Министерство науки и высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Теоретической и Прикладной Информатики

Лабораторная работа №3 «ОБЪЕКТЫ»

по предмету

«**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Факультет: прикладной математики и информатики

Группа: ПМИ-12

Бригада: 9

Студенты: Кулаков А.В.

Попов С.Н.

Кораблев К.В.

Преподаватели: Дворецкая В.К.

Новосибирск, 2023 **Цель работы**

Изучить понятие объекта, средства его создания и уничтожения, а также отношения между объектами.

**Постановка задачи**

1. Ознакомиться с вопросами использования конструкторов и деструкторов для создания и уничтожения объектов и с отношениями между объектами.

2. Модифицировать программу, разработанную в лабораторной работе №2, так чтобы в ней были определены классы, реализующие смесь распределений и эмпирическое распределение.

2.1. Объект класса, реализующего **смесь**, должен содержать в качестве основных атрибутов два объекта класса основного распределения (отношение *композиции* объектов) и параметр смеси. Основные требования к интерфейсу класса – те же, что для класса основного распределения.

Однако смесь может явно пользоваться **только** функциями-членами класса основного распределения, реализующими пп. 2.3.4–2.3.7 задания к лабораторной работе №2. В частности, обеспечение чтения и модификации атрибутов агрегированных объектов в обязанности класса не входит, для этой цели он **должен** предоставлять доступ к каждому из агрегированных объектов целиком. Также в обязанности класса не входит конструирование агрегированных объектов по наборам их параметров, конструктор класса **должен** получать в качестве аргументов объекты класса основного распределения. Если какая-то функциональность класса смеси можетбыть передана классу основного распределения, она **должна**быть передана.

2.2. Класс, реализующий **эмпирическое** распределение, должен содержать в качестве основных атрибутов массив с выборкой, массив, задающий эмпирическую плотность, и размеры массивов. Основные требования к интерфейсу класса – те же, что для класса основного распределения.

Память под массивы должна **динамически** распределяться и освобождаться.

Набор *конструкторов* должен обеспечивать инициализацию массива с выборкой как внешними данными (например, из файла), так и при помощи функций моделирования случайных величин. Для реализации последней возможности в конструкторы должны передаваться ссылки на объекты разрабатываемых классов (*связи* или *клиент-серверные* отношения между объектами), т.е. должны иметься три конструктора для передачи ссылки на объект каждого из трех разрабатываемых классов. Кроме того, необходимо обеспечить класс конструктором копирования и оператором присваивания, осуществляющими *глубокое копирование*.

2.3. При разработке конструкторов максимально использовать **инициализаторы**конструктора.

2.4. Объекты всегда должны находиться в корректном состоянии. При возникновении ошибок необходимо генерировать *исключительные ситуации*, а ихобработку осуществлять **вне** функций-членов классов. Диалог с пользователем (если он предполагается) следует вести **вне** функций-членов классов.

2.5. Настоятельно рекомендуется во всех разрабатываемых классах (основного распределения, смеси и эмпирического распределения) сделать *одинаковые сигнатуры* для функций-членов, реализующих пп. 2.3.4–2.3.7 задания к лабораторной работе №2.

2.6. Разработать функцию (функции), тестирующую возможности новых классов.

**Алгоритм решения задачи**

Поскольку все методы, необходимые для вычисления характеристик распределения и моделирование случайной величины были реализованы в процессе выполнения лабораторной работы №1, нам необходимо оформить их в виде классов.

В закрытой части класса будут описаны следующие атрибуты:

Смесь распределений: компоненты смеси (2 стандартных распределения) и параметр смеси.

Эмпирическое распределение: массив с выборкой, массив с эмпирическими плотностями, размер массива с выборкой, кол-во промежутков, а также закрытые функции, предназначенные только для промежуточных вычислений.

В открытой части классов будет определено следующее:

Конструкторы класса с заданием параметров, с чтением из файла, конструктор со значениями по умолчанию. Методы класса: задание и получение атрибутов, вычисление характеристик распределений, вычисление плотности в заданной точке, построение выборки для графиков, сохранение в файл атрибутов.

**Работа с основным распределением**

Плотность стандартного распределения мы можем посчитать с помощью данной стандартной формулы. Однако мы получим неправильные результаты, если не учтем сдвиг-масштабные преобразования.



По данной формуле мы сможем посчитать плотность с учетом сдвига и масштаба.

Моделирование случайной величины происходит по следующему алгоритму

Шаг 1. Получить реализацию  случайной величины, равномерно распределенной на интервале (0, 1). Если , перейти на шаг 2, иначе перейти на шаг 4.

Шаг 2. Получить реализацию  нормальной случайной величины. Для этого получить реализации ,  случайной величины, равномерно распределенной на интервале (0, 1), и вычислить  по одной из двух формул

, .

Шаг 3. Если  принадлежит интервалу [–, ], то это реализация целевой случайной величины, иначе перейти на шаг 2.

Шаг 4. Получить реализацию  случайной величины, равномерно распределенной на интервале (0, 1). Вычислить

.

Шаг 5. Если , то  – реализация целевой случайной величины, иначе реализацией целевой случайной величины является .

Для учета сдвига и масштаба необходимо так же сдвинуть полученные значения.



Варианты распределений в списке вариантов являются симметричными унимодальными и имеют нулевые матожидание и коэффициент асимметрии.

При сдвиг-масштабных преобразованиях матожидание увеличивается на величину , дисперсия, равная увеличивается в  раз, коэффициент эксцесса, равный  не изменяется, как и коэффициент асимметрии, в случае которого мы можем возвращать ноль.

Вероятность попадания в центральный интервал 

**Смесь распределений**

Функция плотности для смеси распределений:



Где p – параметр смеси.

Моделирование случайной величины для смеси распределений

Шаг 1. Получить реализацию  случайной величины, имеющей равномерное распределение на интервале (0, 1). Если , перейти на шаг 2, иначе перейти на шаг 3.

Шаг 2. Получить реализацию  случайной величины с плотностью . Число  и будет реализацией целевой случайной величины.

Шаг 3. Получить реализацию  случайной величины с плотностью . Число  и будет реализацией целевой случайной величины.

Математическое ожидание вычисляется по формуле:



Дисперсия вычисляется по формуле:



Коэффициента асимметрии вычисляется по формуле:

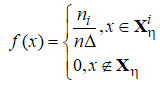


Коэффициент эксцесса:



**Эмпирическое распределение**

Эмпирической плотностью распределения называют функцию

.

Плотность формируется на основе выборки, следовательно, необходимо сгенерировать выбору случайной величины, распределенной по непрерывному закону. Затем сортируем выборку получившуюся выборку.

Далее необходимо определить число промежутков *k*, которое мы высчитываем по формуле Стёрджесса, округляя в большую сторону,

.

На основании предыдущих действий рассчитывается значение длины каждого промежутка:

.

После мы создаем массив, хранящий границы интервалов . Так как для реализации функции плотности эмпирического распределения необходимо не только длина каждого промежутка и количество элементов выборки, но и количество точек в интервале. Для получения этой информации мы проходимся по массиву границ и проверяем к какому интервалу принадлежит наш *х*. Затем ищем границы этого интервала и считаем в нем количество элементов выборки.

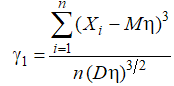
Математическое ожидание для эмпирического распределения:

D:\Study\Лабораторные работы\Прога\5 семестр\ЛР№1\эмп. мат ожидание.png,

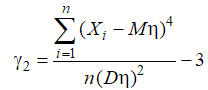
Дисперсия для эмпирического распределения:

,

Коэффициент асимметрии для эмпирического распределения:

,

Коэффициент эксцесса для эмпирического распределения:

.

**Описание программы**

**Модуль main.cpp**

Содержит пользовательское меню, которое предлагает все функции и методы работы с 3 распределениями, предусмотрены проверки входных данных и обработки исключений и вывод сгенерированных данных в файл.

**Функции:**

menu() – главное меню, в котором пользователь выбирает, с каким распределением работает.

std\_menu() – меню работы со стандартным распределением.

mix\_menu() – меню работы со смесью распределений.

emp\_menu() – меню работы с эмпирическим распределением.

calculate() – меню вычислений плотности, характеристик, моделирования случайной величины и вывод в файл.

**Модуль mixture\_pareto\_class.cpp**

Содержит описание всех конструкторов и методов класса для работы со смесью распределений.

**Открытая часть:**

**Конструкторы:**

mixture() – конструктор с параметрами по умолчанию.

mixture(Pareto& std1, Pareto & std2, double p) – конструктор класса из двух существующих объектов стандартного распределения.

mixture(std::string file) – получение атрибутов распределения из файла.

**Методы класса:**

Pareto& component1(), component2() – получение первого и второго распределений соответственно из объекта смеси распределений.

save\_attr() – сохранение текущих атрибутов в файл.

SetP() – задание параметра смеси.

GetP() – получение параметра смеси.

density() – вычисление плотности распределения в точке.

math\_expect() – вычисление математического ожидания.

dispersion() – вычисление дисперсии.

asymmetry() – вычисление коэффициента асимметрии.

excess() – вычисление коэффициента эксцесса.

modeling() – моделирование случайной величины, распределенной по закону заданной смеси.

modeling() – моделирование случайной величины, распределенной по закону заданной смеси.

generate\_x() – генерация выборки распределенной по закону заданной смеси заданного размера.

generate\_xy() – генерация выборки смоделированных случайных величин в формате x y.

xy\_output() – вывод сгенерированной таблицы аргументов и значений в файл.

**Закрытая часть:**

randomizer() – генерация случайной величины на промежутке от 0 до 1.

**Модуль mixture\_pareto\_class.h**

Содержит объявление класса и всех его методов, описанных выше.

**Модуль emp\_pareto\_class.cpp**

Описание методов работы и конструкторов для эмпирического распределения.

**Открытая часть:**

**Конструкторы:**

Empirical(Pareto& std, int n, int \_k) – конструктор класса на основе стандартного распределения.

Empirical(mixture& mix, int n, int \_k) – конструктор класса на основе смеси распределений.

Empirical(Empirical& ed, int n, int \_k) – конструктор класса на основе эмпирического распределения.

Empirical(Empirical& emp) – конструктор копирования.

Empirical(std::vector<double>& selection) – конструктор на основе любой выборки.

Empirical(std::string file) – получение данных из файла.

~Empirical() – деструктор класса.

**Методы класса:**

Empirical& operator = (const Empirical& emp) – перегрузка оператора присваивания.

SetK() – задание кол-ва интервалов эмпирического распределения.

GetSelection() – получение выборки.

GetEmpDensity() – получение значений плотности распределения.

GetSize() – получение размера выборки.

GetIntervalsNumber() – получение кол-ва интервалов эмпирического распределения.

density() – вычисление плотности распределения в точке.

math\_expect() – вычисление математического ожидания.

dispersion() – вычисление дисперсии.

asymmetry() – вычисление коэффициента асимметрии.

excess() – вычисление коэффициента эксцесса.

rand\_var() – моделирование случайной величины по эмпирическому закону распределения.

generate\_xy() – генерация таблицы значений в формате x y.

emp\_output() – вывод значений в файл.

**Закрытая часть:**

delta\_calc() – вычисление длины интервала.

create\_intervals() – создание интервалов.

create\_empirical\_density() – заполнение массива со значениями плотности.

randomizer() – генерация случайной величины на промежутке от 0 до 1.

generate\_x() – генерация выборки смоделированных значений.

qumulative\_probability() – вычисление кумулятивной вероятности.

top\_bound() – вычисление верхней оценки во время моделирования случайной величины.

**Модуль emp\_pareto\_class.h**

Содержит объявление класса и всех его методов, а также подключение необходимых библиотек.

**Модуль test.cpp**

Содержит юнит-тесты для проверки корректности вычислений.

**Код программы**

**Main.cpp**

#include "std\_pareto\_class.h"

#include "mixture\_pareto\_class.h"

#include "emp\_pareto\_class.h"

#define CATCH\_CONFIG\_MAIN

#include "catch.hpp"

using namespace std;

void calculate(Pareto& std);

void calculate(mixture& mix);

void calculate(Empirical& emp);

int run\_unit\_tests(int argc, char\*\* argv) {

int result = Catch::Session().run(argc, argv);

return result;

}

void std\_menu() {

int input=0;

double v, mu, lambda;

Pareto std;

fstream file;

while (true) {

cout << "Действия:" << endl << "1)Ввести параметры вручную" << endl << "2)Загрузить из файла" << endl << "3)Просмотреть текущие параметры"<<endl << "4)Продолжить..." << endl << "5)Назад"<<endl;

cin >> input;

switch (input) {

case 1:

cout << "Введите парамеры v, mu, lambda" << endl;

cin >> v >> mu >> lambda;

std = Pareto(v, mu, lambda);

std.save\_atr("atributes.txt");

break;

case 2:

std = Pareto("atributes.txt");

break;

case 3:

cout << "Текущие параметры Парето-нормального распределения:" << endl << "Параметр формы = " << std.GetForm() << ", " << "параметр сдвига = " << std.GetMu() << ", " << "параметр масштаба = " << std.GetLambda() << endl;

break;

case 4:

calculate(std);

break;

case 5:

return;

}

}

}

void mix\_menu() {

int input = 0;

double v1, mu1, lambda1,v2,mu2,lambda2,p;

Pareto std1;

Pareto std2;

mixture mix;

fstream file;

while (true) {

cout << "Действия:" << endl << "1)Ввести параметры вручную" << endl << "2)Загрузить из файла" << endl << "3)Просмотреть текущие параметры" << endl << "4)Продолжить..." << endl << "5)Назад" << endl;

cin >> input;

switch (input) {

case 1:

cout << "Введите парамеры v1, mu1, lambda1, v2, mu2, lambda2, p" << endl;

cin >> v1 >> mu1 >> lambda1>>v2>>mu2>>lambda2>>p;

std1 = Pareto(v1, mu1, lambda1);

std2 = Pareto(v2, mu2, lambda2);

mix = mixture(std1, std2, p);

mix.save\_atr("mixtr\_attr.txt");

break;

case 2:

mix = mixture("mixtr\_attr.txt");

break;

case 3:

cout << "Текущие параметры смеси распределений:" << endl <<

"Параметр формы первого распределения: " << mix.component1().GetForm() << ", параметр сдвига: " << mix.component1().GetMu() << ", параметр масштаба: " << mix.component1().GetLambda() << endl

<< "Параметр формы второго распределения: " << mix.component2().GetForm() << ", параметр сдвига: " << mix.component2().GetMu() << ", параметр масштаба: " << mix.component2().GetLambda() << endl;

cout << "Параметр смеси: " << mix.GetP() << endl;

break;

case 4:

calculate(mix);

case 5:

return;

}

}

}

void calculate(Pareto& std) {

int input = 0, n=2000;

double x = 0;

while (true) {

cout << "Дальнейшие действия: " << endl << "1)Посмотреть текущие параметры" << endl << "2)Вычислить плотность в точке" << endl <<

"3)Вычислить характеристики" << endl << "4)Строим график" << endl << "5)Выход" << endl;

cin >> input;

switch (input) {

case 1:

cout << "Параметр формы = " << std.GetForm() << ", параметр сдвига = " << std.GetMu() << ", параметр масштаба = " << std.GetLambda() << endl;

break;

case 2:

cout << "Введите точку x" << endl;

cin >> x;

cout<<"Плотность в выбранной точке = "<<std.pareto\_std\_distribution(x) << endl;

break;

case 3:

cout << "Мат.ожидание = " << std.math\_expect() << endl;

cout << "Дисперсия = " << std.dispersion() << endl;

cout << "Асиммметрия = " << std.asymmetry() << endl;

cout << "Эксцесс = " << std.excess\_koef() << endl;

break;

case 4:

cout << "Введите размер выборки для графика:" << endl;

cin >> n;

std.xy\_output(n);

cout << "Данные успешно получены!" << endl;

break;

case 5:

return;

}

}

}

void calculate(mixture& mix) {

int input = 0, n = 2000;

double x = 0;

while (true) {

cout << "Дальнейшие действия: " << endl << "1)Посмотреть текущие параметры" << endl << "2)Вычислить плотность в точке" << endl <<

"3)Вычислить характеристики" << endl << "4)Строим график" << endl << "5)Выход" << endl;

cin >> input;

switch (input) {

case 1:

cout << "Первое распределение:" << endl << endl << "Параметр формы = " << mix.component1().GetForm() << ", параметр сдвига = " <<

mix.component1().GetMu() << ", параметр масштаба = " << mix.component1().GetLambda() << endl<<endl;

cout<<"Второе распределение: "<<endl<<endl<< "Параметр формы = " << mix.component2().GetForm() << ", параметр сдвига = " <<

mix.component2().GetMu() << ", параметр масштаба = " << mix.component2().GetLambda() << endl;

cout << "Параметр смеси = " << mix.GetP() << endl;

break;

case 2:

cout << "Введите точку x" << endl;

cin >> x;

cout << "Плотность в выбранной точке = " << mix.density(x) << endl;

break;

case 3:

cout << "Мат.ожидание = " << mix.math\_expect() << endl;

cout << "Дисперсия = " << mix.dispersion() << endl;

cout << "Асиммметрия = " << mix.asymmetry() << endl;

cout << "Эксцесс = " << mix.excess() << endl;

break;

case 4:

cout << "Введите размер выборки для графика:" << endl;

cin >> n;

mix.xy\_output(n);

cout << "Данные успешно получены!" << endl;

break;

case 5:

return;

}

}

}

void calculate(Empirical& emp) {

int input = 0, n = 2000;

double x = 0;

vector<double> sample;

vector<pair<double, double>> graph;

ofstream output;

while (true) {

cout << "Дальнейшие действия: " << endl << "1)Вывести текущие параметры" << endl << "2)Вычислить плотность в точке" << endl <<

"3)Вычислить характеристики" << endl << "4)Построить график" << endl << "5)Выход" << endl;

cin >> input;

switch (input) {

case 1:

cout << "Размер выборки = " << emp.GetSize() << endl;

cout << "Кол-во интервалов = " << emp.GetIntervalsNumber() << endl;

break;

case 2:

cout << "Введите точку x" << endl;

cin >> x;

cout << "Плотность в выбранной точке = " << emp.density(x) << endl;

break;

case 3:

cout << "Мат.ожидание = " << emp.math\_expect() << endl;

cout << "Дисперсия = " << emp.dispersion() << endl;

cout << "Асиммметрия = " << emp.asymmetry() << endl;

cout << "Эксцесс = " << emp.excess() << endl;

break;

case 4:

emp.emp\_output(output);

cout << "Успешно!" << endl;

break;

case 5:

return;

}

}

}

void emp\_menu() {

int input = 0;

double v1, mu1, lambda1,v2,mu2,lambda2,p;

Pareto std1;

Pareto std2;

mixture mix;

Empirical emp(mix,10,1);

fstream file;

while (true) {

cout << "Действия(на основе чего строим эмпирическое распределение):" << endl << "1)Стандартное" << endl << "2)Смесь" << endl << "3)Назад" << endl;

cin >> input;

switch (input) {

case 1:

cout << "Введите парамеры v, mu, lambda" << endl;

cin >> v1 >> mu1 >> lambda1;

std1 = Pareto(v1, mu1, lambda1);

emp = Empirical(std1, 2000, 1);

calculate(emp);

break;

case 2:

cout << "Введите парамеры v1, mu1, lambda1, v2, mu2, lambda2, p" << endl;

cin >> v1 >> mu1 >> lambda1 >> v2 >> mu2 >> lambda2 >> p;

std1 = Pareto(v1, mu1, lambda1);

std2 = Pareto(v2, mu2, lambda2);

mix = mixture(std1, std2, p);

emp = Empirical(mix, 2000, 1);

calculate(emp);

break;

case 3:

return;

}

}

}

void menu() {

int input;

while (true) {

cout << "Выберите распределение:" << endl << "1)Стандартное" << endl << "2)Смесь распределений" << endl << "3)Эмпирическое"<<endl<<"4)Выход" << endl;

cin >> input;

switch (input) {

case 1:

std\_menu();

break;

case 2:

mix\_menu();

break;

case 3:

emp\_menu();

break;

case 4:

return;

}

}

}

int main(int argc, char\*\* argv) {

setlocale(LC\_ALL, "Russian");

try {

//

menu();

}

catch (runtime\_error ex) {

cout << "Ошибка при работе с файлами: " << ex.what();

}

catch (int error) {

if (error == 1) {

cout << "Параметры распределения указаны неккоректно" << endl;

}

}

//run\_unit\_tests(argc, argv);

}

**Mixture\_pareto\_class.cpp**

#include "mixture\_pareto\_class.h"

//открытые функции

void mixture::SetP(double \_p) {

if (\_p < 0 || \_p > 1)

{

throw 1;

}

else {

this->p = \_p;

}

}

double mixture::GetP() {

return this->p;

}

Pareto& mixture::component1() {

return pareto1;

}

Pareto& mixture::component2() {

return pareto2;

}

void mixture::save\_atr(std::string file) {//сохранение атрибутов в файл

std::ofstream output;

output.open(file);

std::vector<double> buffer;

buffer.push\_back(this->component1().GetForm());

buffer.push\_back(this->component1().GetMu());

buffer.push\_back(this->component1().GetLambda());

buffer.push\_back(this->component2().GetForm());

buffer.push\_back(this->component2().GetMu());

buffer.push\_back(this->component2().GetLambda());

buffer.push\_back(this->GetP());

if (!output) {

throw std::runtime\_error("Невозможно найти файл");

};

for (int i = 0; i < 7; i++) {

output << buffer[i] << " ";

}

output.close();

}

double mixture::density(double x) {

return (1 - p) \* pareto1.pareto\_std\_distribution(x) + p \* pareto2.pareto\_std\_distribution(x);

}

double mixture::math\_expect() {

return (1 - p) \* pareto1.math\_expect() + p \* pareto2.math\_expect();

}

double mixture::dispersion() {

return (1 - p) \* (pow(pareto1.math\_expect(), 2) + pareto1.dispersion()) +

p \* (pow(pareto2.math\_expect(), 2) + pareto2.dispersion()) -

pow(math\_expect(), 2);

}

double mixture::asymmetry() {

return ((1 - p) \* (pow((pareto1.math\_expect() - math\_expect()), 3) + 3 \* (pareto1.math\_expect() - math\_expect()) \* pareto1.dispersion() + pow(pareto1.dispersion(), 3 / 2) \* pareto1.asymmetry()) +

p \* (pow((pareto2.math\_expect() - math\_expect()), 3) + 3 \* (pareto2.math\_expect() - math\_expect()) \* pareto2.dispersion() + pow(pareto2.dispersion(), 3 / 2) \* pareto2.asymmetry())) /

pow(dispersion(), 3 / 2);

}

double mixture::excess() {

return ((1 - p) \* (pow((pareto1.math\_expect() - math\_expect()), 4) + 6 \* pareto1.dispersion() \* pow((pareto1.math\_expect() - math\_expect()), 2) +

4 \* (pareto1.math\_expect() - math\_expect()) \* pow(pareto1.dispersion(), 3 / 2) \* pareto1.asymmetry() + pow(pareto1.dispersion(), 2) \* pareto1.excess\_koef()) +

p \* (pow((pareto2.math\_expect() - math\_expect()), 4) + 6 \* pareto2.dispersion() \* pow((pareto2.math\_expect() - math\_expect()), 2) +

4 \* (pareto2.math\_expect() - math\_expect()) \* pow(pareto2.dispersion(), 3 / 2) \* pareto2.asymmetry() + pow(pareto2.dispersion(), 2) \* pareto2.excess\_koef()) - 3) /

pow(dispersion(), 2);

}

double mixture::modeling() {

double r = randomizer();

if (r > p) {

return pareto1.pareto\_modeling();

}

else {

return pareto2.pareto\_modeling();

}

}

std::vector<double> mixture::generate\_x(int n) {

std::vector<double> x;

for (int i = 0; i < n; ++i) {

x.push\_back(modeling());

}

sort(x.begin(), x.end());

return x;

}

std::vector<std::pair<double,double>> mixture::generate\_xy(int n) {

std::vector<std::pair<double, double>> result;

auto x = this->generate\_x(n);

for (int i = 0; i < n; i++) {

result.push\_back(std::make\_pair(x[i], this->density(x[i])));

}

return result;

}

void mixture::xy\_output(int n) {

auto result = this->generate\_xy(n);

std::ofstream file("mixture\_xy\_output.txt");

for (int i = 0; i < n; i++) {

file << result[i].first << '\t' << result[i].second << std::endl;

}

file.close();

}

//конструкторы

mixture::mixture():p(0.5) {//конструктор по умолчанию

pareto1.SetForm(3.3);

pareto2.SetForm(3.3);

pareto1.SetLambda(1);

pareto2.SetLambda(2);

pareto1.SetMu(-2);

pareto2.SetMu(2);

}

mixture::mixture(Pareto& std1, Pareto& std2, double \_p)://конструктор из существующих двух стандратных распределений

pareto1(std1), pareto2(std2),p(\_p>0 and \_p<1 ? \_p:throw 1){}

mixture::mixture(std::string file) {//создание объекта с помощью чтения арибутов из файла

std::ifstream input;

input.open(file);

if (!input) {

throw std::runtime\_error("Невозможно найти файл");

};

if (!file.empty()) {

std::vector<double> buffer;

for (int i = 0; i < 7; i++) {

double buf;

input >> buf;

//std::cout << buf;

if (!input) {

throw std::runtime\_error("Неправильные данные");

}

else {

buffer.push\_back(buf);

}

}

pareto1.SetForm(buffer[0]);

pareto1.SetMu(buffer[1]);

pareto1.SetLambda(buffer[2]);

pareto2.SetForm(buffer[3]);

pareto2.SetMu(buffer[4]);

pareto2.SetLambda(buffer[5]);

this->SetP(buffer[6]);

if (pareto1.GetForm() <= 1 || pareto1.GetLambda() <= 0 || pareto2.GetForm() <= 1 || pareto2.GetLambda() <= 0 ) {

throw 1;

}

}

else {

throw std::runtime\_error("Файл пустой");

}

}

//закрытые функции

double mixture::randomizer(){

double r;

do r = (double)rand() / RAND\_MAX; while (r == 0 || r == 1);

return r;

}

**Mixture\_pareto\_class.h**

#pragma once

#include <iostream>

#include <cmath>

#include <vector>

#include <algorithm>

#include <fstream>

#include "std\_pareto\_class.h"

class mixture {

private:

double p;

Pareto pareto1;

Pareto pareto2;

double randomizer();

public:

mixture();

mixture(Pareto& std1, Pareto& std2, double p);

mixture(std::string file);

void SetP(double p);

double GetP();

Pareto& component1();

Pareto& component2();

void save\_atr(std::string file);

double density(double x);

double math\_expect();

double dispersion();

double asymmetry();

double excess();

double modeling();

std::vector<double> generate\_x(int n);

std::vector<std::pair<double,double>> generate\_xy(int n);

void xy\_output(int n);

};

**Модуль emp\_pareto\_class.cpp**

#include "emp\_pareto\_class.h"

//3 конструктора с моделированием случайных величин

Empirical::Empirical(Pareto& std, int n, int \_k)://эмпирическая выборка по стандратному распределению

size(n > 1 ? n : throw 1), k(\_k > 2 ? \_k : (int)log2(size) + 1) {

selection = std.generate\_x\_std(size);

empirical\_density = create\_empirical\_density(delta\_calc(), create\_intervals(delta\_calc()));

}

Empirical::Empirical(mixture& mix, int n, int \_k) ://эмпирическая выборка по смеси распределений

size(n > 1 ? n : throw 1), k(\_k > 2 ? \_k : ((int)log2(size) + 1)) {

selection = mix.generate\_x(n);

empirical\_density = create\_empirical\_density(delta\_calc(), create\_intervals(delta\_calc()));

}

Empirical::Empirical( Empirical& ed, int n, int \_k) ://эмпирическая выборка

size(n), k(\_k > 2 ? k : (int)log2(size) + 1) {

selection = ed.generate\_x(size);

empirical\_density = create\_empirical\_density(delta\_calc(), create\_intervals(delta\_calc()));

}

Empirical::Empirical(Empirical& emp) ://конструктор копирования

size(emp.size > 1 ? emp.size : throw 1), k(emp.k > 2 ? emp.k : ((int)log2(size) + 1)), selection(emp.selection), empirical\_density(emp.empirical\_density) {

empirical\_density = create\_empirical\_density(delta\_calc(), create\_intervals(delta\_calc()));

}

Empirical::Empirical(std::vector<double>& selection) ://для абсолютно случайной любой выборки

size(selection.size()), k((int)log2(size) + 1) {

this->selection = selection;

empirical\_density = create\_empirical\_density(delta\_calc(), create\_intervals(delta\_calc()));

}

Empirical& Empirical::operator = (const Empirical& emp) { //оператор присваивания

if (this == &emp) {

return \*this;

}

selection = emp.selection;

empirical\_density = emp.empirical\_density;

size = emp.size;

k = emp.k;

return \*this;

}

Empirical::Empirical(std::string file\_name) {//конструктор чтением из файла

std::ifstream file;

file.open(file\_name);

if (!file.is\_open()) {

throw std::runtime\_error("Невозможно найти файл");

}

double x;

while (file >> x) {

selection.push\_back(x);

}

size = selection.size();

k = (int)log2(size) + 1;

empirical\_density = create\_empirical\_density(delta\_calc(), create\_intervals(delta\_calc()));

}

//деструктор

Empirical::~Empirical() {

selection.clear();

empirical\_density.clear();

}

//сеттер кол-ва интервалов

void Empirical::SetK( int k) {

if (k >= 2) {

this->k = k;

}

else {

this->k = (int)log2(size) + 1;

}

empirical\_density = create\_empirical\_density(delta\_calc(), create\_intervals(delta\_calc()));

}

//рандомайзер

double Empirical::randomizer(){

double r;

do r = (double)rand() / RAND\_MAX; while (r == 0 || r == 1);

return r;

}

//геттеры

std::vector<double> Empirical::GetSelection(){

return selection;

}

std::vector<double> Empirical::GetEmpDensity(){

return empirical\_density;

}

int Empirical::GetSize(){

return size;

}

int Empirical::GetIntervalsNumber(){

return k;

}

//вычисления

double Empirical::delta\_calc(){

return (selection[size - 1] - selection[0]) / (k);

}

std::vector<double> Empirical::create\_intervals(const double delta){

std::vector<double> intervals;

double slider = selection[0];

for (int i = 0; i < k + 1; ++i) {

intervals.push\_back(slider);

slider += delta;

}

return intervals;

}

std::vector<double> Empirical::create\_empirical\_density(const double delta, std::vector<double> intervals){

std::vector<double> result;

int j = 0;

for (int i = 0; i < intervals.size() - 1; ++i) {

int count = 0;

for (j; j < size; ++j) {

if (i + 1 == intervals.size() - 1) {

if (selection[j] <= intervals[i + 1]) {

++count;

}

else {

break;

}

}

else {

if (selection[j] < intervals[i + 1]) {

++count;

}

else {

break;

}

}

}

result.push\_back(count / (size \* delta));

}

return result;

}

double Empirical::density(double x) {//плотность в точке

double slider = selection[0];

if (x < slider) {

return 0;

}

for (int i = 0; i < k; ++i) {

slider += delta\_calc();

if (i == k - 1) {

if (x <= slider) {

return empirical\_density[i];

}

}

else {

if (x < slider) {

return empirical\_density[i];

}

}

}

return 0;

}

double Empirical::math\_expect(){

double sum = 0;

for (int i = 0; i < size; ++i) {

sum += selection[i];

}

return sum / size;

}

double Empirical::dispersion(){

double sum = 0;

double exp\_val = math\_expect();

for (int i = 0; i < size; ++i) {

sum += pow(selection[i] - exp\_val, 2);

}

return sum / size;

}

double Empirical::asymmetry(){

double sum = 0;

double exp\_val = math\_expect();

for (int i = 0; i < size; ++i) {

sum += pow(selection[i] - exp\_val, 3);

}

return sum / (size \* pow(dispersion(), 3 / 2));

}

double Empirical::excess(){

double sum = 0;

double exp\_val = math\_expect();

for (int i = 0; i < size; ++i) {

sum += pow(selection[i] - exp\_val, 4);

}

return (sum / (size \* pow(dispersion(), 2))) - 3;

}

double Empirical::qumulative\_probability(int i) {//кумулятивка

double q = 0;

for (int j = 0; j <= i; ++j) {

q += empirical\_density[j];

}

return q;

}

double Empirical::top\_bound(){

double sum = 0;

for (int i = 0; i < k; ++i) {

sum += empirical\_density[i];

}

return sum;

}

double Empirical::rand\_var(){//моделирование случайной величины

double r;

do r = (double)rand() / RAND\_MAX \* top\_bound();

while (r == 0 || r == top\_bound());

for (int i = 0; i < k - 1; ++i) {

if (r > qumulative\_probability(i) and r < qumulative\_probability(i + 1)) {

do r = (double)rand() / RAND\_MAX \* ((selection[0] + delta\_calc() \* (i + 1)) - (selection[0] + delta\_calc() \* i)) + (selection[0] + delta\_calc() \* (i + 1));

while (r == (selection[0] + delta\_calc() \* i) || r == (selection[0] + delta\_calc() \* (i + 1)));

break;

}

}

return r;

}

std::vector<double> Empirical::generate\_x(int n) {//генерация выборки x

std::vector<double> result;

for (int i = 0; i < n; ++i) {

result.push\_back(rand\_var());

}

sort(result.begin(), result.end());

return result;

}

std::vector<std::pair<double, double>> Empirical::generate\_xy(){//генерация выборки xy

std::vector <std::pair<double, double >> result;

for (int j = 0; j < size; ++j) {

result.push\_back(std::make\_pair(selection[j], density(selection[j])));

}

return result;

}

///

void Empirical::emp\_output(std::ofstream& file){//вывод в файл

auto pairs = generate\_xy();

file.open("emp\_output.txt");

for (int i = 0; i < size; ++i) {

file << pairs[i].first << "\t" << pairs[i].second << std::endl;

}

file.close();

}

**Модуль emp\_pareto\_class.h**

#pragma once

#include <iostream>

#include <cmath>

#include <vector>

#include <algorithm>

#include <fstream>

#include "mixture\_pareto\_class.h"

#include "std\_pareto\_class.h"

class Empirical {

private:

std::vector<double> selection;

std::vector<double> empirical\_density;

int size = 0;

int k = 0;

double delta\_calc();

std::vector<double> create\_intervals(const double delta);

std::vector<double> create\_empirical\_density(const double delta, std::vector<double> intervals);

double randomizer();

std::vector<double> generate\_x(int n);

double qumulative\_probability(int i);

double top\_bound();

public:

Empirical(Pareto& std, int n, int \_k);

Empirical(mixture& mix, int n, int \_k);

Empirical(Empirical& ed, int n, int \_k);

Empirical(Empirical& emp);

Empirical(std::vector<double>& selection);

Empirical(std::string file);

~Empirical();

Empirical& operator = (const Empirical& emp);

void SetK(int k);

std::vector<double> GetSelection();

std::vector<double> GetEmpDensity();

int GetSize();

int GetIntervalsNumber();

double density(double x);

double math\_expect();

double dispersion();

double asymmetry();

double excess();

double rand\_var();

std::vector<std::pair<double, double>> generate\_xy();

void emp\_output(std::ofstream& file);

};

**Модуль test.cpp**

#include "catch.hpp"

#include "std\_pareto\_class.h"

#include "mixture\_pareto\_class.h"

#include "emp\_pareto\_class.h"

bool equal(const double& x, const double& y) {

if (abs(x - y) <= 0.005) {

return true;

}

else return false;

}

TEST\_CASE("Standart pareto distribution") {

Pareto pareto(3.3,0,1);

CHECK(equal(pareto.dispersion(),1)==true);

CHECK(equal(pareto.excess\_koef(), 0.05) == true);

CHECK(equal(pareto.central\_interval(), 0.999) == true);

CHECK(equal(pareto.pareto\_std\_distribution(0), 0.399) == true);

CHECK(equal(pareto.math\_expect(), 0) == true);

CHECK(equal(pareto.asymmetry(), 0) == true);

}

TEST\_CASE("Shift-scale transformation") {

Pareto pareto(3.3, 2, 2);

CHECK(equal(pareto.dispersion(), 4.01) == true);

CHECK(equal(abs(pareto.excess\_koef()), 2.809) == true);

CHECK(equal(pareto.central\_interval(), 0.999) == true);

CHECK(equal(pareto.pareto\_std\_distribution(0), 0.12) == true);

CHECK(equal(pareto.math\_expect(), 2) == true);

CHECK(equal(pareto.asymmetry(), 0) == true);

}

TEST\_CASE("Trivial mixture case") {

Pareto std1(3.3, 2, 2);

Pareto std2(3.3, -2, 2);

mixture mix(std1, std2, 0.5);

CHECK(equal(mix.math\_expect(), 0) == true);

CHECK(equal(mix.dispersion(), 8.01) == true);

CHECK(equal(mix.asymmetry(), 0) == true);

CHECK(equal(mix.excess(), 0.998) == true);

}

TEST\_CASE("Math expect case") {

Pareto std1(2.3,2,1);

Pareto std2(3.3,-2,2);

mixture mix(std1, std2, 0.5);

CHECK(equal(mix.math\_expect(), 0) == true);

CHECK(equal(mix.dispersion(), 6.58) == true);

CHECK(equal(mix.asymmetry(), -1.31) == true);

CHECK(equal(mix.excess(), 1.327) == true);

}

TEST\_CASE("Dispersion test") {

Pareto std1(3.3, 0, 1);

Pareto std2(3.3, 0, 3);

mixture mix(std1, std2, 0.5);

CHECK(equal(mix.math\_expect(), 0) == true);

CHECK(equal(mix.dispersion(), 5.015) == true);

CHECK(equal(mix.asymmetry(), 0) == true);

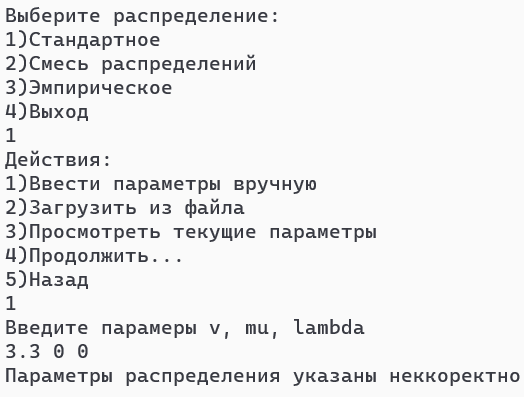
CHECK(equal(mix.excess(), -4.917) == true);

}

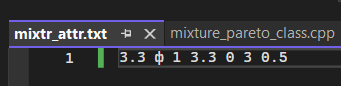
**Тесты программы**

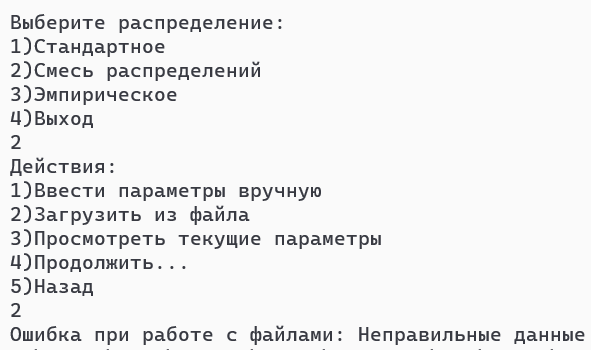
1.Обработка исключений

1.1. Некорректные входные данные

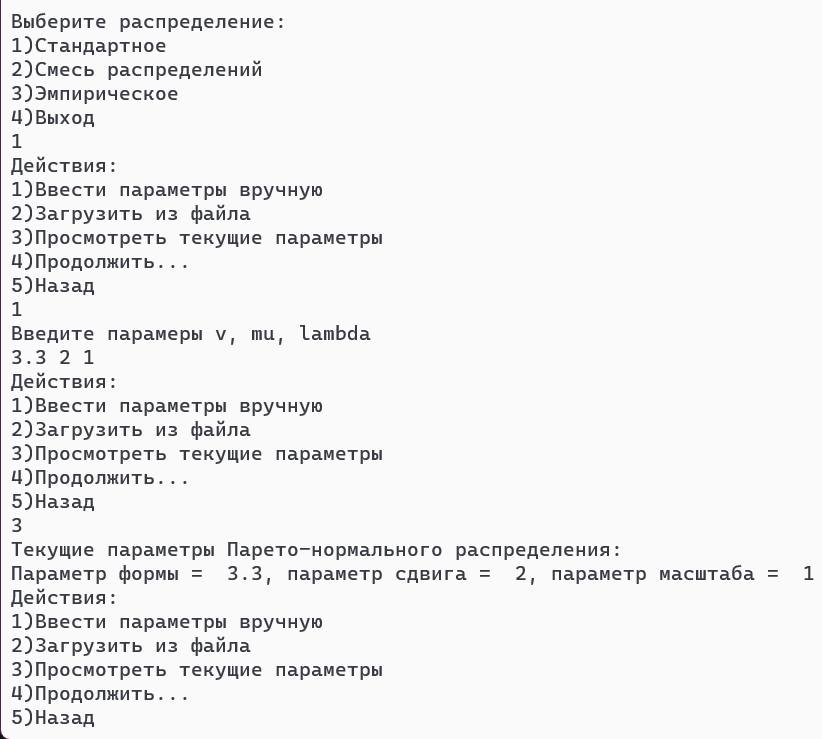


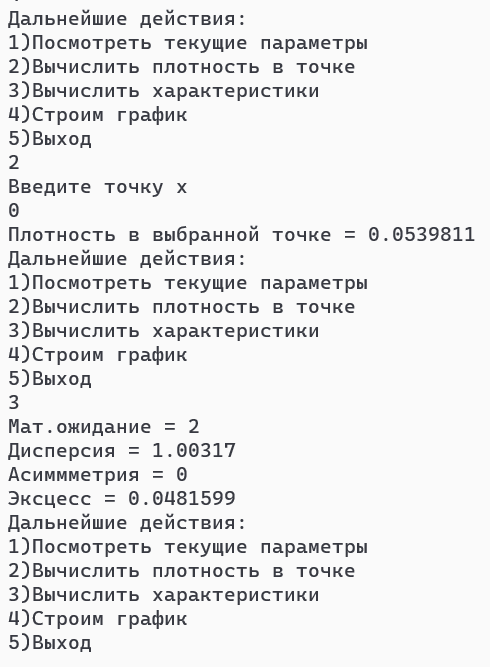
1.2. Ошибка при работе с файлом



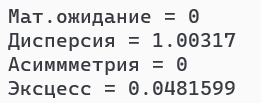


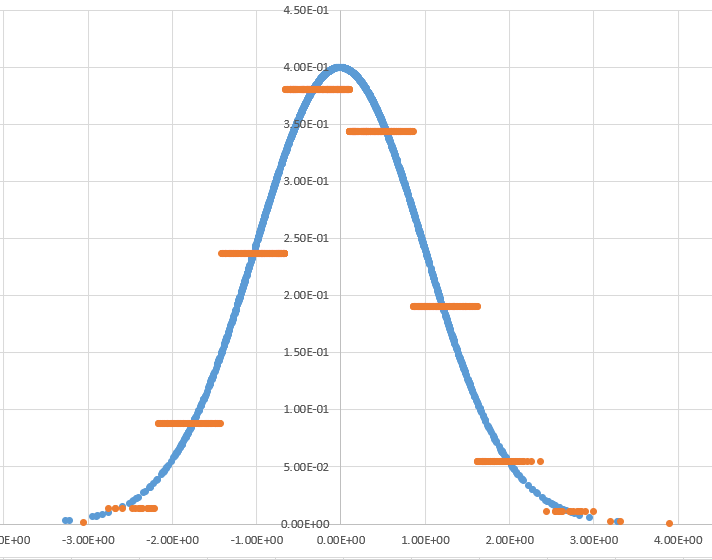
2.Работоспособность меню и программы



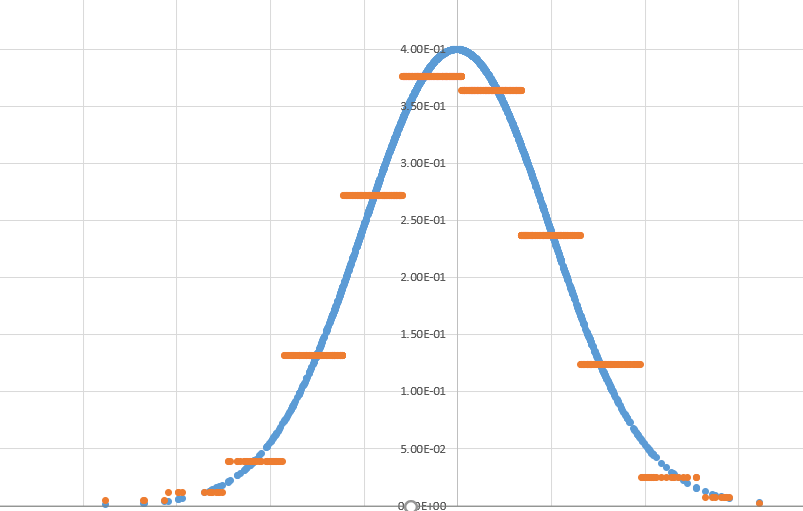


3. Тест для стандартного распределения(v=3.3, mu=0, lambda=1, n =2000)





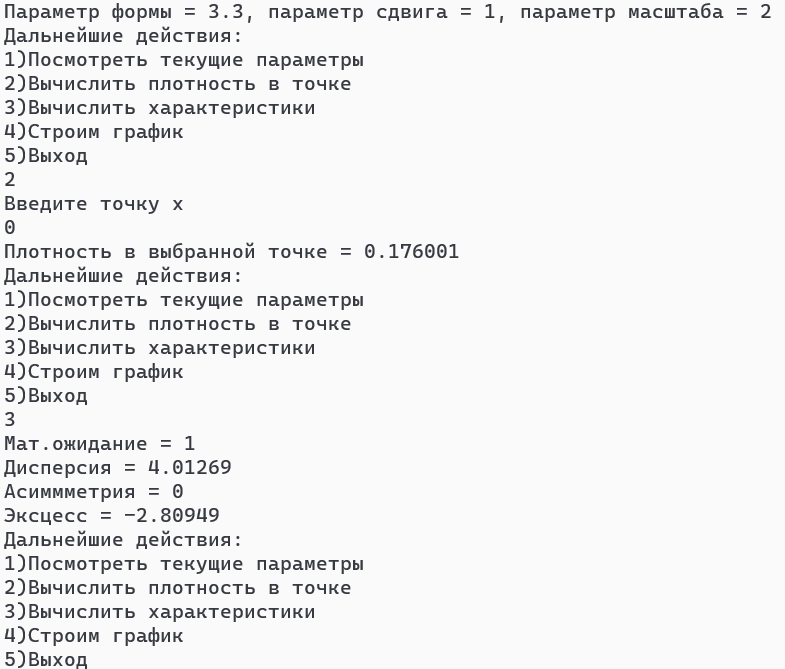
Лабораторная работа №3

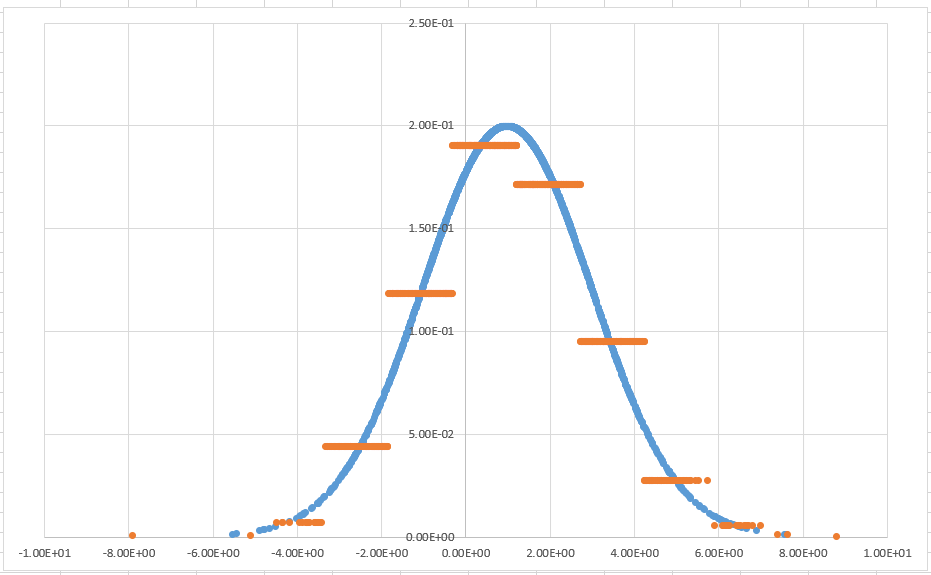


Лабораторная работа №1

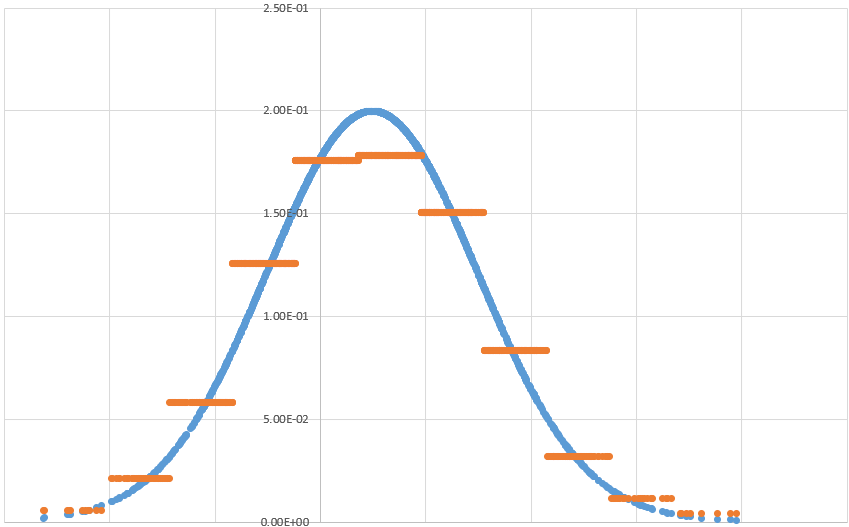
Как мы видим, вычисления выполнены корректно, а сравнение графиков показало, что и моделирование случайной величины верно.

4.Тест сдвиг-масштабных преобразований(v=3.3, mu=1, lambda=2, n=2000)





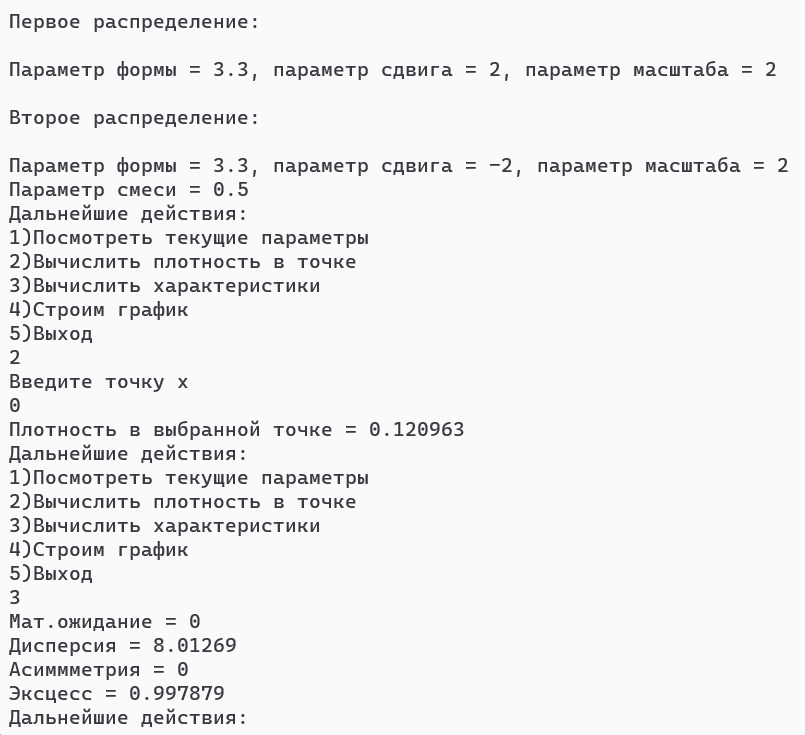
Лабораторная работа №3

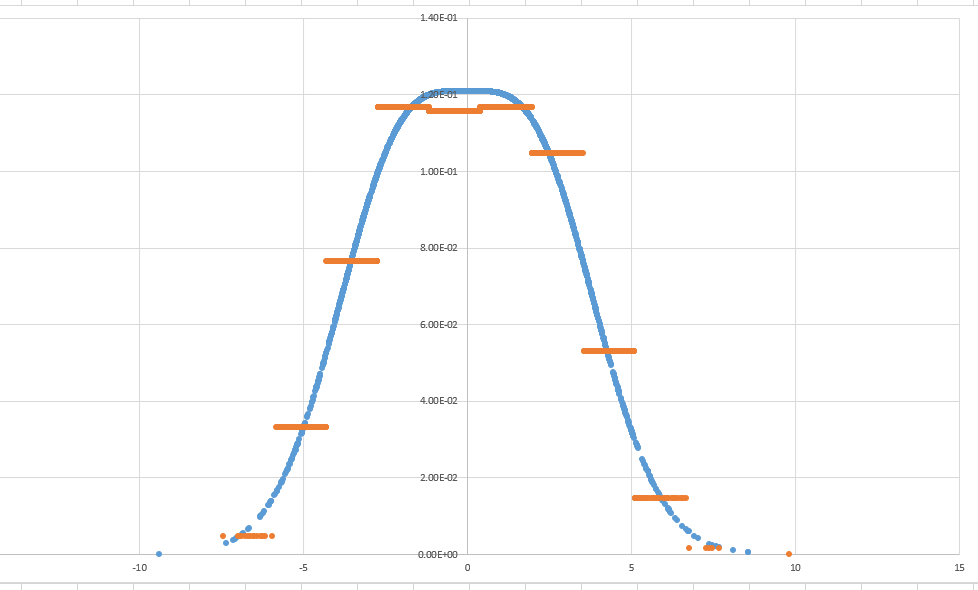


Лабораторная работа №1

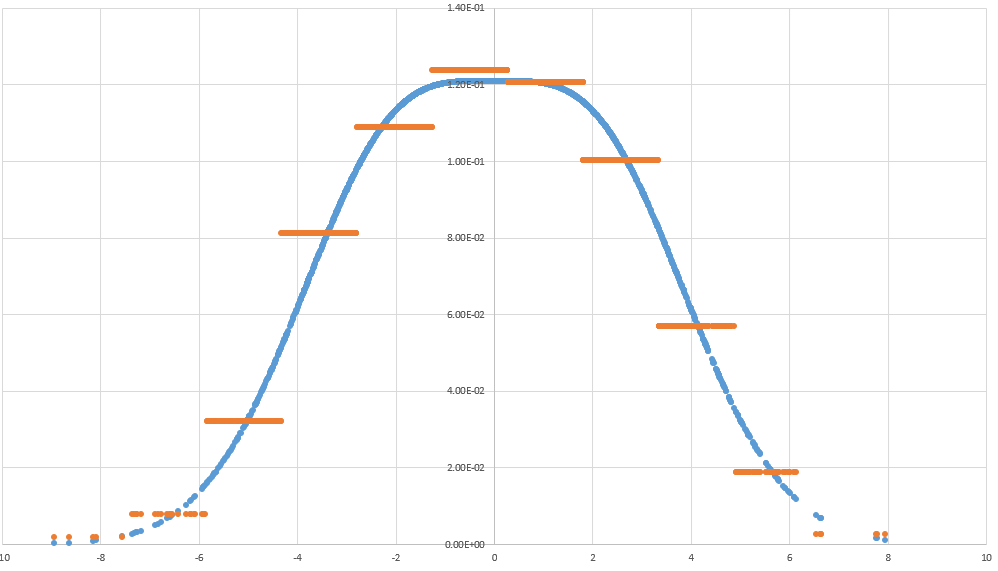
Опять наблюдаем соответствие всех характеристик, а также графиков, полученных ранее.

5.Тест для тривиального случая смеси распределения





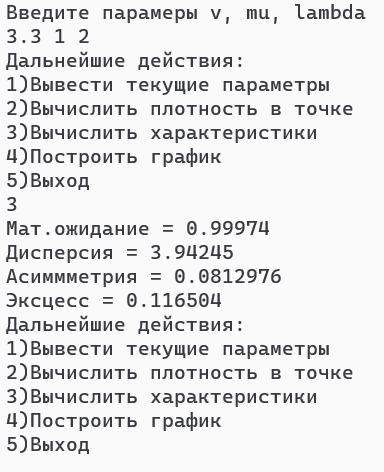
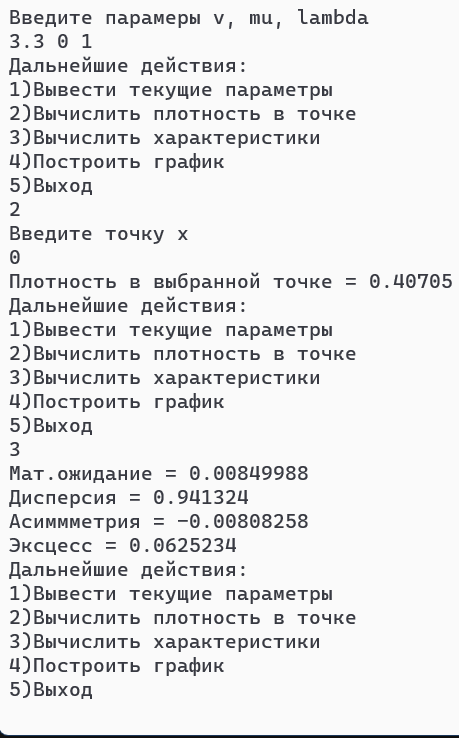
Лабораторная работа №3



Лабораторная работа №1

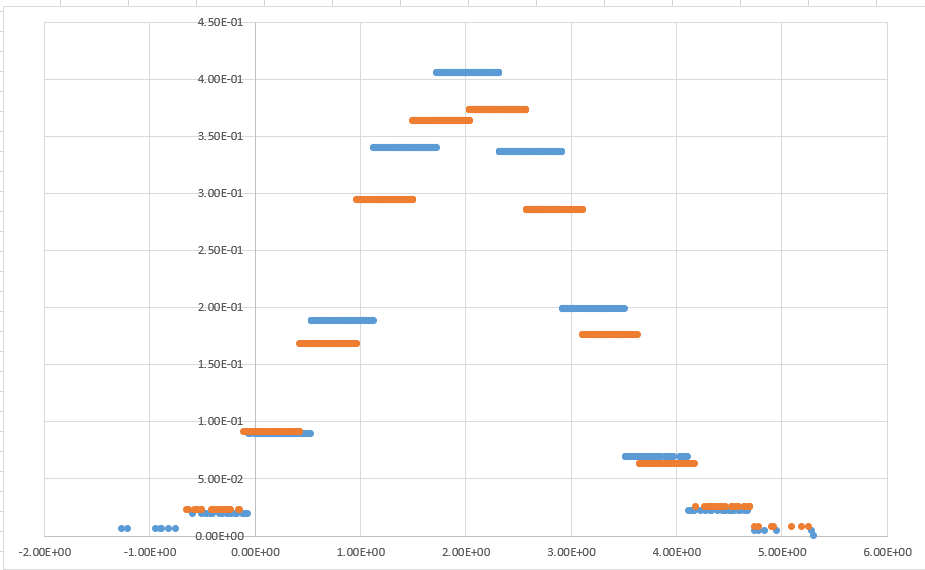
Все полученные данные опять совпадают.

6.Проверка эмпирического распределения



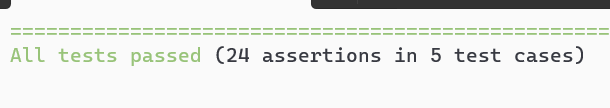
Значения совпадают с полученными ранее теоретическими, что говорит о корректности работы эмпирического распределения.

7.Формирование эмпирического распределения по эмпирическому распределению



Полученные графики, расположены близко друг к другу, что и ожидалось.

8.Проверка юнит-тестов



**Вывод по работе**

Используя наработки, первой лабораторной работы был реализован создали классы методы, которых моделируют характеристики смеси распределений и эмпирического распределения. Была обеспечена персистентность объекта средствами сохранения и считывания атрибутов класса с файла, а также реализованы обработки исключений для разных ошибочных ситуаций, которые могут возникнуть при работе с классом.