Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____РК_____

КАФЕДРА ___Компьютерные системы автоматизации производства РК-9_____

_____

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовому проекту на тему:

_____

_____

_____

_____

_____

_____

_____

Студент _____ _____ _ Клеванец И.С. _

(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____ _Урусов А.В.___

(Подпись, дата) (И.О.Фамилия)

Москва, 2011

Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой __РК9____
(Индекс)
_____ _____
(И.О.Фамилия)
« ___ » _____ 20 __ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине_____ _____

___ _____
(Тема курсового проекта)
_____

Студент__*Клеванец И.С. РК9-91*_____
(Фамилия, инициалы, индекс группы)

График выполнения проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

1. Техническое задание

_____
_____
_____
_____

2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на ___ листах формата А4.
2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.)_____
*_лист 1— постановка задачи;*_____
*_лист 2 – диаграмма компонентов;*_____
*_лист 3— диаграммы классов, синтаксическая диаграмма;*_____
*_лист 4— алгоритм работы генератора;*_____
_лист 5—результаты;_____
Дата выдачи задания « » _____ 2011г.

Руководитель курсового проекта _____ ____*Урусов А.В.*_____
(Подпись, дата) (И.О.Фамилия)

Студент _____ ___*Клеванец И.С.*_____
(Подпись, дата) (И.О.Фамилия)

Примечание:
1. Задание оформляется в двух экземплярах; один выдаётся студенту, второй хранится на кафедре.

Оглавление

1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1.1. Введение

В рамках имитационного моделирования постоянно приходится сталкиваться со случайными величинами. Как правило, это величины, связанные со временами (периоды прихода заказов, время обработки заказов), реже – с количествами. Большинство случайных величин подчиняется законам распределения. Основные законы распределения: равномерный, нормальный, экспоненциальный. Программный продукт РДО может работать с каждым из них.

1.2. Основания для разработки

Программный продукт РДО может генерировать последовательность псевдослучайных величин по равномерному, нормальному, экспоненциальному законам распределения, но не может работать с треугольным законом распределения. Есть необходимость в расширении возможностей РДО, а именно в возможности работы с треугольным законом.

1.3. Назначение разработки

Разработать подсистему для генерации псевдослучайных величин, распределенных по треугольному закону.

1.4. Требования к программе или программному изделию

а) Требования к функциональным характеристикам:

- обеспечение РДО возможности работать с треугольным законом распределения с условием передачи параметров;
- интегрирование в РДО в каждом функциональном блоке, из которого доступны законы распределения;
- доступность из каждого интерфейса РДО (текстовый и графический);
- кросс-платформенный код ПО.

б) Требования к надежности:

- стабильная работа РДО при использовании треугольного закона из каждого интерфейса;
- первичная проверка вводимых условий;
- автоматическое тестирование.

в) Условия эксплуатации:

- полностью соответствуют условиям эксплуатации РДО.

г) Требования к составу и параметрам технических средств:

- полностью соответствуют требованиям к РДО.

д) Требования к информационной и программной совместимости:

- полностью соответствуют требованиям к РДО.

1.5. Требования к программной документации

а) документация для пользователей (синтаксис, входные параметры, примеры использования);

б) документация для разработчиков РДО (комментарии к коду программы, обеспечение совместимости с автоматической системой генерации документации к исходному коду РДО).

1.6. Стадии и этапы разработки

а) разработка алгоритмической части кода;

б) разработка интерфейсных частей кода:

- для текстового интерфейса;
- для графического интерфейса;

в) организация связей между алгоритмической и интерфейсными частями кода;

г) разработка автоматических тестов для уже имеющихся в РДО и вновь разрабатываемого законов распределения;

д) разработка документации для пользователей;

е) разработка документации для разработчиков РДО.

1.7. Порядок контроля и приемки

а) запуск и работа имитационной модели с использованием треугольного закона распределения;

б) правильное срабатывание автоматических тестов;

в) правильный вид гистограммы, построенной на основе выборки, полученной из разработанного кода.

2. КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ

2.1. Компоненты РДО

Система имитационного моделирования РДО безусловно является сложной и статически, и динамически. На это указывает сложная иерархическая структура системы со множеством различных связей между компонентами и ее сложное поведение во времени.

Ярко выраженная иерархическая структура и модульность системы определяют направление изучения системы сверху вниз. Т.е. принцип декомпозиции применяется до тех пор, пока не будет достигнут уровень абстракции, представление на котором нужных объектов не нуждается в дальнейшей детализации для решения данной задачи.

Для отображения зависимости между компонентами системы РДО и выделения среди них модернизируемых служит соответствующая диаграмма в нотации UML.

Базовый функционал представленных на диаграмме компонентов:

rdo_kernel реализует функции ядра системы. Не изменяется при разработке системы.

RAO-studio.exe реализует графический интерфейс пользователя. Не изменяется при разработки системы.

rdo_repository реализует управление потоками данных внутри системы и отвечает за хранение и получение информации о модели. Не изменяется при разработке системы.

rdo_mbuilder реализует функционал, используемый для программного управления типами ресурсов и ресурсами модели. Не изменяется при разработке системы.

rdo_converter конвертирует модели созданные в старой версии РДО, производя резервное копирование файлов оригинальной модели. Благодаря нему обеспечивается обратная совместимость версий системы. Не изменяется при разработке системы.

rdo_simulator управляет процессом моделирования на всех его этапах. Он осуществляет координацию и управление компонентами rdo_runtime и rdo_parser. Не изменяется при разработке системы.

rdo_parser производит лексический и синтаксический разбор исходных текстов модели, написанной на языке РДО. Модернизируется при разработке системы.

rdo_runtime отвечает за непосредственное выполнение модели, управление базой данных, базой знаний, планирование и выполнение событий, и работу процессов. Модернизируется при разработке системы.

rdo_calc отвечает за формирование калков, универсальных блоков, из которых формируются арифметические, логические, логические цепочики.

Таким образом, основные изменения должны затронуть модули rdo_calc, rdo_parser и rdo_runtime.

2.2. Работа с алгоритмической частью

Алгоритмы получения псевдослучайных чисел, распределенных по каждому из уже существующих в РДО законов, расположены в блоке rdo_runtime, в файле rdo_random_distribution.inl. В этом же блоке будет и алгоритм для треугольного закона. Он будет реализован в виде нового класса.

2.3. Работа с интерфейсом РДО

2.3.1. Текстовый интерфейс

После решения поставленной задачи в распоряжении пользователя РДО должен появиться инструмент запуска генератора последовательности псевдослучайных чисел, работающего по треугольному закону. Этим инструментом станет новые инструкции, доступные в теле функций РДО, а так же в теле процессов и паттернов РДО.

Таким образом, лексический и синтаксический анализаторы компонента rdo_parser должны начать правильно обрабатывать новые конструкции языка.

В теле функций РДО предусмотрены три типа объектов исходных данных:

- константы;
- функции;
- последовательности ПСВ.

Описание объекта типа последовательности ПСВ[5]:

Значение вызываемой последовательности определяется генератором псевдослучайных чисел с соответствующим распределением либо выбирается из числа значений, заданных при описании последовательности.

Синтаксис описания последовательности имеет вид:

```
$Sequence <имя_последовательности> : <тип_значения_последовательности>
$Type = <тип_последовательности> <значение_базы_генератора>
$End
```

имя_последовательности

Имя последовательности представляет собой простое имя. Имена должны быть различными для всех последовательностей и не должны совпадать с ранее определенными именами.

тип_значения_последовательности

Тип значения последовательности - это один из возможных в языке типов данных. При описании типов значений последовательности возможны ссылки на типы параметров ресурсов и типы символьных констант.

тип_последовательности

нормальный, равномерный, треугольный законы распределения

В последний блок и будут вноситься дополнения.

Синтаксис вызова последовательности, описанной в теле функций:

имя_последовательности (параметр_1 [, параметр_2 [, параметр_3]])

Параметры, их количественный и качественный состав, для каждого закона свои.

2.3.2. Графический интерфейс

В блоках Create и Process выбираются параметры работы этих блоков.

Для блока Create выбирается количество транзактов, которые он сгенерирует, параметры интервала между появлениями транзактов, группа, к которой будут принадлежать транзакты. Для блока Process выбирается тип обработки транзакта (его работа с ресурсами), сами ресурсы, время обработки транзакта.

Случайные величины в блоке Create используются для задания интервала прихода заявок в систему. В блоке Process для задания времени обработки заявки. Эти времена так же, как и в текстовом интерфейсе, описываются типом закона распределения и его параметрами.

Необходимо добавить возможность выбор треугольного закона и задания его параметров.

3. ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

3.1. Проектирование алгоритмической части

Алгоритмы работы имеющихся генераторов случайных чисел реализованы с помощью метода обратного преобразования. Для того чтобы сохранить единообразие в системе, генератор последовательности псевдослучайных чисел, работающий по треугольному закону, так же следует реализовать через обратное преобразование. [3]

Суть этого метода сводится к тому, что для получения распределения, требуемого вида (F(x)), нужно применить обратную этому виду функцию ($F^{-1}(x)$) к базовому равномерному распределению на интервале [0,1); где F(x) – функция распределения.

3.2. Проектирование интерфейсной части

3.2.1. Новые синтаксические конструкции

Тело функций РДО

На концептуальном этапе проектирования был сделан вывод о необходимости реализации новой инструкции языка РДО, функция которой будет заключаться в вызове генератора случайных чисел, работающего по треугольному закону распределения.

В синтаксис этой инструкции должно быть включено имя генератора, который должен быть запущен (имя последовательности):

triangular

В остальном синтаксическая конструкция формирования последовательности остается неизменной.

Тело процессов и паттернов РДО

Вызов этой функции должен выполняться с помощью синтаксиса:

имя_последовательности (левая граница диапазона распределения,
точка под вершиной треугольника,
правая граница распределения)

3.2.2. Новые элементы графического интерфейса

В блоках Create и Process в выдающем меню необходимо добавить строку, соответствующую треугольному закону, а так же в диалоговом окне дать возможность пользователю ввести параметры закона. Данные будут миновать rdo_parser, т.к. нет синтаксических конструкций, которые ему надо разбирать, поэтому данные будут отправляться в rdo_calc. Далее в rdo_runtime.

3.3. Проектирование связей между алгоритмической и интерфейсной части

Связь между алгоритмической и интерфейсной частью обеспечивается с помощью калков. Это универсальные блоки передачи данных в рамках РДО. С их помощью инкапсулируются знаки арифметических операций, имена переменных, функций, ресурсов, их типы и т.д.

Графический интерфейс передаст тип закона и его параметры в калки, и блок rdo_calc передаст их в runtime. Текстовый интерфейс сначала разберет синтаксис, после этого передаст в калки и потом в runtime.

3.4. Документация

Документация для пользователя реализована в виде справки, вызываемой из диалогового окна РДО в меню «Помощь», строка «Содержание», или нажатием клавиши F1.

В документации для пользователя представлены инструкции к использованию текстового интерфейса для получения последовательности псевдослучайных величин, распределенных по треугольному закону. Описаны параметры вызова, а также приведены примеры использования треугольного закона.

4. РАБОЧЕЕ ПРОЕКТИРОВАНИЕ

4.1. Проектирование алгоритмической части

Функция плотности распределения вероятности для треугольного закона имеет вид:

$$f(x) = \begin{cases} \frac{-2*(x-b)}{(c-a)*(a-b)} + \frac{2}{c-a}, x \in [a;\ b); \\ \frac{-2*(x-b)}{(c-a)*(c-b)} + \frac{2}{c-a}, x \in [b;\ c); \\ 0\ , x \in (-\infty; a) \cup (c;\ +\infty). \end{cases} \quad (1)$$

График функции соотвественно имеет вид:

Рис.1. График. Функция плотности распределения случайной величины по треугольному закону

где a = 2 – левая граница диапазона распределения,
b = 7 – точка под вершиной треугольника,
c = 10 – правая граница диапазона распределения.

Для алгоритма необходима функция распределения вероятности:

$$F(x) = \int f(x)dx \quad (2)$$

f(x) – линейная функция, потому F(x) – квадратичная. Обратная функция – квадратный корень.

Из-за того, что f(x) и F(x) не непрерывны, существует необходимость сначала определить, для которой из частей необходимо искать обратную функцию. Это делается с помощью базового равномерно распределения. Если полученное число меньше или равно (b-a)/(c-a), то следует получать обратную функцию для левой части графика. Если больше, то для правой.

Результатом применения квадратного корня к величине, лежащей в интервале от 0 до 1, будет величина, лежащая в том же интервале, поэтому квадратный корень помножается на ширину левой (b-a) или правой (c-b) части соответственно.

4.2. Смысл использования базы генератора

Парадокс заключается в том, что в рамках данного проекта надо случайную величину заставить подчиняться определенному закону, когда на самом деле по-настоящему случайной величины нет. Вся текущая работа выполняется на цифровой вычислительной технике. Эта техника подчиняется строгим законам. Можно сказать, она строго детерминирована, иначе она не смогла бы полноценно функционировать. Поэтому случайности тут взяться неоткуда. Она бы постоянно давала сбои, если бы смогла запуститься.

На самом деле в мире техники получить случайное число – большая проблема. Очень часто работа со случайными числами сводятся к двум этапам: найти основу для вычислений (случайность) и преобразовать ее для определенных нужд. Последняя часть и выполняется в данном курсовом проекте.

Основа – чаще всего число (база генератора), которое получают из разных источников. Этот этап реализуют по-разному. Часто в качестве базы используют время, т.к. это число постоянно меняется. Но это не идеальное решение: оно меняется медленно. Иногда, когда нужно получить одно-единственное число, пользователя просят подвигать мышью, чтобы позже на основе тех перемещений курсора получить базу: опыт сложно повторим, поэтому можно считать величину случайной. Одна из компаний, предоставляющих интернет-услуги, которой нужно было ежедневно генерировать множество случайных чисел, использовала погодные данные в качестве баз генератора.

В данном курсовом проекте источником случайности является пользователь, который вводит базу генератора, которую должен использовать генератор псевдослучайных чисел. Пользователь вводит базу для первого числа. Далее РДО получает новые базы на основе введенной пользователем, естественно, по четким правилам. Такой подход позволяет обеспечить повторяемость опытов, а случайность каждого нового эксперимента определяется фантазией пользователя.

Ниже приведен алгоритм определения базы генератора для нового числа из последовательности.

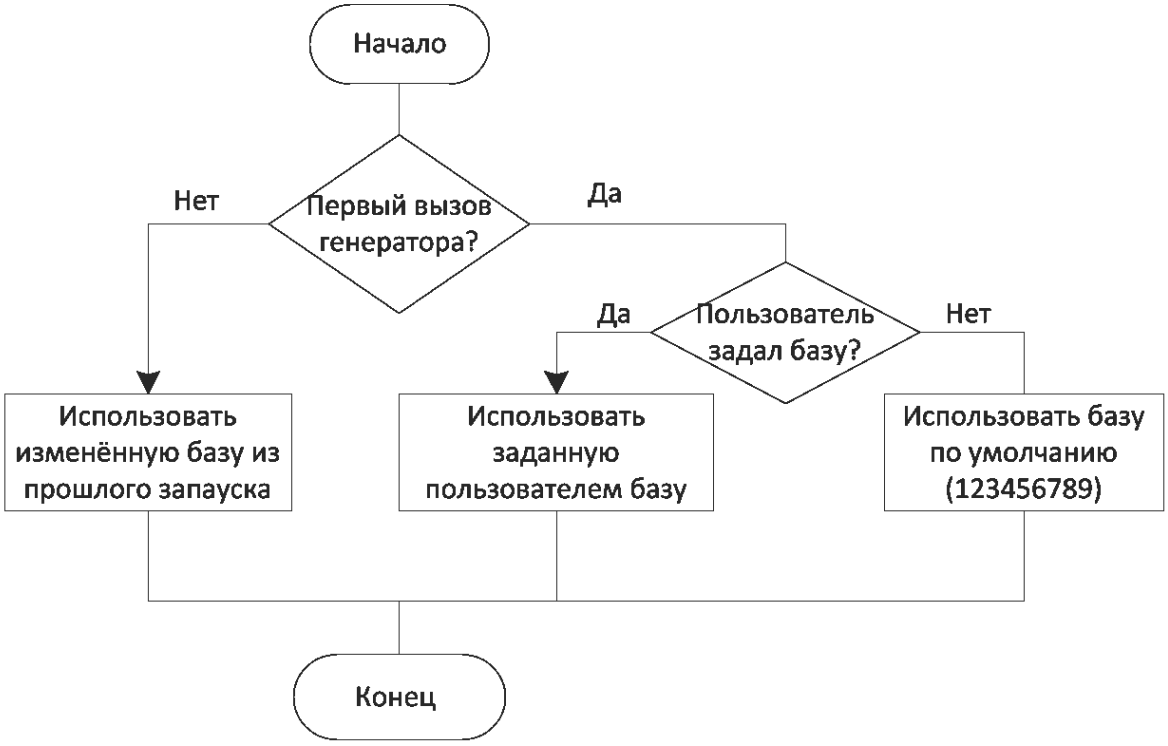

Рис.2. Блок-схема. Алгоритм определения базы для генератора псевдослучайных чисел.

4.3. Описание базового равномерного распределения

Суть заключается в том, что база все время растет и переполняет память, которая ограничена 4 байтами. Лишнее отбрасывается. 69069 - то число, благодаря которому получается все гладко и красиво, объяснения этому явлению найти не удалось. +1 - Благодаря этому можно не беспокоится из-за того, что база обратится в ноль, и базовое генератор будет возвращать только лишь 0. Функция возвращает максимально возможное для 4 байт число плюс 1. Т.е. результатом работы генератора является приведенная база генератора. В итоге возвращается псевдо-случайное число, значение в диапазоне [0, 1).

Ниже приведен алгоритм работы генератора.

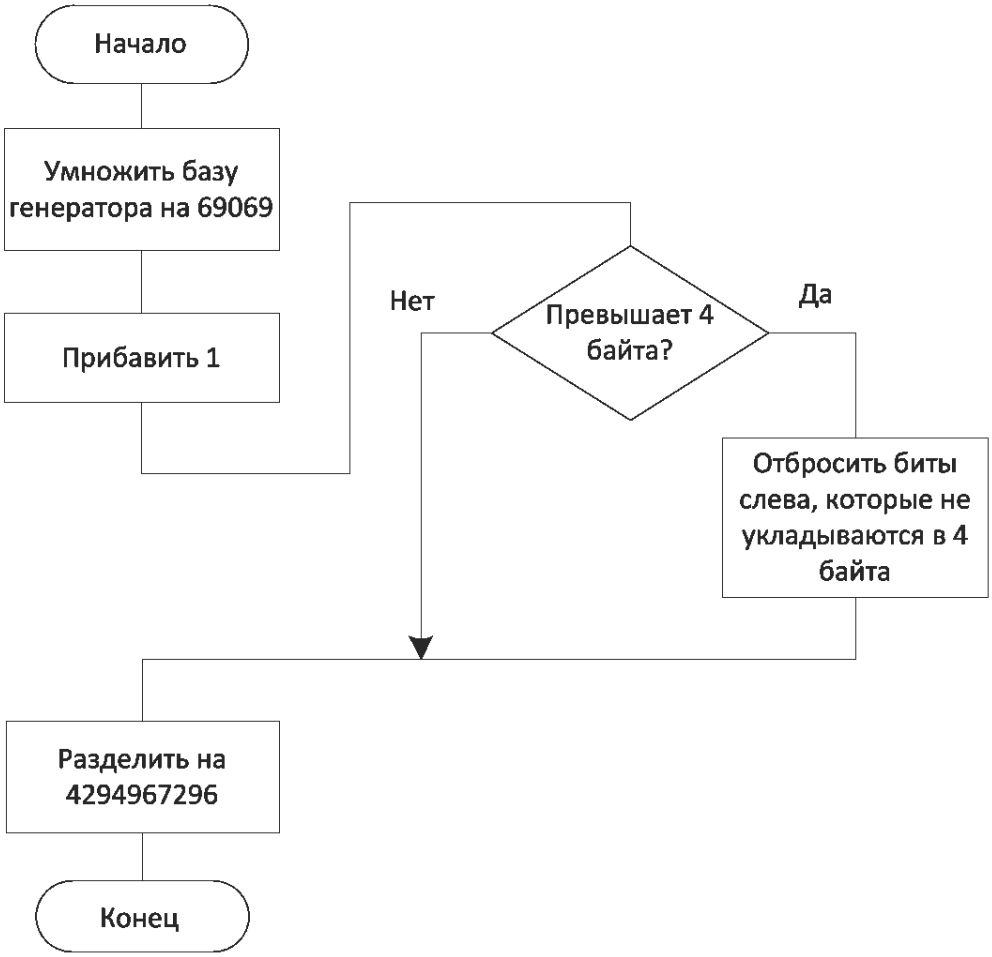

Рис.3. Блок-схема. Алгоритм работы генератора чисел из базового равномерного распределения

4.4. Документация

Документация для разработчиков РДО генерируется с помощью автоматизированного средства получения документации к программному коду Doxygen. Эта система формирует описание классов, функций, их параметров с помощью определенным образом форматированных комментариев к коду. Представляет описание в виде структурированной системы, содержащей подробное описание кода программы, диаграммы зависимостей классов, а так же сам код программы.

В документации для разработчиков РДО представлено описание класса, генерирующего последовательность псевдослучайных чисел, распределенных по треугольному закону, метод этого класса, логика работы этого метода, параметры этого метода.

5. АВТОМАТИЧЕСКИЕ ТЕСТЫ

5.1. Введение

Они необходимы для того, чтобы удостовериться в том, что любые изменения в коде не затронули работу алгоритма, генерирующего последовательность псевдослучайных чисел по треугольному закону, а так же по нормальному, равномерному, экспоненциальному.

Так же они необходимы для того, чтобы качественно оценить работу этого алгоритма: даже если на работу генератора случайных чисел что-то повлияло, важно понять, насколько адекватна его работа.

Помимо специализированных тестов в блоке rdo_rutime встроена первичная проверка параметров, с которыми вызывается закон распределения. Ее суть заключается в том, что rdo_runtime теперь проверяет, удовлетворяют ли параметры неравенству:

$$a \leq b \leq c \quad (3)$$

Очевидно, что для закона распределения вершина треугольника не может оказаться слева от левой границы или справа от правой. Так же правая граница должна быть правее левой. Т.е. проверка не даст пользователю неправильно воспользоваться синтаксисом вызова генератора псевдослучайных чисел.

5.2. Тест на неизменность работы кода

Такой тест можно реализовать, сравнив результаты его работы сейчас и результаты, полученные ранее (эталонные). Эталонные данные – те, что алгоритм генерирует на данный момент. В этих данных есть уверенность, т.к. гистограммы, построенные на выборках из этих данных, имеют правильный вид. Выборки в 100 000 значений для каждого генератора будет достаточно. Эталонные выборки хранятся в файлах текстового типа. Точность записываемых данных составляет 17 знаков после запятой. База генератора, используемая для обоих тестов: 123456789.

5.3. Качественный тест работы генератора

Для проверки гипотезы о законе распределения принят критерий согласия Пирсона (критерий кси-квадрат). Подробное описание этого критерия в исследовательской части.

Объем выборки для критерия Пирсона составляет 2000 значений. Ему соответствует количество интервалов равное 39. При степени доверия критерию в 95%:

$F_{\text{табл}} = 50{,}9985$

6. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

6.1. Введение

Исследования проводились относительно критерия согласия Пирсона. Он используется для качественной оценки работы генераторов псевдослучайных величин, работающих по нормальному, экспоненциальному, равномерному и треугольному законам. Это один из старых и относительно слабых методов, но он позволяет удостовериться в том, что получен именно тот закон и с теми параметрами, которые ожидались.

6.2. Алгоритм работы

Нужно получить выборку проверяемого генератора псевдослучайной величины. Весь диапазон, на котором определена случайная величина, разбивается на k интервалов. Далее для каждого интервала подсчитывается количество чисел f_i из имеющейся выборки, которые принадлежат этому интервалу. [1]

Формула для вычисления числа-критерия:

$$F = \frac{(f_i - n*P_i)^2}{n*P_i} \quad (4)$$

где f_i – количество чисел, принадлежащих i-ому интервалу,

n – объем выборки,

P_i – теоретическая вероятность попадания случайной величины на i-ый интервал.

Если выполняется неравенство:

$$F < F_{\text{табл}} \quad (5)$$

где $F_{\text{табл}}$ – табличное значение, соответствующее определенной степени доверия критерию и числу степеней свободы, которое вычисляется как (k – 3).

6.3. Выбор интервалов разбиения

6.3.1. Выбор числа интервалов

Существует множество методик выбора количества интервалов. Они дают довольно сильно отличающиеся количества интервалов. Объединяет их лишь то, что количество интервалов зависит от объема выборки N.

Например, формула Старджесса:

$$k = log_2 N + 1 \quad (6)$$

Формула Брукса и Каррузера:

$$k = 5lgN \quad (7)$$

В исследовании [2]:

$$k = \frac{4}{\text{к}} lg \frac{N}{10} \quad (8)$$

где

$$к = \frac{1}{\sqrt{\mu_4 * \sigma^4}} (9)$$

где μ_4 – четвертый центральный момент случайной величины, σ – стандартное (среднеквадратическое) отклонение.

Рекомендации ВНИИМим. Д. И. Менделеева приведены в таблице 1.

Таблица 1. Сопоставление количества интервалов и объема выборки

N	k
40 – 100	7 - 9
100 - 500	8 - 12
500 – 1000	10 - 16
1000 – 10000	12 – 22

Помимо всего прочего важно, чтобы выполнялось условие $n^* P_i \geq 5$.

В рамках данного курсового проекта я воспользовался табличными данными из [4]

6.3.2. Выбор граничных точек интервалов группирования

а) Интервалы можно выбрать равной длины. В этом случае необходимо рассчитать значения вероятностей P_i

б) Интервалы можно выбрать равновероятными, т.е. площадь под графиком функции плотности вероятности на таких интервалах будет одинаковая. $P_i = 1/k$. В этом случае выбор граничных точек оказывается весьма трудоемким.

в) Можно разбивать выборку на интервалы в соответствии с асимптотически оптимальным группированием для данного закона распределения. Такое разбиение наиболее предпочтительно, так как повышает способность критерия различать близкие гипотезы. Но такое решение не очень хорошо масштабируемое.

В данном проекте использовались интервалы равной длины, т.к. это решение масштабируемое, а так же не требует громоздких и медленных вычислений.

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта были получены следующие результаты:

Проведено предпроектное исследование системы генерации псевдослучайных величин РДО и сформулированы предпосылки создания в системе инструмента для генерации последовательности псевдослучайных величин, распределенных по треугольному закону.

На этапе концептуального проектирования системы укрупненно показано внутреннее устройство РДО и выделены те компоненты, которые потребуют внесения изменений в ходе этой работы. Так же были обзорно рассмотрены эти компоненты.

На этапе технического проектирования разработан новый синтаксис вызова последовательностей, разработана документация для пользователя.

На этапе рабочего проектирования разработана структура автоматических тестов, а также первичная проверка вводимых условий для треугольного закона. Написан программный код для реализации спроектированных ранее алгоритмов работы и архитектуры компонентов rdo_parser и rdo_runtime системы РДО. Программный код является кросс-платформенным. Разработана документация для разработчиков РДО. Проведены отладка и тестирование новой системы, в ходе которых исправлялись найденные ошибки. Работоспособность системы была проверена на разработанной ранее тестовой модели-прототипе, использующей треугольный закон распределения.

Таким образом, поставленная цель курсового проекта достигнута в полном объеме.

СПИСОК ЛИТЕРАТУРЫ

1) Р 50.1.033-2001. Прикладная статистика. Правила проверки согласия опытного распределения с теоретическим. Часть I. Критерии типа хи-квадрат

2) Алексеева И.У. Теоретическое и экспериментальное исследование законов распределения погрешностей, их классификация и методы оценки их параметров: Автореф. дис. на соиск. учен, степени кан. техн. наук. - Л., 1975. - 20 с.

3) Кельтон В., Лоу А. Имитационное моделирование. Классика CS. 3-е изд. – Спб.: Питер: Киев: Издательская группа BHV, 2004. – 847 с.: ил.

4) Емельянов В.В., Ясинский С.И. Генерация случайных чисел с заданным законом распределения при имитационном моделировании ГПС: Методические указания. – М.: Изд-во МГТУ, 1993. – 16 с., ил.

5) Справка РДО. [В Интернете] http://rdo.rk9.bmstu.ru/forum/download/file.php?id=4

6) Справка Doxygen. [В Интернете] http://www.stack.nl/~dimitri/doxygen/commands.html

7) Емельянов В.В., Ясинский С.И. Введение в интеллектуальное моделирование сложных дискретных систем и процессов. Язык РДО. – М.: «АНВИК», 1998. – 427 с., ил. 136.

СПИСОК ПРИМЕНЯЕМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. Microsoft Office 2010
2. КОМПАС – 3D V10
3. Mathcad 14
4. Microsoft Paint 6.1
5. Visual Paradigm 8.0
6. Microsoft Visual Studio 2010

Приложение 1. Программный код интерфейсной части (текстовой).

Часть файла rdofun.y

```
fun_seq_triangular
   : fun_seq_header RDO_triangular RDO_End
   {
         RDOFUNSequence::LPRDOFUNSequenceHeader pHeader = PARSER-
>stack().pop<RDOFUNSequence::RDOFUNSequenceHeader>($1);
         ASSERT(pHeader);
         LPRDOFUNSequence pSequence =
rdo::Factory<RDOFUNSequenceTriangular>::create(pHeader);
         ASSERT(pSequence);
         pSequence->createCalcs();
         $$ = PARSER->stack().push(pSequence);
   }
   | fun_seq_header RDO_triangular RDO_INT_CONST RDO_End
   {
         RDOFUNSequence::LPRDOFUNSequenceHeader pHeader = PARSER->
stack().pop<RDOFUNSequence::RDOFUNSequenceHeader>($1);
         ASSERT(pHeader);
         LPRDOValue pValue = PARSER->stack().pop<RDOValue>($3);
         ASSERT(pValue);
         LPRDOFUNSequence pSequence =
rdo::Factory<RDOFUNSequenceTriangular>::create(pHeader, pValue->value().getInt());
         ASSERT(pSequence);
         pSequence->createCalcs();
         $$ = PARSER->stack().push(pSequence);
   }
   | fun_seq_header RDO_triangular RDO_INT_CONST error
   {
         PARSER->error().error(@4, _T("После базы ожидается ключевое слово $End"));
   }
   | fun_seq_header RDO_triangular error
   {
         PARSER->error().error(@3, _T("После типа последовательности ожидается база
генератора или ключевое слово $End"));
   }
   ;
```

Приложение 2. Программный код формирования калков для передачи данных в генератор псевдослучайных чисел из текстового интерфейса

Часть файла rdofun.cpp

```
// ---------------------------------------------------------------------------
// ---------- RDOFUNSequenceTriangular
// ---------------------------------------------------------------------------
RDOFUNSequenceTriangular::RDOFUNSequenceTriangular(CREF(LPRDOFUNSequenceHeader)
pHeader, int seed)
	: RDOFUNSequence(pHeader, seed)
{
	if (m_pHeader->getTypeInfo()->type()->typeID() != rdoRuntime::RDOType::t_int &&
m_pHeader->getTypeInfo()->type()->typeID() != rdoRuntime::RDOType::t_real)
	{
		RDOParser::s_parser()->error().error(src_info(),
rdo::format(_T("Последовательность '%s' может возвращять значения только целого или
вещественного типа"), src_text().c_str()));
	}
}

void RDOFUNSequenceTriangular::createCalcs()
{
	PTR(rdoRuntime::RandGeneratorTriangular) pGenerator = new
rdoRuntime::RandGeneratorTriangular();
	m_pInitCalc = rdo::Factory<rdoRuntime::RDOCalcSeqInit>::create(m_seed,
pGenerator);
	RDOParser::s_parser()->runtime()->addInitCalc(m_pInitCalc);
	m_pNextCalc =
rdo::Factory<rdoRuntime::RDOCalcSeqNextTriangular>::create(pGenerator);
	initResult();
}

LPRDOFUNArithm RDOFUNSequenceTriangular::createCallCalc(REF(LPRDOFUNParams)
pParamList, CREF(RDOParserSrcInfo) seq_src_info) const
{
	if (pParamList->getParamList()->getContainer().size() != 3)
	{
		RDOParser::s_parser()->error().error(seq_src_info, rdo::format(_T("Для
треугольного закона распределения '%s' нужно указать три параметра: левую границу,
точку под высотой треугольника, правую границу"), name().c_str()));
	}

	rdoRuntime::LPRDOCalcFunctionCall pFuctionCall =
rdo::Factory<rdoRuntime::RDOCalcFunctionCall>::create(m_pNextCalc);
	ASSERT(pFuctionCall);

	LPTypeInfo            pType =
rdo::Factory<TypeInfo>::create(rdo::Factory<RDOType__real>::create(),
RDOParserSrcInfo());
	rdoRuntime::LPRDOCalc pArg1 = pParamList->getCalc(0, pType);
	rdoRuntime::LPRDOCalc pArg2 = pParamList->getCalc(1, pType);
	rdoRuntime::LPRDOCalc pArg3 = pParamList->getCalc(2, pType);

	pFuctionCall->addParameter(pArg1);
	pFuctionCall->addParameter(pArg2);
	pFuctionCall->addParameter(pArg3);

	LPExpression pExpression = rdo::Factory<Expression>::create(
		m_pHeader->getTypeInfo(),
		pFuctionCall,
		pParamList->src_info()
	);
```

```
	ASSERT(pExpression);
	LPRDOFUNArithm pArithm = rdo::Factory<RDOFUNArithm>::create(pExpression);
	ASSERT(pArithm);
	pArithm->setSrcInfo(seq_src_info);
	if (pArithm->typeID() == rdoRuntime::RDOType::t_enum)
	{
		RDOParser::s_parser()->error().error(src_info(), _T("Внутренняя ошибка
парсера"));
	}
	return pArithm;
}
```

Приложение 3. Программный код интерфейсной части (графической). Блок Create.

Часть файла rdoprocess_shape_create_MJ.cpp

```
void RPShapeCreateMJ::generate()
{
	RPShapeDataBlock::zakonRaspr zakon;
	switch(gtype)
	{
		case 0: // константа
			zakon = RPShapeDataBlock::Const;
			break;
		case 1: // нормальный
			zakon = RPShapeDataBlock::Normal;
			break;
		case 2: // равномерный закон
			zakon = RPShapeDataBlock::Uniform;
			break;
		case 3: // треугольный
			zakon = RPShapeDataBlock::Triangular;
			break;
		case 4: // экспоненциальный
			zakon = RPShapeDataBlock::Exp;
			break;
	}
```

Часть файла rdoprocess_shape_create_dlg1_MJ.cpp

```
void RPShapeCreateDlg1_MJ::OnCbnSelchange1()
{

int cur = m_create_dlg1_combo1_MJ.GetCurSel();
UpdateData(TRUE);
	switch(cur) // определяем активные окна исходя из закона
	{

…

case 3: // труегольный
	m_create_dlgl_exp_text_MJ.SetWindowText(_T("Левая граница"));
	m_create_dlgl_disp_text_MJ.ShowWindow(SW_SHOW);
	m_create_dlgl_disp_text_MJ.SetWindowText(_T("Высота"));
	m_create_dlg1_disp_control_MJ.ShowWindow(SW_SHOW);
	m_create_dlgl_max_text_MJ.ShowWindow(SW_SHOW);
	m_create_dlgl_max_text_MJ.SetWindowText(_T("Правая граница"));
	m_create_dlg1_max_control_MJ.ShowWindow(SW_SHOW);
	break;

…

	}
	UpdateData(FALSE);
}
```

Приложение 4. Программный код интерфейсной части (графической). Блок Process.

Часть файла rdoprocess_shape_process_MJ.cpp

```
void RPShapeProcessMJ::generate()
{
	RPShapeDataBlock::zakonRaspr zakon;
	switch(gtype)
	{
		case 0: // константа
			zakon = RPShapeDataBlock::Const;
			break;
		case 1: // нормальный
			zakon = RPShapeDataBlock::Normal;
			break;
		case 2: // равномерный закон
			zakon = RPShapeDataBlock::Uniform;
			break;
		case 3: // треугольный
			zakon = RPShapeDataBlock::Triangular;
			break;
		case 4: // экспоненциальный
			zakon = RPShapeDataBlock::Exp;
			break;
	}
```

Часть файла rdoprocess_shape_process_dlg1_MJ.cpp

```
void RPShapeProcessDlg1_MJ::OnCbnSelchange1()
{
int cur = m_gtype.GetCurSel();

switch(cur) // определяем активные окна исходя из закона
{

…

case 3: // труегольный
	m_proc_dlg1_exp_text_MJ.SetWindowText(_T("Левая граница"));
	m_proc_dlg1_disp_text_MJ.ShowWindow(SW_SHOW);
	m_proc_dlg1_disp_text_MJ.SetWindowText(_T("Высота"));
	m_proc_dlg1_disp_control_MJ.ShowWindow(SW_SHOW);
	m_proc_dlg1_max_text_MJ.ShowWindow(SW_SHOW);
	m_proc_dlg1_max_text_MJ.SetWindowText(_T("Правая граница"));
	m_proc_dlg1_max_control_MJ.ShowWindow(SW_SHOW);
			break;

…

}
}
```

Приложение 5. Программный код формирования калков для передачи данных в генератор псевдослучайных чисел из графического интерфейса

Часть файла procgui.cpp

```
rdoRuntime::LPRDOCalcFunctionCall ProcGUICalc::getTriangularCalc(ruint base, double
arg1, double arg2, double arg3)
{
	PTR(rdoRuntime::RandGeneratorTriangular) pGenerator = new
rdoRuntime::RandGeneratorTriangular();
	rdoRuntime::LPRDOCalcSeqInit pInitCalc =
rdo::Factory<rdoRuntime::RDOCalcSeqInit>::create(base, pGenerator);
	ASSERT(pInitCalc);
	m_pRuntime->addInitCalc(pInitCalc);
	rdoRuntime::LPRDOCalcSeqNext pNextCalc =
rdo::Factory<rdoRuntime::RDOCalcSeqNextTriangular>::create(pGenerator);
	ASSERT(pNextCalc);
	rdoRuntime::LPRDOCalcFunctionCall pFuctionCall =
rdo::Factory<rdoRuntime::RDOCalcFunctionCall>::create(pNextCalc);
	ASSERT(pFuctionCall);
	rdoRuntime::LPRDOCalcConst pArg1 =
rdo::Factory<rdoRuntime::RDOCalcConst>::create(arg1);
	rdoRuntime::LPRDOCalcConst pArg2 =
rdo::Factory<rdoRuntime::RDOCalcConst>::create(arg2);
	rdoRuntime::LPRDOCalcConst pArg3 =
rdo::Factory<rdoRuntime::RDOCalcConst>::create(arg3);
	pFuctionCall->addParameter(pArg1);
	pFuctionCall->addParameter(pArg2);
	pFuctionCall->addParameter(pArg3);
	return pFuctionCall;
}
```

Приложение 6. Передача данных из интерфейсной части в алгоритмическую. Первичная проверка параметров

Часть файла sequence.cpp

```
// ----------------------------------------------------------------------------
// -------------------- RDOCalcSeqNextTriangular
// ----------------------------------------------------------------------------
RDOValue RDOCalcSeqNextTriangular::getNextValue(CREF(LPRDORuntime) pRuntime)
{
	RDOValue from = pRuntime->getFuncArgument(0);
	RDOValue top  = pRuntime->getFuncArgument(1);
	RDOValue to   = pRuntime->getFuncArgument(2);
	if ((from > top) || (top > to))
	{
		pRuntime->error(rdo::format(_T("Для треуголного закона распределения
нужно указать левую границу, точку под высотой треугольника, правую границу: %s, %s,
%s")
			, from.getAsString().c_str()
			, top.getAsString().c_str()
			, to.getAsString().c_str()
		, this));
	}
	return m_gen->next(from.getDouble(), top.getDouble(), to.getDouble());
}
```

Приложение 7. Программный код алгоритмической части

Часть файла rdo_random_distribution.inl

```
// --------------------------------------------------------------------------------
// -------------------- RandGeneratorTriangular
// --------------------------------------------------------------------------------
inline RandGeneratorTriangular::RandGeneratorTriangular(long int seed)
	: RandGenerator(seed)
{}

inline double RandGeneratorTriangular::next(double from, double top, double to)
{
	double result;
	if (u01() > (top-from)/(to-from))
	{
		result = -(to-top)*(sqrt(u01()) - 1);
	}
	else
	{
		result = (top-from)*(sqrt(u01()) - 1);
	}
	return result + top;
}
```

Приложение 8. Тестовая модель-прототип

Файл DZ.rtp:

```
$Resource_type ресурсы: permanent
$Parameters
        состояние           : ( Свободен, Занят )
$End
```

Файл DZ.rss:

```
$Resources
        ресурс              : ресурсы trace Свободен
$End
```

Файл DZ.evn:

```
$Pattern Событие_1: event trace
$Relevant_resources
        _ресурс: ресурс Keep
$Body
        _ресурс
        Convert_event
                Событие_2.Planning(Time_now +
                        Последовательность_1(160,190,200));
$End

$Pattern Событие_2: event trace
$Relevant_resources
        _ресурс: ресурс Keep
$Body
        _ресурс
        Convert_event
                состояние = Занят;
$End
```

Файл DZ.fun:

```
$Sequence Последовательность_1 : real
$Type = triangular 1234
$End
```

Файл DZ.smr:

```
Событие_1.planning(Time_now)
Terminate_if ресурс.состояние == Занят
```

Файл DZ.pmd:

```
$Results
    Длительность        : get_value Time_now
$End
```

Приложение 9. Программный код автоматического теста

```
/*!
  \copyright (c) RDO-Team, 2011
  \file      main.cpp
  \authors   Урусов Андрей (rdo@rk9.bmstu.ru)
  \authors   Клеванец Игорь (impus@hotbox.ru)
  \date      2.10.2011
  \brief     Тест законов распределения
  \indent    4T
*/

// ---------------------------------------------------------------------------- PCH
// ---------------------------------------------------------------------- INCLUDES
#define BOOST_TEST_MODULE RDOSequencesTest
#include <iostream>
#include <fstream>
#include <vector>
#include <math.h>
#include <boost/test/included/unit_test.hpp>
#include <boost/bind.hpp>
#include <boost/function.hpp>
// ---------------------------------------------------------------------- SYNOPSIS
#include "utils/rdofile.h"
#include "utils/platform.h"
#include "simulator/runtime/rdo_random_distribution.h"
// --------------------------------------------------------------------------------

#ifdef COMPILER_VISUAL_STUDIO
      #define  __SCANF  sscanf_s
#else  // not COMPILER_VISUAL_STUDIO
      #define  __SCANF  sscanf
#endif // COMPILER_VISUAL_STUDIO

typedef std::vector<double> Container;
typedef std::vector<ruint>  ContainerInt;
typedef const tstring contstr;

const long int g_seed                 = 123456789;                  //!< база
генератора
contstr        g_fileNormalName       = _T("data_normal.txt");      //!< файл данных
contstr        g_fileUniformName      = _T("data_uniform.txt");     //!< файл данных
contstr        g_fileExponentialName  = _T("data_exponential.txt"); //!< файл данных
contstr        g_fileTriangularName   = _T("data_trinagular.txt");  //!< файл данных

const ruint    g_count                = 100000;                     //!< количество
генерируемых данных
const double   g_main                 = 10.0;                       //!< параметр
закона экспоненциального и нормального
const double   g_var                  = 1.0;                        //!< параметр
закона нормального
const double   g_from                 = 1.0;                        //!< параметр
закона равномерного и треугольного
const double   g_to                   = 7.0;                        //!< параметр
закона равномерного и треугольного
const double   g_top                  = 5.0;                        //!< параметр
закона треугольного
const ruint    g_precision            = 20;                         //!< точность
вещественного числа при выводе в поток

const ruint    g_countOfExamples      = 2000;                       //!< количество
чисел в выборке
const ruint    g_countOfR             = 39;                         //!< число
разрядов
```

```
const double   pi                       = 3.141592653;                //!< фундаментальная константа
const double   g_ksiEtalon              = 50.9985;                    //!< табличное значение. 95% вероятность того, что это действительно тот самый закон распределения

// ------------------------------------------------------------------------------------
// -------Templates
// ------------------------------------------------------------------------------------
template <class T, class F, class contstr>
void onGenerateData(F binder, contstr g_fileName)
{
	if (rdo::File::exist(g_fileName.c_str()))
		return;

	T sequence(g_seed);
	Container test;
	test.reserve(g_count);

	for (ruint i = 0; i < g_count; ++i)
	{
		test.push_back(binder.operator()(&sequence));
	}

	std::ofstream stream(g_fileName.c_str());
	stream.precision(g_precision);
	STL_FOR_ALL(test, it)
	{
		stream << *it << std::endl;
	}
}

template <class T, class F, class contstr>
void onCheckData(F binder, contstr g_fileName)
{
	std::ifstream stream(g_fileName.c_str());
	BOOST_CHECK(stream.good());

	Container test;
	test.reserve(g_count);
	T sequence(g_seed);
	for (ruint i = 0; i < g_count; ++i)
	{
		test.push_back(binder.operator()(&sequence));
	}

	stream.precision(g_precision);
	STL_FOR_ALL(test, it)
	{
		BOOST_CHECK(stream.good());
		tstring str;
		stream >> str;

		double val;
		BOOST_CHECK(__SCANF(str.c_str(), _T("%lf"), &val) == 1);
		BOOST_CHECK(val == *it);
			if (val != *it)
		{
			std::cout.precision(g_precision);
			std::cout << *it << std::endl;
			std::cout << val << std::endl;
		}
	}
}

template <class T,class F>
```

```
double  area (F binder, double n, double m)
{
	double k = 1;
	double S1 = 1;
	double S2 = 0;
	ruint t = 10;
	while (fabs(S1-S2)/S1 > 0.01)
	{
		S2 = S1;
		S1 = 0;
		for (ruint g = 0; g < t + 1; ++g)
		{
			if ((g == 0) || (g == t - 1))
				k = 0.5;
			S1 += k*(binder.operator()(n + g*(m-n)/t));
			k = 1;
		}
		S1 *= (m-n)/t;
		t  *= 10;
	}
	return S1;
}

template <class T, class G, class F, class S>
void onCheckKsi(F binder, S binderSeq, double left, double right)
{
	Container x;
	x.reserve(g_countOfR + 1);
	double elem = (right-left)/(g_countOfR*1.0);    //расстояние между точками на
прямой

	for (ruint i = 0; i < g_countOfR + 1; ++i)
	{
		x.push_back(left + elem*i);
	}

	Container vb;                                                    //контейнер
для хранения выборки
	vb.reserve(g_countOfExamples);

	G sequence(g_seed);                                              //выборка
	for (ruint i = 0; i < g_countOfExamples; ++i)
	{
		vb.push_back(binderSeq.operator()(&sequence));
	}

	Container f_vb;                                                  //контейнер
для хранения количества попаданий на интервал
	f_vb.reserve(g_countOfR);

	for(ruint i = 0; i < g_countOfR; ++i)                //нахождение количества
попаданий на интервал
	{
		ruint freq = 0;
		for(ruint k = 0; k < g_countOfExamples; ++k)
		{
			if((vb[k] > x[i]) & (vb[k] <= x[i+1]))
			{
				++freq;
			}
		}
		f_vb.push_back(freq);
	}

	Container F_etalon;
```

```
	F_etalon.reserve(g_countOfR);

	for (ruint i = 0; i < g_countOfR; ++i)
	{
		F_etalon.push_back(area<T>(binder, x[i], x[i+1]));
	}

	double ksi = 0;
	for(ruint i = 0; i < g_countOfR; ++i)
	{
		double ksiTemp = F_etalon[i]*g_countOfExamples;
		ksi += (f_vb[i] - ksiTemp)*(f_vb[i] - ksiTemp)/ksiTemp;
	}
	BOOST_CHECK(ksi <= g_ksiEtalon);
			if (ksi >  g_ksiEtalon)
	{
		std::cout << ksi << std::endl;
	}
}
// ------------------------------------------------------------------------------

class SequenceNormal
{
public:
	SequenceNormal(double main, double var)
		: m_main(main)
		, m_var (var )
	{}

	double get(double x) const
	{
		return 1/(sqrt(2*pi)*m_var*exp((x - m_main)*(x -
m_main)/(2*m_var*m_var)));
	}

private:
	double m_main;
	double m_var;
};

class SequenceExponential
{
public:
	SequenceExponential(double main)
		: m_main(main)
	{}

	double get(double x) const
	{
		return 1/(m_main*exp(x/m_main));
	}

private:
	double m_main;
};

class SequenceUniform
{
public:
	SequenceUniform(double min, double max)
		: m_min(min)
		, m_max(max)
	{}

	double get(double x) const
```

```
	{
		UNUSED(x);
		return 1/(m_max-m_min);
	}

private:
	double m_min;
	double m_max;
};

class SequenceTriangular
{
public:
	SequenceTriangular(double min, double top, double max)
		: m_min(min-top)
		, m_top(top)
		, m_max(max-top)
	{}

	double get(double x) const
	{
		x -= m_top;
		double temp;
		if (x < 0)
		{
			temp = -2*x/((m_max - m_min)*m_min) + 2/(m_max - m_min);
		}
		else
		{
			temp = -2*x/((m_max - m_min)*m_max) + 2/(m_max - m_min);
		}
		return temp;
	}

private:
	double m_min;
	double m_top;
	double m_max;
};

BOOST_AUTO_TEST_SUITE(RDOSequencesTest)

// --------------------------------------------------------------------------------
// -------Normal sequence
// --------------------------------------------------------------------------------
BOOST_AUTO_TEST_CASE(RDONormalTestCreate)
{
	onGenerateData<rdoRuntime::RandGeneratorNormal>
		(boost::bind(&rdoRuntime::RandGeneratorNormal::next, _1, g_main, g_var),
g_fileNormalName);
}

BOOST_AUTO_TEST_CASE(RDONormalTestCheck)
{
	onCheckData<rdoRuntime::RandGeneratorNormal>
		(boost::bind(&rdoRuntime::RandGeneratorNormal::next, _1, g_main, g_var),
g_fileNormalName);

	SequenceNormal normal(g_main, g_var);
	onCheckKsi<SequenceNormal, rdoRuntime::RandGeneratorNormal>
		(boost::bind(&SequenceNormal::get, normal, _1),
		boost::bind(&rdoRuntime::RandGeneratorNormal::next, _1, g_main, g_var),
		g_main-4*g_var,
		g_main+4*g_var);
}
```

```
// --------------------------------------------------------------------------------

// --------------------------------------------------------------------------------
// -------Uniform sequence
// --------------------------------------------------------------------------------
BOOST_AUTO_TEST_CASE(RDOUniformTestCreate)
{
	onGenerateData<rdoRuntime::RandGeneratorUniform>
		(boost::bind(&rdoRuntime::RandGeneratorUniform::next, _1, g_from, g_to),
g_fileUniformName);
}

BOOST_AUTO_TEST_CASE(RDOUniformTestCheck)
{
	onCheckData<rdoRuntime::RandGeneratorUniform>
		(boost::bind(&rdoRuntime::RandGeneratorUniform::next, _1, g_from, g_to),
g_fileUniformName);

	SequenceUniform uniform(g_from, g_to);
	onCheckKsi<SequenceUniform, rdoRuntime::RandGeneratorUniform>
		(boost::bind(&SequenceUniform::get, uniform, _1),
		boost::bind(&rdoRuntime::RandGeneratorUniform::next, _1, g_from, g_to),
		g_from,
		g_to);
}
// --------------------------------------------------------------------------------

// --------------------------------------------------------------------------------
// -------Exponential sequence
// --------------------------------------------------------------------------------
BOOST_AUTO_TEST_CASE(RDOExponentialTestCreate)
{
	onGenerateData<rdoRuntime::RandGeneratorExponential>
		(boost::bind(&rdoRuntime::RandGeneratorExponential::next, _1, g_main),
g_fileExponentialName);
}

BOOST_AUTO_TEST_CASE(RDOExponentialTestCheck)
{
	onCheckData<rdoRuntime::RandGeneratorExponential>
		(boost::bind(&rdoRuntime::RandGeneratorExponential::next, _1, g_main),
g_fileExponentialName);

	SequenceExponential exponential(g_main);
	onCheckKsi<SequenceExponential, rdoRuntime::RandGeneratorExponential>
		(boost::bind(&SequenceExponential::get, exponential, _1),
		boost::bind(&rdoRuntime::RandGeneratorExponential::next, _1, g_main),
		0,
		7*g_main);
}
// --------------------------------------------------------------------------------

// --------------------------------------------------------------------------------
// -------Triangular sequence
// --------------------------------------------------------------------------------
BOOST_AUTO_TEST_CASE(RDOTriangularTestCreate)
{
	onGenerateData<rdoRuntime::RandGeneratorTriangular>
		(boost::bind(&rdoRuntime::RandGeneratorTriangular::next, _1, g_from,
g_top, g_to), g_fileTriangularName);
}

BOOST_AUTO_TEST_CASE(RDOTriangularTestCheck)
{
	onCheckData<rdoRuntime::RandGeneratorTriangular>
```

```
            (boost::bind(&rdoRuntime::RandGeneratorTriangular::next, _1, g_from,
g_top, g_to), g_fileTriangularName);

      SequenceTriangular triangular(g_from, g_top, g_to);
      onCheckKsi<SequenceTriangular, rdoRuntime::RandGeneratorTriangular>
            (boost::bind(&SequenceTriangular::get, triangular, _1),
            boost::bind(&rdoRuntime::RandGeneratorTriangular::next, _1, g_from,
g_top, g_to),
            g_from,
            g_to);
}
// --------------------------------------------------------------------------------

BOOST_AUTO_TEST_SUITE_END()
```