2).Range.Text := 'График функции';
WordTable.Cell(2,1).Range.Text := 'Метод обратной функции' + #13
     + 'Статистика ' + IntToStr(stat) + #13
     + 'Дисперсия ' + FloatToStrF(variance,ffGeneral,7,5) + #13
     + 'Мода ' + FloatToStrF(mode,ffGeneral,7,5) + #13
     + 'Отклонение ' + FloatToStrF(deviation,ffGeneral,7,5)+#13;
```

Для размещения графика в таблице отчета используется функция копирования битового изображения в буфер обмена и вставка содержимого буфера обмена в документ:

```
Chart1.CopyToClipboardBitmap;
WordTable.Cell(2,2).Range.Paste;
```

Когда документ сформирован и визуализирован в MS Word, пользователь может закрыть его, сохранить, отформатировать, дополнить своими комментариями и т.д. Вид документа, для формирования которого был представлен код, показан на рис. 10.

Отчет по моделированию случайной величины

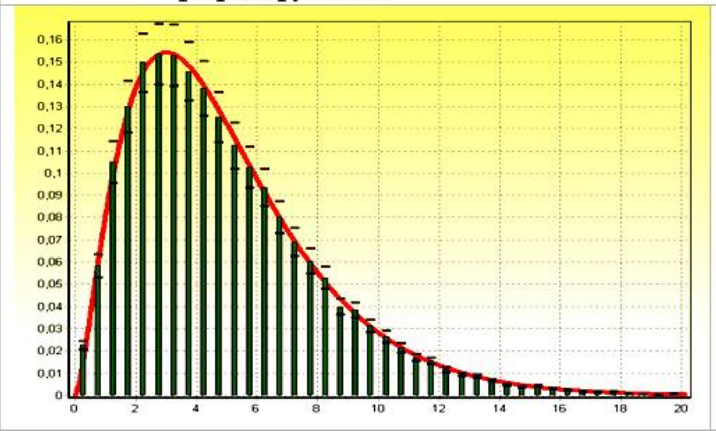
Интегральные результаты	График функции
Метод: Метод Неймана Статистика 10000 Дисперсия 0,2724814 Мода 2,5 Отклонение 0,6306778	

Рис. 10. Отчет по моделированию распределения χ^2

После окончания работы с сервером MS Word необходимо от него отключиться:

```
WordApplication1.Disconnect;
```

4.2. TListView для визуализации результатов моделирования

В разделе приведен пример использования компонента TListView для визуализации и анализа результатов моделирования. Работа модуля сводится к тому, что по мере выполнения расчетов с разными входными параметрами сохраняется история вычислений. При этом сохраняются интегральные результаты и графическое представление полученной выборки. Пользователь имеет возможность сравнивать результаты последнего эксперимента и выбранного в истории, а также изменять последовательность экспериментов в истории или удалять их (рис. 11).

Для хранения истории результатов в табличном виде используется компонент TListView, который находится на вкладке Win32 палитры компонентов Delphi. После размещения компонента на форме необходимо установить его свойства:

- ViewStyle → vsReport;
- ReadOnly, RowSelect → true;
- GridLines (линии таблицы) → true;

- в редакторе свойства Columns указать подписи для всех столбцов.

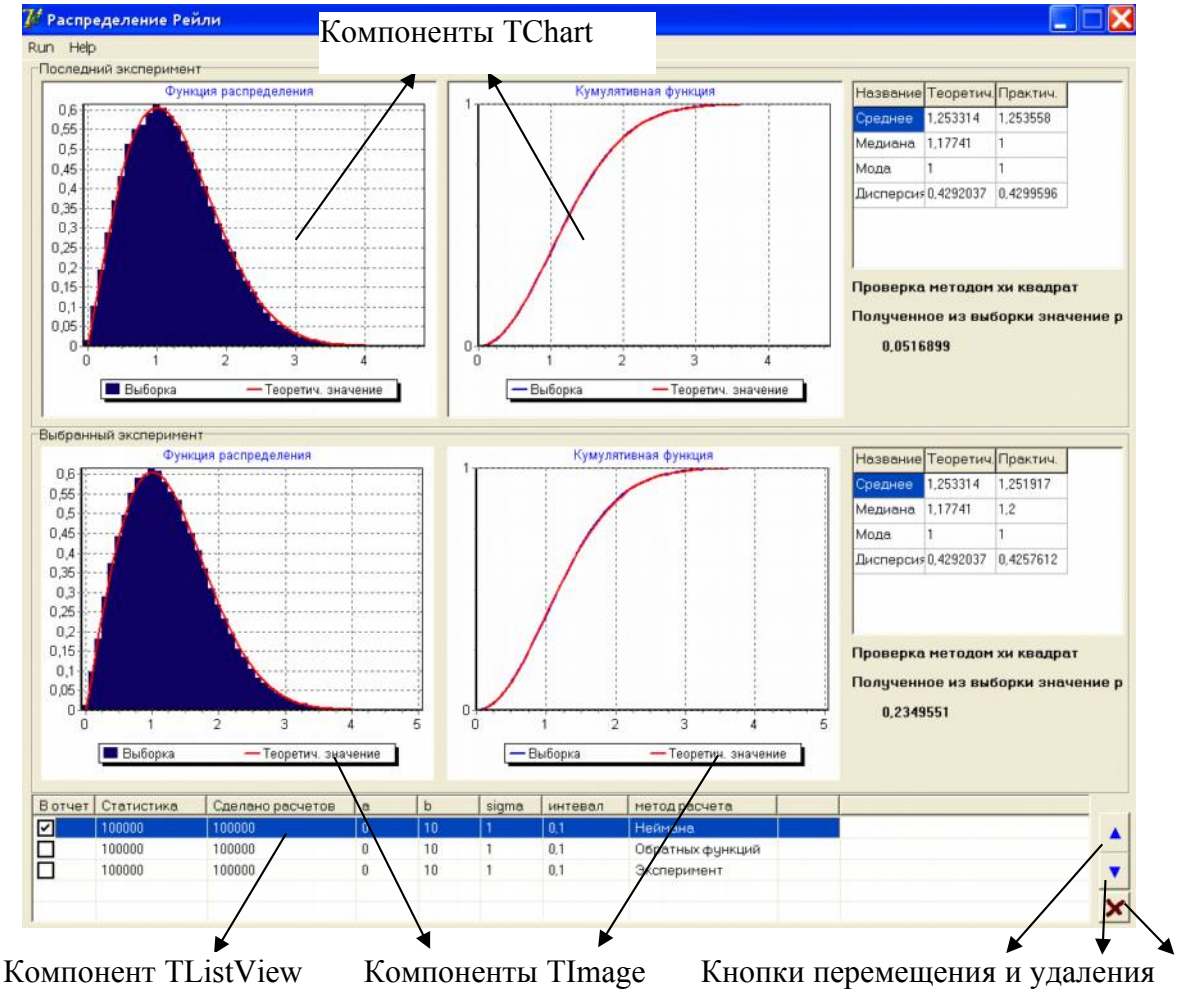


Рис. 11. Форма для визуализации результатов

Создадим класс TExper для хранения входных параметров, интегральных результатов вычислений и битового образа графика.

```
TExper = class(TObject)
Public
   //различные интегральные результаты
   disp : TBitmap; //графики
   selected : boolean; //нужно ли записывать в отчет
   //конструктор класса
   constructor Create(результаты);
end;
```

После расчета программа визуализирует полученную выборку на графике и сохраняет результаты в глобальной переменной TempExper – экземпляре класса TExper**.** Для сохранения в объектной переменной битового образа графика используются следующие фрагменты кода:

```
Form1.Chart1.CopyToClipboardBitmap; //Копируем в буфер обмена
. . .
//из буфера обмена в битовый массив
var i:integer;
bmp : TBitmap;
begin
   if Clipboard.HasFormat(CF_BITMAP) then
   begin
      bmp := TBitmap.Create;
      bmp.LoadFromClipBoardFormat(cf_BitMap,
          ClipBoard.GetAsHandle(cf_Bitmap),0);
      TempExper.disp := bmp;
   end;
end;
```

Для хранения информации о расчетах в программе используется «разреженный массив» ExperList, реализованный как массив указателей. После сохранения результатов в переменной TempExper, в массив ExperList добавляется указатель `ExperList.Add(TempExper)`. Теперь заполним записями элемент TListView.

```
procedure TForm1.ListRefresh;
var
  i : Integer;
  ListItem: TListItem;
  Temp : TExper; // переменная класса записи
begin
  with ListView1 do
  begin
    Clear;//очищаем все записи
    Checkboxes := true;//разрешаем CheckBox в 1 столбце
```

```
    ViewStyle := vsReport;
    for i:=0 to ExperList.Count - 1 do
    begin
      Temp := ExperList[i];//Массив записей
      ListItem := Items.Add;//создаем новую запись в ListView
      ListItem.SubItems.Add(IntToStr(Temp.statistics));
      //Тут добавляем столько данных, сколько столбцов
      ListItem.Checked := Temp.selected;
      ListItem.ImageIndex := i;//номер записи будет
       полезен   //при работе с функциями перемещения и
       удаления.
    end;
  end;
end;
```

Первый столбец содержит поле CheckBox, это дает возможность пользователю отмечать интересующие записи для формирования отчета. На рис. 11 видно, что на двух верхних графиках представлены результаты последнего расчета, а на двух нижних графиках – результаты выделенного из списка расчета. Пользователь щелчком мыши на интересующей строке списка выделяет результаты для визуализации на нижних графиках. Для реакции на это действие пользователя необходимо создать обработчик события OnSelectItem компонента TListView.

```
procedure TForm1.ListView1SelectItem(Sender: TObject; Item:
TListItem;  Selected: Boolean);
var   Temp : TExper;
i : integer;
begin
 //если выделили существующую запись – делаем все
 //строки списка активными, иначе нет
 GroupBox1.Visible := Selected;//вывод интегр.результатов
 SpeedButton1.Enabled := Selected;//кнопка переместить вверх
 SpeedButton2.Enabled := Selected;//кнопка переместить вниз
 SpeedButton3.Enabled := Selected;//кнопка удалить
```

```
 Temp := ExperList[Item.Index];
 RedrawIntegralRes(1, Temp); //рисуем скриншот графика
 Image1.Picture.Assign(Temp.disp);
end;
```

Если в список результатов будет занесено большое количество записей, может быть удобным перемещение записей по списку и возможность удаления записи. Обработчик события нажатие кнопки «Переместить вверх»:

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  if(ListView1.Selected.ImageIndex > 0) then
  begin
     //iSelected глобальная перем. –номер выделенной записи
    iSelected := ListView1.Selected.ImageIndex;
    SaveList;//если в поле CheckBox выставлен флажок –
    надо   //сохранить его, т.к. при перерисовке все будет
    утеряно
    ExperList.Move(ListView1.Selected.ImageIndex,
    ListView1.Selected.ImageIndex - 1);
    ListRefresh; //обновляем список
    ListView1.ItemIndex := iSelected - 1;
     //выделенная запись переместилась вверх
  end;
end;

//процедура сохранения списка
procedure TForm1.SaveList;
var i:integer;
Temp : TExper;
begin
  with ListView1 do
  begin
    for i:=0 to Items.Count-1 do
    begin
      Temp := ExperList[i];
```

```
            Temp.selected := ListView1.Items[i].Checked;
          end;
        end;
      end;
```

Аналогичным образом работают процедуры перемещения записи вниз и удаления.

4.3. TChart для визуализации результатов моделирования

Для визуализации результатов моделирования можно использовать возможности, которые предоставляет компонент TChart. Вид интерфейса вывода для приведенного ниже примера показан на рис. 12. Верхнюю часть формы занимает график (TChart), нижнюю – сетка (TStringGrid), поля для ввода значений параметров (TEdit) и кнопки управления (TButton) для запуска и прерывания расчета.

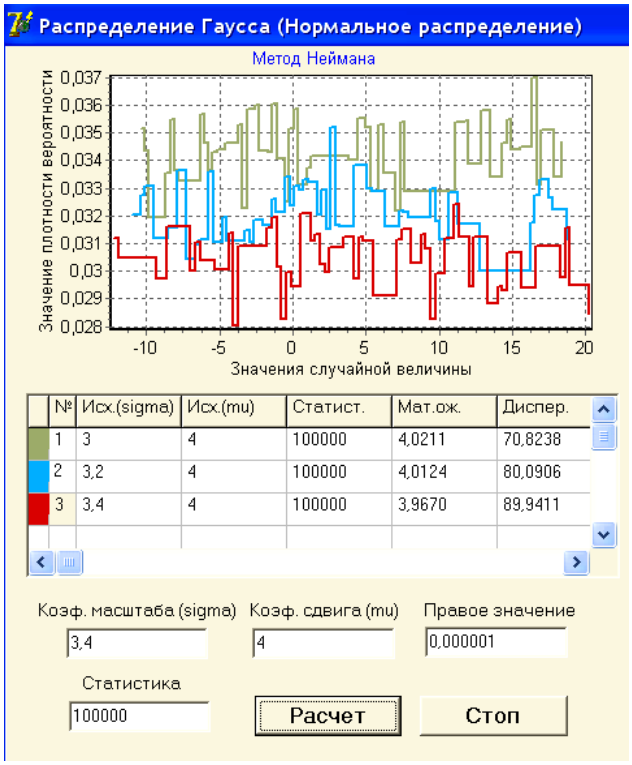


Рис. 12. Интерфейс вывода результатов

При визуальном анализе результатов хороший эффект достигается за счет того, что на одном графике размещают сразу несколько кривых. При этом в полной

мере можно использовать когнитивные возможности компонента TChart – перемещать кривые внутри поля графика, увеличивать масштаб выделенной области графика. Если отдельно отображать графические и интегральные результаты расчетов, важно установить полное соответствие между ними. Пример реализован следующим образом: после каждого расчета динамически создается новая серия и добавляется на график, в сетку добавляется строка с интегральными характеристиками результатов расчета. Цвет кривой графика выбирается случайным способом, один из столбиков добавляемой в сетку строки окрашивается в цвет серии, чтобы визуально показать соответствие графических и интегральных результатов. Для сетки используется всплывающее меню, которое позволяет удалить отмеченную строку из сетки и соответствующую ей кривую из графика (рис. 13, 14). В этом примере для хранения графических результатов используется внутренняя память TChart. Коды этого примера можно использовать как основу для реализации любого алгоритма взаимодействия пользователя с программой.

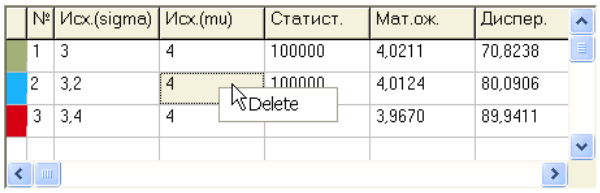


Рис. 13. Удаление строки результатов расчета

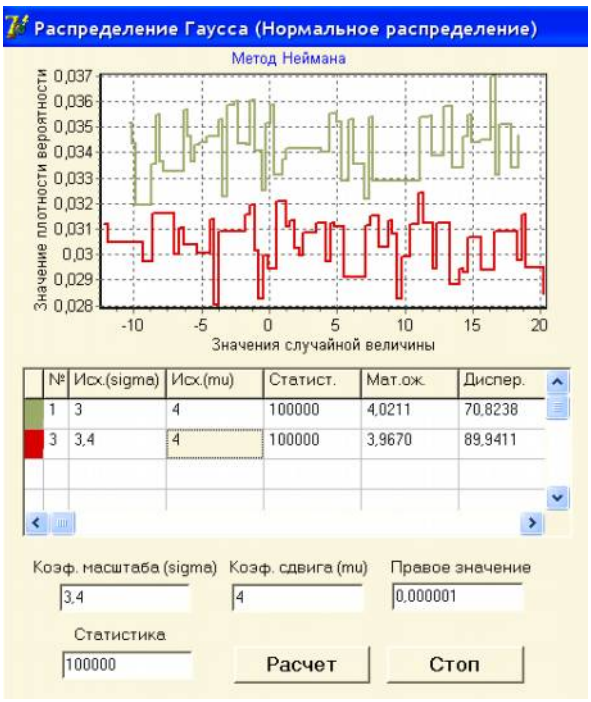


Рис. 14. Интерфейс вывода после операции удаления

Для реализации нужно объявить две глобальные переменные:

```
var
      MySr:TLineSeries;//объектная переменная: серия типа Line
      NomSeries:integer;//счетчик серий
```

Фрагмент кода для создания новой серии после очередного расчета:

```
    MySr:=TLineSeries.Create(self);    //собственник – форма
    MySr.ParentChart:=Chart2; //родительский компонент – график
    MySr.Stairs:=true; //вид кривой – гистограмма
    MySr.LinePen.Width:=2; //толщина кривой
    MySr.SeriesColor:=RGB(trunc(random*255),
       trunc(random*255),trunc(random*255)); //цвет кривой
    NomSeries:=Chart2.SeriesCount; //счетчик кривых
    for i:=0 to NeymList.Count-1 do //перебор элементов
     //разреженного массива, в нем сохраняли результаты расчета
      begin
       temp:=NeymList.Items[i];
        Chart2.Series[NomSeries-1].AddXY(temp.x, temp.y);
     end;
```

```
StringGrid2.Cells[0,NomSeries]:='    '; //цвет
StringGrid2.Cells[1,NomSeries]:=IntToStr(NomSeries); //номер
StringGrid2.Cells[2,NomSeries]:=FloatToStr(sigma); //Исх.дан.
StringGrid2.Cells[3,NomSeries]:=FloatToStr(mu); //Исх.данные
StringGrid2.Cells[4,NomSeries]:=IntTostr(Statistic); //Стат.
StringGrid2.Cells[5,NomSeries]:=FloatToStrF(MatO,ffFixed,6,4);
StringGrid2.Cells[6,NomSeries]:=FloatToStrF(Disp,ffFixed,6,4);
//В 4 столбик выводим мат.ожидание,в 5 – дисперсию
StringGrid2.Col:=1; //устанавливаем фокус сетки на 2 столбик
StringGrid2.RowCount:=StringGrid2.RowCount+1;
```

Для того, чтобы в сетке показать цветом соответствие с графическими результатами, отображенными в виде кривых на графике, необходим обработчик события OnDrawCell «Отрисовка ячеек сетки»:

```
procedure TForm1.StringGrid2DrawCell(Sender: TObject; ACol,
ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
 MCol: longint absolute ACol; // номер текущего столбца
 MRow: longint absolute ARow; // номер текущей строки
begin
 with (Sender as TStringGrid) do
  begin
   Canvas.Font := Font; // фонт формы
   Canvas.Font.Color := clBlack; // цвет текста в ячейке
   if (gdFixed in State) then // если рисуется фикс.строка
    Canvas.Brush.Color :=clBtnFace // цвет стандартный
   else begin
    if (gdFocused in State)or (gdSelected in State) then
     // если рисуем выделенную ячейку или с фокусом
      Canvas.Brush.Color:=clBtnFace// цвет стандартный
    else // обычные ячейки
     if (MRow<=Chart2.SeriesCount) then begin
```

```
      // рисуем непустые строки
      if (MCol=0) then // первый столбец – цветной
        Canvas.Brush.Color:=Chart2.Series[MRow-1].SeriesColor
        // цвет ячейки такой же как у соответствующей серии
      else
        Canvas.Brush.Color:=clWhite //остальные ячейки
     end
     else Canvas.Brush.Color:=clWhite; //ячейки пустых строк
   end;
   Canvas.FillRect(Rect); //залить цветом прямоугольник ячейки
   Canvas.TextOut(Rect.Left+4,Rect.Top,Cells[MCol, MRow]);
   //вывести в ячейку текст
  end;
end;
```

В процедуре отрисовки ячеек сетки анализируем, отрисовка какой текущей ячейки сетки производится. Если это ячейка фиксированной строки, ячейка с фокусом или выделенная – рисуем ее серым цветом. Если ячейка принадлежит первому столбцу той строки, результаты которой представлены на графике, рисуем ее цветом серии графика, все остальные ячейки – белые. После отрисовки фона ячейки выводим текст. Процедура удаления строки сетки и соответствующей кривой графика:

```
procedure TForm1.Delete1Click(Sender: TObject);
var
  i,MRow:integer;
begin
 MRow:=StringGrid2.Row;
 if MRow=0 then exit;
 if Chart2.SeriesCount=0 then exit;
 MySr:=Chart2.SeriesList.Items[MRow-1];//Присвоить указатель
   //на серию графика, номер которой выбирается по
   выделенной //строке сетки
 Chart2.RemoveSeries(MySr); //Удалить серию из графика.
```

```
//Выполнение этого дествия инициирует перерисовку графика
 if Chart2.SeriesCount=0 then
  MySr.Free; //если нет серий – освободить память
 //блок для удаления отмеченной строки из сетки
for i:=MRow to StringGrid2.RowCount-2 do
  begin
   StringGrid2.Rows[i]:=StringGrid2.Rows[i+1];
   StringGrid2.Rows[i+1].Clear;
  end;
end;
```

При закрытии формы нужно выполнить действия по освобождению памяти, если она распределена под объектную переменную типа `TlineSeries`, и очистить серии графика:

```
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
var i:integer;
begin
 if Chart2.SeriesCount>0 then MySr.Free;
 for i:=0 to Chart2.SeriesCount-1 do
  Chart2.Series[i].Clear;
end;
```

5. Разработка справочной системы

Как уже говорилось в разделе 2, программа должна содержать справочную систему, в которой присутствуют, по крайней мере, три основных раздела: *User guide* (учебник пользователя с описанием решаемой задачи и методов ее решения), *How to get result* (инструкция пользователя по работе с программой), *About* (сведения об авторах, контакты).

Для разработки справочной системы существует достаточно много разных возможностей. Мы предлагаем к рассмотрению реализацию справки в виде *.chm файла. Такой файл представляет собой набор html страниц,

скомпилированный в единый chm файл. Вначале разрабатываются и тестируются отдельные html-страницы, после чего после компиляции с помощью HTML Help Workshop (бесплатно распространяется в сети Интернет) создается единый справочный chm файл.

Для вызова файла справки может быть использована Win API функция ShellExecute, которая позволяет открывать исполнимый файл или документ – подготовленный chm файл. Для возможности использования этой функции необходимо в секцию `uses` добавить библиотеку `ShellApi`. Рассмотрим пример вызова файла справки, если он находится в поддиректории help той директории, где расположена программа:

```
var HelpDir:string;
begin
HelpDir:= GetCurrentDir+'\help\myhelp.chm';
ShellExecute(Application.MainForm.Handle, nil, PChar(HelpDir),
      nil, nil, SW_SHOW);
end;
```

6. Список цитируемых источников

1. Гулд Х. , Тобочник Я. Компьютерное моделирование в физике: В 2-х частях. Часть 2: Пер. с англ. – М. : Мир, 1990. – 400 с.
2. Соболь И. Численные методы Монте-Карло. – М.: Наука, 1973. – 312 с.
3. Сван Том. Delphi 4. Библия разработчика: Пер. с англ. – К.; М.; СПб. : Диалектика, 1998. – 672 с.
4. Флёнов М. Библия Delphi. – СПб. : БХВ-Петербург, 2007. – 880 с.
5. Флёнов М. Delphi в шутку и всерьёз: что умеют хакеры. – СПб. : Питер, 2006. – 270 с.
6. Флёнов М. Delphi 2005. Секреты программирования. – СПб. : Питер, 2006. – 266 с.
7. Пашеку Хавьер. Программирование в Borland Delphi 2006 для профессионалов : Пер. с англ. – М. : Издательский дом «Вильямс», 2006. – 944 с.

8. Хладни Иван. Внутренний мир Borland Delphi 2006 : Пер. с англ. – М. : Издательский дом «Вильямс», 2006. – 768с.
9. Delphi Russian Knowledge Base, http://www.forum.sources.ru
10. Shneiderman B. Designing the User Interface, 3-rd edn. – Reading, MA: Addison-Wesley, 1998.

Навчальне видання

Волков Мирослав Олександрович**,**
Сорочкін Олександр Сергійович**,**
Лазурик Валентина Михайлівна**,**
Лазурик Валентин Тимофійович

Розробка програм генераторів псевдовипадкових чисел

Російською мовою

Коректор С. В. Гончарук
Комп'ютерна верстка В. М. Лазурик
Розробка дизайну макета обкладинки І. М. Дончик

Підписано до друку Формат 60х84/16.
Папір офсетний. Друк ризографічний.
Ум.-друк. арк. 1,63. Обл.-вид. арк. 1,75. Наклад 100 прим.
Ціна договірна

61077, Харків, майдан Свободи, 4,
Харківський національний університет імені В. Н. Каразіна,
Видавництво ХНУ імені В. Н. Каразіна
Свідоцтво про державну реєстрацію ВОО № 948011 від 03.01.03