Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Новосибирский государственный технический университет»

Кафедра теоретической и прикладной информатики

Отчёт

по практическому заданию  
«Устойчивые методы оценивания параметров статистических моделей»

курса «Современные проблемы прикладной математики и

наукоемкого программного обеспечения»

Вариант №2.а

Выполнил:

студент ФПМИ,

гр. ПММ-01

Кастин В. С.

Проверил:

Профессор каф. ТПИ

Лисицин Д. В.

Новосибирск 2020

**Цель работы.** Изучить методы робастного оценивания параметра сдвига распределений случайных величин.

**Порядок выполнения.**

1. Разработать программу, которая реализует:

– генерацию наборов данных с заданным в варианте чистым распределением и засоренным распределением, использовать засоряющие распределения, совпадающие с чистым с точностью до значений параметров сдвига и масштаба;

– вычисление выборочных характеристик: среднего арифметического, медианы, дисперсии, коэффициентов асимметрии и эксцесса.

2. Провести проверку генератора чистого распределения путем сравнения выборочных характеристик с их теоретическими значениями на выборках большого объема (как альтернативу можно использовать какие-либо критерии согласия, например, хи-квадрат, в том числе с использованием стороннего программного обеспечения);

3. Для выборок с разными видами распределений вычислить оценки параметра сдвига в виде среднего арифметического и выборочной медианы. Использовать выборки, имеющие следующие виды распределений:

– чистое распределение;

– засоренное распределение с симметричным засорением (равные сдвиги у чистого и засоряющего распределений, масштаб у засоряющего – больше);

– засоренное распределение с асимметричным засорением (неравные сдвиги у чистого и засоряющего распределений).

Сравнить устойчивость оценок для распределений разных видов (минимум по две выборки для каждого вида).

Для работы будет использовано двустороннее экспоненциальное распределение с параметром формы v = 1.5.

1. Разработать программу, которая реализует:

– генерацию наборов данных с заданным в варианте чистым распределением и засоренным распределением, использовать засоряющие распределения, совпадающие с чистым с точностью до значений параметров сдвига и масштаба;

– вычисление выборочных характеристик: среднего арифметического, медианы, дисперсии, коэффициентов асимметрии и эксцесса.

Код программы приведён в приложении.

2. Провести проверку генератора чистого распределения путем сравнения выборочных характеристик с их теоретическими значениями на выборках большого объема (как альтернативу можно использовать какие-либо критерии согласия, например, хи-квадрат, в том числе с использованием стороннего программного обеспечения);

Проверка генератора чистого распределения будет проведена с помощью таких критериев согласия как критерий хи-квадрат Пирсона и критерий Колмогорова, реализованных в программе ISW, N = 16600:



Критерий хи-квадрат Пирсона показывает, что эмпирическое распределение близко к теоритическому.

3. Для выборок с разными видами распределений вычислить оценки параметра сдвига в виде среднего арифметического и выборочной медианы. Использовать выборки, имеющие следующие виды распределений:

– чистое распределение;

– засоренное распределение с симметричным засорением (равные сдвиги у чистого и засоряющего распределений, масштаб у засоряющего – больше);

– засоренное распределение с асимметричным засорением (неравные сдвиги у чистого и засоряющего распределений).

Сравнить устойчивость оценок для распределений разных видов (минимум по две выборки для каждого вида).

Выборки №1

Чистое распределение (сдвиг 0, масштаб 1):

Параметр сдвига в виде среднего арифметического: -0.008;

Параметр сдвига в виде выборочной медианы: -0.003.

Засорённое распределение с симметричным засорением (сдвиг 0, масштаб 1.5):

Параметр сдвига в виде среднего арифметического: 0.004;

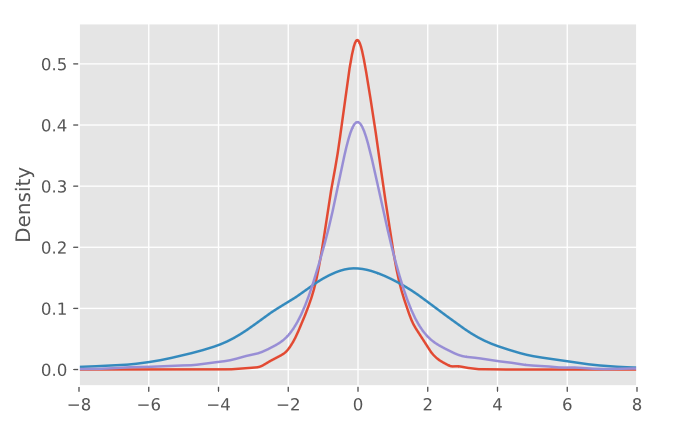
Параметр сдвига в виде выборочной медианы: 0.004.

Засорённое распределение с асимметричным засорением (сдвиг 1, масштаб 2):

Параметр сдвига в виде среднего арифметического: 0.471;

Параметр сдвига в виде выборочной медианы: 0.007.

График функций плотности для чистой, засоряющей и засоренной плотности в случае симметричного засорения:

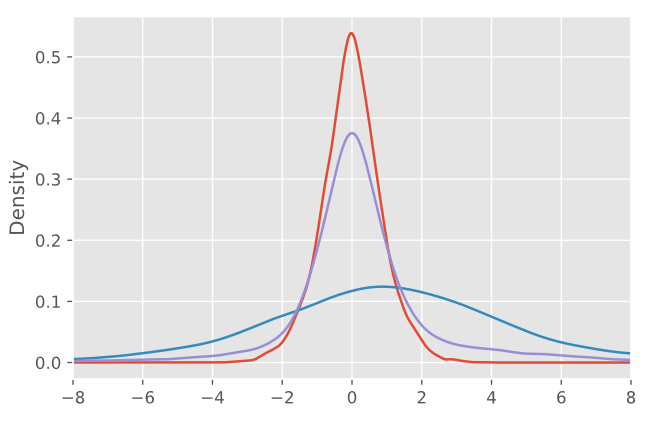


1. Чистое распределение

2. Засорённое распределение

3. Засоряющее распределение

График функций плотности для чистой, засоряющей и засоренной плотности в случае асимметричного засорения:



1. Чистое распределение

2. Засорённое распределение

3. Засоряющее распределение

График функции влияния оценок чистого распределения:



Выборки №2

Чистое распределение:

Параметр сдвига в виде среднего арифметического: -0.001;

Параметр сдвига в виде выборочной медианы: 0.003.

Засорённое распределение с симметричным засорением (сдвиг 0, масштаб 1.5):

Параметр сдвига в виде среднего арифметического: 0.001;

Параметр сдвига в виде выборочной медианы: 0.007.

Засорённое распределение с асимметричным засорением (сдвиг 1, масштаб 2):

Параметр сдвига в виде среднего арифметического: 0.456;

Параметр сдвига в виде выборочной медианы: -0.01.

**Вывод**

В данной работе были изучены методы робастного оценивания параметра сдвига распределений случайных величин.

Для этого была разработана программа, реализующая генерацию наборов данных с двусторонним экспоненциальным распределением и вычисляющая выборочные характеристики.

Была проведена проверка генератора чистого распределения путём проверки на согласии с помощью критериев хи-квадрат Пирсона и критерия Колмогорова, где критерий хи-квадрат Пирсона показал, что эмпирическое распределение близко к теоритическому.

Также были вычислены оценки параметра сдвига в виде среднего арифметического и выборочной медианы для выборок с разными видами распределений. В ходе сравнения оценок разных видов распределений видно, что значения оценок медианы довольно близки, а вот в случае оценки среднего ситуация обратная, что говорит об устойчивости медианы и неустойчивости среднего.

**Приложение**

#include <random>

#include <iostream>

#include <cmath>

#include <random>

#include <chrono>

#include <fstream>

#include <algorithm>

#include <vector>

**using** **namespace** std;

//Класс для генерации случайных чисел

**class** **RandomGenerator**

{

**public:**

**static** std::mt19937 & getMt19937();

**private:**

RandomGenerator();

~RandomGenerator() {}

**static** RandomGenerator& instance();

RandomGenerator(RandomGenerator **const**&) = **delete**;

RandomGenerator& **operator**= (RandomGenerator **const**&) = **delete**;

std::mt19937 mMt;

};

RandomGenerator::RandomGenerator() {

std::random\_device rd;

**if** (rd.entropy() != **0**) {

std::seed\_seq seed{ rd(), rd(), rd(), rd(), rd(), rd(), rd(), rd() };

mMt.seed(seed);

}

**else** {

**auto** seed = std::chrono::high\_resolution\_clock::now().time\_since\_epoch().count();

mMt.seed(seed);

}

}

RandomGenerator& RandomGenerator::instance() {

**static** RandomGenerator s;

**return** s;

}

std::mt19937 & RandomGenerator::getMt19937() {

**return** RandomGenerator::instance().mMt;

}

//Среднее арифмитическое

**double** arithmeticMean(**double** \*array, **int** length)

{

**double** sum = **0**;

**for** (**int** i = **0**; i < length; i++)

{

sum += array[i];

}

**return** (sum / length);

}

**double** D(**double** \*array, **double** mean, **int** length)

{

**double** sum = **0**;

**for** (**int** i = **0**; i < length; i++)

{

sum += pow((array[i] - mean), **2**);

}

**return** (sum / length);

}

**double** asymmetry(**double** \*array, **double** mean, **double** D, **int** length)

{

**double** sum = **0**;

**for** (**int** i = **0**; i < length; i++)

{

sum += pow((array[i] - mean), **3**);

}

**return** (sum / (length \* pow(D, **3**/**2**)));

}

**double** kurtosis(**double** \*array, **double** mean, **double** D, **int** length)

{

**double** sum = **0**;

**for** (**int** i = **0**; i < length; i++)

{

sum += pow((array[i] - mean), **4**);

}

**return** (sum / (length \* pow(D, **2**)));

}

**int** main()

{

**int** N;

cout << "Enter N: ";

cin >> N;

cout << endl;

**const** **double** v = **1.5**;

**const** **double** e = **2.718**;

**double** b = pow((**2**/(e \* v)), **1**/v);

**const** **double** ep = **0.3**;

**double** m = **1.5**;

**double** s = **0**;

//Генерация псевдослучайной величины u

std::mt19937 &mt = RandomGenerator::getMt19937();

std::uniform\_real\_distribution<**double**> dist(**0.0**, **1.0**);

**double** u[N] = {};

**int** i = **0**;

**size\_t** j = **0**;

**while** (i < N)

{

u[i] = dist(mt);

//cout << u[i] << "\n";

i++;

j++;

}

std::cout << "**\n**";

//Генерация псевдослучайной величины r

std::mt19937 &mt2 = RandomGenerator::getMt19937();

std::uniform\_real\_distribution<**double**> dist2(**0.0**, **1.0**);

**double** r[N] = {};

i = **0**;

j = **0**;

**while** (i < N)

{

r[i] = dist2(mt2);

//cout << r[i] << "\n";

i++;

j++;

}

//Генерация псевдослучайной величины r1

std::mt19937 &mt3 = RandomGenerator::getMt19937();

std::uniform\_real\_distribution<**double**> dist3(**0.0**, **1.0**);

**double** r1[N] = {};

i = **0**;

j = **0**;

**while** (i < N)

{

r1[i] = dist3(mt3);

//cout << r[i] << "\n";

i++;

j++;

}

//Моделирование случайной величины x, имеющей стандартное чистое распределение

i = **0**;

**double** x[N] = {};

**while** (i < N)

{

x[i] = (b\*(**2**\*r[i] - **1**))/u[i];

//Если условие выполняется, то генерация i-го члена массива (выборки) происходит заново

**if**(pow(u[i], **2**) > exp(-(pow(abs(x[i]), v))))

{

u[i] = dist(mt);

r[i] = dist(mt2);

**continue**;

}

//cout << x[i] << endl;

i++;

}

i = **0**;

**while** (i < N)

{

u[i] = dist(mt);

//cout << u[i] << "\n";

i++;

j++;

}

i = **0**;

**while** (i < N)

{

r[i] = dist2(mt2);

//cout << r[i] << "\n";

i++;

j++;

}

ofstream clearY("ClearY.dat");

clearY << "ClearY**\n**0 " << N << "**\n**";

ofstream **dirtyY**("DirtyY.dat");

dirtyY << "DirtyY**\n**0 " << N << "**\n**";

//Моделирование случайной величины y, имеющей стандартное симметричное засорённое распределение

i = **0**;

**double** y[N] = {};

**while** (i < N)

{

y[i] = (b\*(**2**\*r[i] - **1**))/u[i];

//Моделирование чистых значений по тому же принципу, что и в цикле до этого

**if**(pow(u[i], **2**) > exp(-(pow(abs(y[i]), v))))

{

u[i] = dist(mt);

r[i] = dist2(mt2);

**continue**;

}

//Моделирование засорённых значений, на которые влияют параметры масштаба и сдвига, если условие выполняется

**if**(r1[i] > (**1** - ep))

{

y[i] = s + m\*y[i];

dirtyY << y[i] << endl;

}

**else** clearY << y[i] << endl;

//cout << y[i] << endl;

i++;

}

i = **0**;

**while** (i < N)

{

u[i] = dist(mt);

//cout << u[i] << "\n";

i++;

j++;

}

i = **0**;

**while** (i < N)

{

r[i] = dist2(mt2);

//cout << r[i] << "\n";

i++;

j++;

}

i = **0**;

**while** (i < N)

{

r1[i] = dist3(mt3);

//cout << u[i] << "\n";

i++;

j++;

}

ofstream clearZ("ClearZ.dat");

clearZ << "ClearZ**\n**0 " << N << "**\n**";

ofstream **dirtyZ**("DirtyZ.dat");

dirtyZ << "DirtyZ**\n**0 " << N << "**\n**";

//Моделирование случайной величины y, имеющей стандартное асимметричное засорённое распределение

m = **2**;

s = **1**;

i = **0**;

**double** z[N] = {};

**while** (i < N)

{

z[i] = (b\*(**2**\*r[i] - **1**))/u[i];

//Моделирование чистых значений по тому же принципу, что и в цикле до этого

**if**(pow(u[i], **2**) > exp(-(pow(abs(z[i]), v))))

{

u[i] = dist(mt);

r[i] = dist2(mt2);

**continue**;

}

//Моделирование засорённых значений, на которые влияют параметры масштаба и сдвига, если условие выполняется

**if**(r[i] > (**1** - ep))

{

z[i] = s + m\*z[i];

dirtyZ << z[i] << endl;

}

**else** clearZ << z[i] << endl;

//cout << z[i] << endl;

i++;

}

//Запись значений чистой выборки

ofstream Rx("ResultsX.dat");

Rx << "ResultsX**\n**0 " << N << "**\n**";

**for** (**int** i = **0**; i < N; i++)

{

Rx << x[i] << endl;

}

//Запись значений засорённой выборки

ofstream Ry("ResultsY.dat");

Ry << "ResultsY**\n**0 " << N << "**\n**";

**for** (**int** i = **0**; i < N; i++)

{

Ry << y[i] << endl;

}

//Запись значений засорённой выборки

ofstream Rz("ResultsZ.dat");

Rz << "ResultsZ**\n**0 " << N << "**\n**";

**for** (**int** i = **0**; i < N; i++)

{

Rz << z[i] << endl;

}

cout << endl;

//Арифмитическое среднее для выборок

cout << "Arithmetic mean for X is: " << arithmeticMean(x, N) << endl;

cout << "Arithmetic mean for Y is: " << arithmeticMean(y, N) << endl;

cout << "Arithmetic mean for Z is: " << arithmeticMean(z, N) << endl;

cout << endl;

//Выборочная дисперсия

cout << "D for X is: " << D(x, arithmeticMean(x, N), N) << endl;

cout << "D for Y is: " << D(y, arithmeticMean(y, N), N) << endl;

cout << "D for Z is: " << D(z, arithmeticMean(z, N), N) << endl;

cout << endl;

//Выборочная асимметрия

cout << "Asymmetry for X is: " << asymmetry(x, arithmeticMean(x, N), D(x, arithmeticMean(x, N), N), N) << endl;

cout << "Asymmetry for Y is: " << asymmetry(y, arithmeticMean(y, N), D(y, arithmeticMean(y, N), N), N) << endl;

cout << "Asymmetry for Z is: " << asymmetry(z, arithmeticMean(z, N), D(z, arithmeticMean(z, N), N), N) << endl;

cout << endl;

//Выборочный эксцесс

cout << "Kurtosis for X is: " << kurtosis(x, arithmeticMean(x, N), D(x, arithmeticMean(x, N), N), N) << endl;

cout << "Kurtosis for Y is: " << kurtosis(y, arithmeticMean(y, N), D(y, arithmeticMean(y, N), N), N) << endl;

cout << "Kurtosis for Z is: " << kurtosis(z, arithmeticMean(z, N), D(z, arithmeticMean(z, N), N), N) << endl;

cout << endl;

//Медиана

sort(x, x+N);

cout << "X median is " << (x[N/**2**] + x[N/**2** - **1**]) / **2** << endl;

sort(y, y+N);

cout << "Y median is " << (y[N/**2**] + y[N/**2** - **1**]) / **2** << endl;

sort(z, z+N);

cout << "Z median is " << (z[N/**2**] + z[N/**2** - **1**]) / **2** << endl;

**return** **0**;

}