МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ НЭТИ | Факультет прикладной математики и информатики

Кафедра прикладной математики

Лабораторная работа

по дисциплине «Современные проблемы прикладной математики и наукоемкого программного»

ВВЕДЕНИЕ В БЕЗОПАСНОСТЬ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Бригада ГРУШЕВ АНДРЕЙ
ТОЛМАЧЁВ ПАВЕЛ
ШУШКАРЕЕВ АРМАН

Группа ПММ-42

Вариант 12

Преподаватели ЛИСИЦИН ДАНИИЛ ВАЛЕРЬЕВИЧ

Новосибирск, 2024

1. Цель работы

Изучить методы устойчивого оценивания параметра сдвига распределений непрерывных случайных величин.

2. Постановка задачи

Вариант 12. Составное обобщенное гауссовское распределение со следующим значением параметра v: в) 0.9 .

Составное обобщенное гауссовское распределение (с дополнительным параметром, равным 7) имеет плотность

$$f(x,v)=\frac{1}{K}\begin{cases}\exp[-|x|^7], |x|\le v\\ \exp[-a(|x|-v)-v^7], |x|>v\end{cases}$$

где $v>0, a=7v^6, K=\frac{2}{7}\left[\gamma\left(\frac{1}{7},v^7\right)+\frac{e^{-v^7}}{v^6}\right]$, $\gamma(c,z)=\int_0^z x^{c-1}e^{-x}dx$ – неполная гамма-функция.

Плотность распределения в центре соответствует усеченному обобщенному гауссовскому распределению с параметром формы 7, а хвосты распределения являются лапласовскими. При неограниченном увеличении параметра v распределение приближается к обобщенному гауссовскому. При приближении параметра формы к 0 распределение будет приближаться к лапласовскому, но дисперсия при этом будет неограниченно возрастать. Данное распределение является наименее благоприятным в классе засоренных обобщенных гауссовских распределений с параметром формы 7 и уровнем засорения $1-\frac{2\Gamma\left(\frac{1}{7}\right)}{7K}$, где Γ - гамма-функция.

Дисперсия определяется формулой

$$\sigma^2=\frac{2}{K}\left[\frac{\gamma\left(\frac{3}{7},v^7\right)}{7}+e^{-v^7}\left(\frac{2}{a^3}+\frac{2v}{a^2}+\frac{v^2}{a}\right)\right],$$

коэффициент эксцесса определяется формулой

$$\gamma_2=\frac{2}{\sigma^4K}\left[\frac{\gamma\left(\frac{5}{7},v^7\right)}{7}+e^{-v^7}\left(\frac{24}{a^5}+\frac{24v}{a^4}+\frac{12v^2}{a^3}+\frac{4v^3}{a^2}+\frac{v^4}{a}\right)\right]-3$$

В таблице приведены значения дисперсии, коэффициента эксцесса, константы K и вероятности попадания в центральный интервал $P=\frac{2}{7K}\gamma\left(\frac{1}{7},v^7\right)$ для некоторых значений параметра формы.

v	0.55	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1	1.1	∞
σ^2	53.5	7.44	3.23	1.58	0.893	0.586	0.444	0.375	0.342	0.320	0.32
γ_2	2.98	2.79	2.48	1.93	1.16	0.35	-0.28	2.33	-0.67	-1.02	-1.04
K	11.3	4.90	3.62	2.88	2.44	2.19	2.04	1.95	1.91	1.88	1.87
P	0.097	0.264	0.383	0.512	0.638	0.749	0.836	0.901	0.945	0.988	1

Для моделирования случайной величины можно использовать следующий алгоритм, включающий в себя применение метода суперпозиции для составной плотности, метода исключения для моделирования центра распределения, метода обратной функции для моделирования правого хвоста и метода функциональных преобразований (зеркальное отражение) для моделирования левого хвоста. Все величины r_i являются реализациями случайной величины, равномерно распределенной на интервале $(0,1)$.

Шаг 1. Получить реализацию r_1. Если $r_1 \leq P$, перейти на шаг 2 , иначе перейти на шаг 4.

Шаг 2. Получить реализацию z нормальной случайной величины (способ ее моделирования см. в пп. 1), вычислить $x_1 = 7^{-\frac{1}{7}}z$.

Шаг 3. Получить реализацию r_2. Если $|x_1| \leq v$ и

$$\ln(r_2) \leq -|x_1|^7 + \frac{7^{\frac{2}{7}}x_1^2}{2} - \frac{5}{14},$$

то x_1 - реализация целевой случайной величины, иначе перейти на шаг 2 .

Шаг 4. Получить реализацию r_3, вычислить $x_2 = v - \frac{\ln(r_3)}{a}$.

Шаг 5. Если $r_1 < \frac{1+P}{2}$, то x_2 − реализация целевой случайной величины, иначе реализацией целевой случайной величины является $-x_2$.

3. Результаты работы

Проверка генератора распределения

- Сравнение выборочных характеристик с их теоретическими значениями на выборке большого объема (N порядка $10^4 - 10^7$)

Чистое распределение $\boldsymbol{\theta = 0},\ \boldsymbol{\lambda = 1}$

	$N_1 = 3 \cdot 10^4$	$N_2 = 3 \cdot 10^5$	$N_3 = 3 \cdot 10^6$	Теорет.
Среднее арифметическое	4.735207e-04	-3.796111e-04	-4.762013e-04	0
Выборочная медиана	7.442362e-03	-1.001556e-04	-2.779833e-04	0
Дисперсия	4.432014e-01	4.431132e-01	4.438013e-01	0.444
Коэффициент асимметрии	-1.016549e-02	-1.714923e-03	-2.448224e-04	0
Коэффициент эксцесса	-3.065138e-01	-2.994894e-01	-2.837722e-01	-0.28

Симметричное засорение $\boldsymbol{\varepsilon = 0.3, \theta = 0},\ \boldsymbol{\lambda = 3}$

$$\boldsymbol{g(x, \theta, \lambda) = (1 - \varepsilon) f\left(\frac{x - 0}{1}\right) + \varepsilon h\left(\frac{x - 0}{3}\right)}$$

	$N_1 = 3 \cdot 10^4$	$N_2 = 3 \cdot 10^5$	$N_3 = 3 \cdot 10^6$	Теорет.
Среднее арифметическое	1.313852e-03	-7.352431e-04	-7.876373e-05	0
Выборочная медиана	-2.125672e-03	2.830673e-05	-3.498440e-04	0
Дисперсия	3.250191e-01	3.246742e-01	3.252202e-01	0.33
Коэффициент асимметрии	-5.033742e-02	-1.133862e-03	3.221228e-05	0
Коэффициент эксцесса	5.447048e-01	5.442455e-01	5.533183e-01	0.55

Асимметричное засорение $\varepsilon = 0.3, \theta = 1.5,\ \lambda = 1.5$

$$g(x, \theta, \lambda) = 0.7 * f\left(\frac{x+0}{1}\right) + 0.3 * h\left(\frac{x+1.5}{1.5}\right)$$

	$N_1 = 3 \cdot 10^4$	$N_2 = 3 \cdot 10^5$	$N_3 = 3 \cdot 10^6$	Теорет.
Среднее арифметическое	3.630326e-01	3.608726e-01	3.599402e-01	0,36
Выборочная медиана	4.007729e-01	3.914857e-01	3.904017e-01	0,39
Дисперсия	6.942449e-01	6.985881e-01	6.981759e-01	0.70
Коэффициент асимметрии	2.997128e-02	3.978502e-02	3.988752e-02	0.04
Коэффициент эксцесса	-5.120802e-01	-5.266748e-01	-5.334398e-01	-0.53

Оценки параметра сдвига

Минимум по три выборки с одинаковыми значениями всех параметров

$$N:\ 100 - 500, \qquad \varepsilon:\ 0.1 - 0.4$$

При выборе параметров засорения ориентироваться на график чистой, засоряющей и засоренной плотностей.

Сравнить устойчивость оценок для распределений указанных видов по их отклонению от истинного значения. Сопоставить результаты сравнения со свойствами функций влияния оценок.

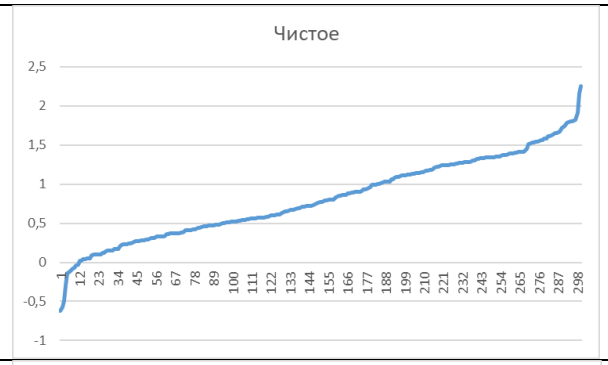
Для всех выборок: $N: 300,\ \varepsilon:\ 0.30$

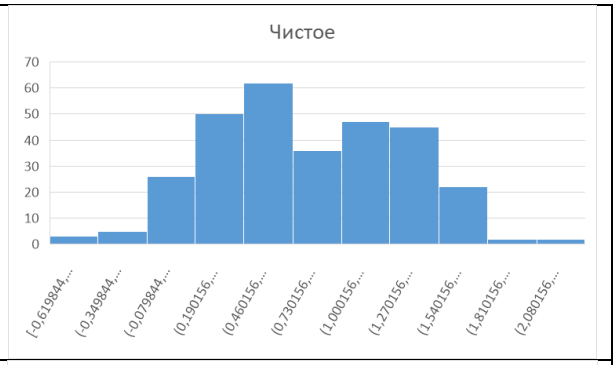
- **Для чистого распределения**

$$\theta = 1,\ \lambda = 1.25$$

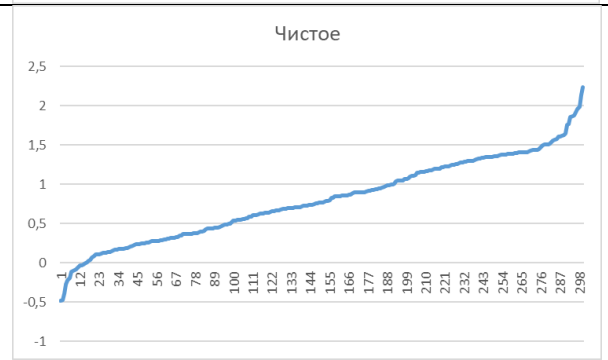
График (значения наблюдений как функции от номеров наблюдений).

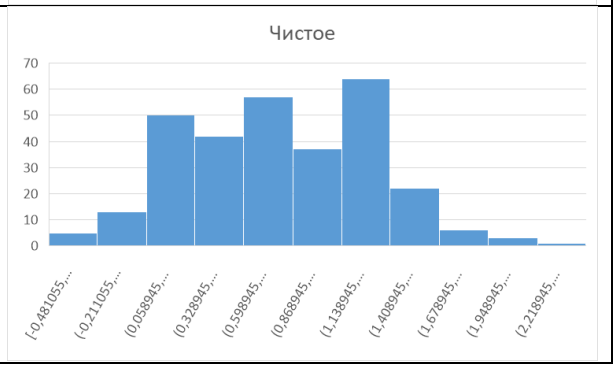
1



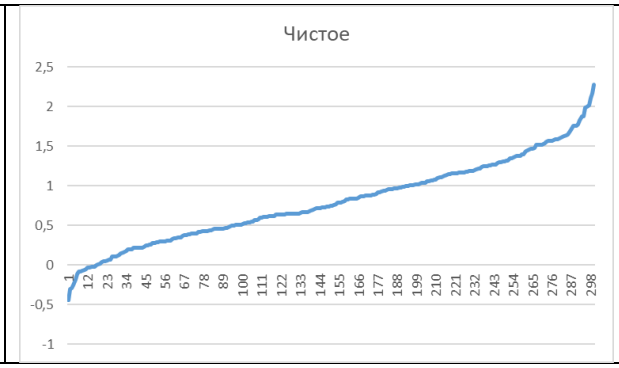


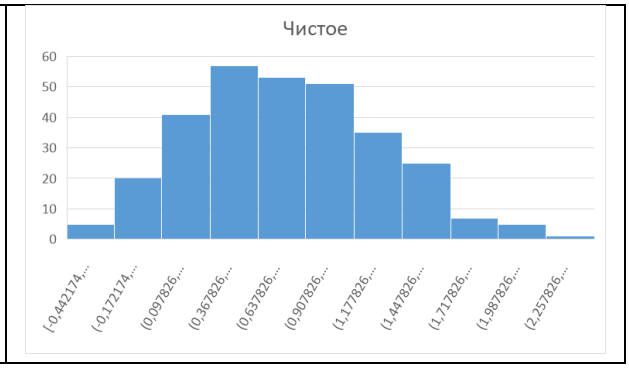
2





3



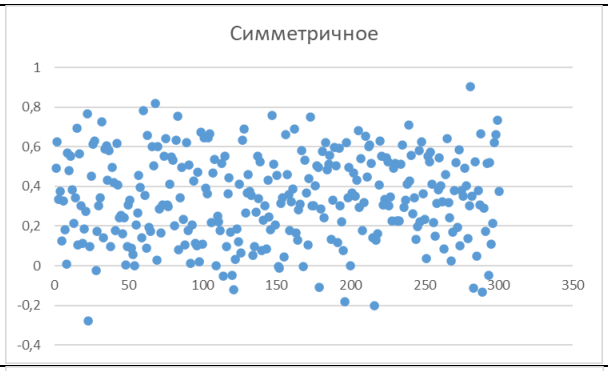
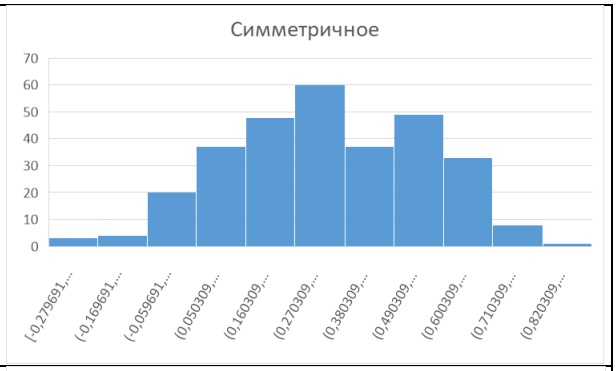
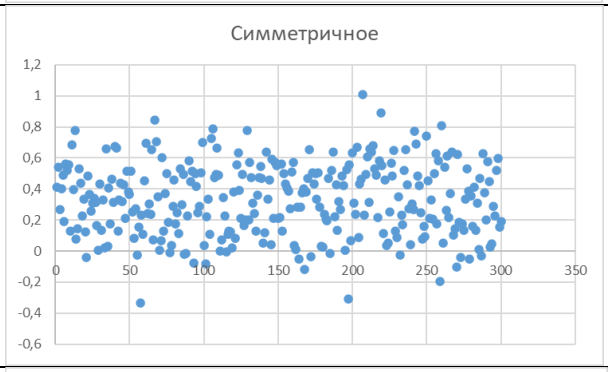
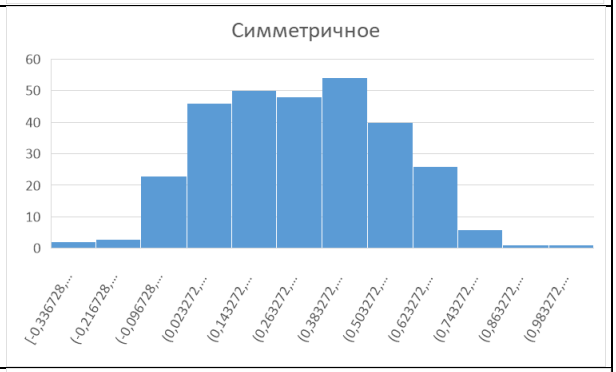
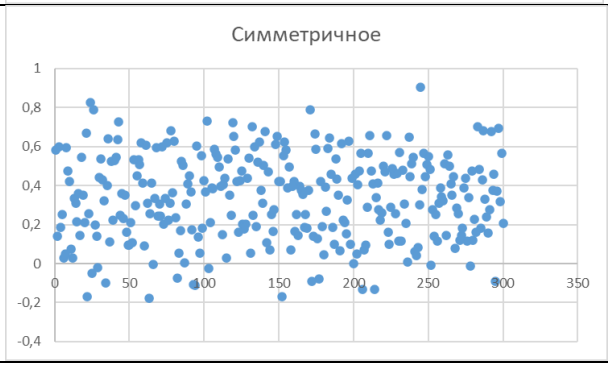
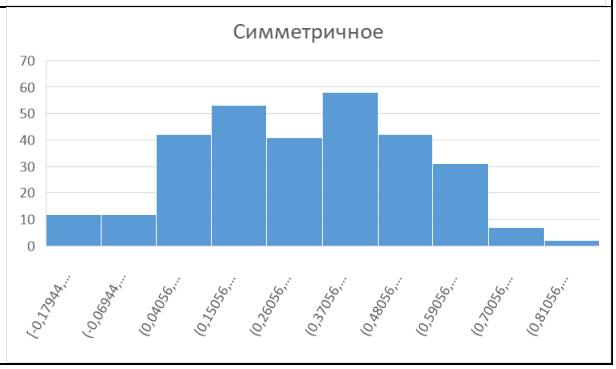


Оценки	Выборка 1	Выборка 2	Выборка 3	Истинное
Среднее арифметическое	8.132475e-01	8.009960e-01	7.989284e-01	0,8
Выборочная медиана	7.713477e-01	7.652102e-01	7.467556e-01	0,8
Усеченное среднее с разными уровнями усечения $0 \leq a \leq 0.5$ (0.05, 0.10, 0.15)	8.113987e-01 8.077888e-01 8.067978e-01	7.979909e-01 7.982107e-01 7.986934e-01	7.896302e-01 7.856339e-01 7.801901e-01	0,8
Оценка максимального правдоподобия	8.313389e-01	8.235383e-01	8.563133e-01	0,8
Обобщенные радикальные оценки с разными значениями параметра $\delta > 0$ (0.1, 0.5, 1)	8.329465e-01 8.382204e-01 8.424007e-01	8.226276e-01 8.190844e-01 8.148638e-01	8.538398e-01 8.443885e-01 8.341200e-01	0,8

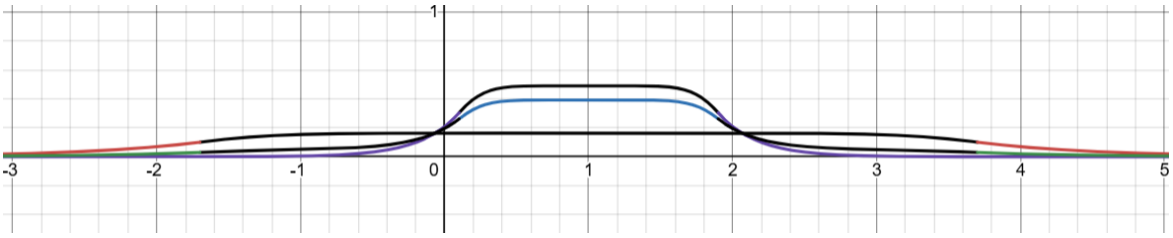
- **Для засоренного распределения с симметричным засорением (равные сдвиги у чистого и засоряющего распределений, масштаб у засоряющего больше в 2–4 раза, чем у чистого)**

$$\theta = 1, \ \ \lambda = 3$$

График (значения наблюдений как функции от номеров наблюдений).

1		
2		
3		

Сравнение чистой, засоряющей и засоренной плотности

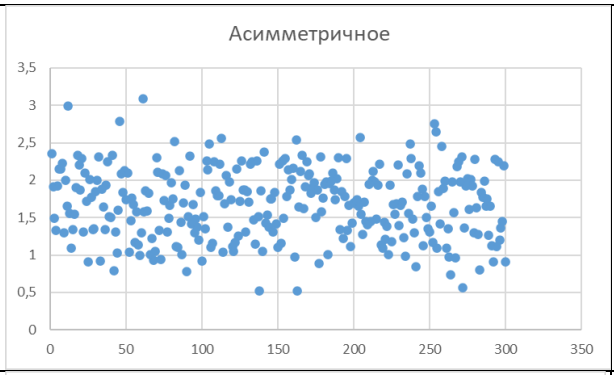
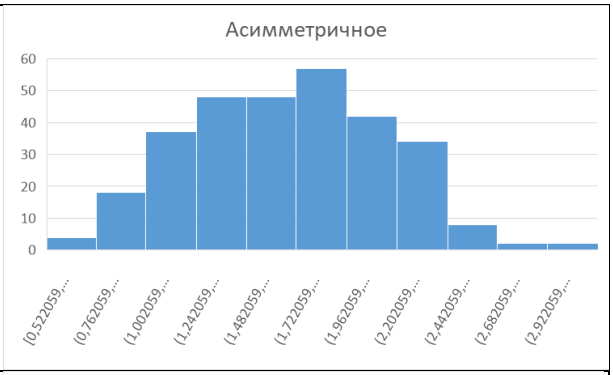
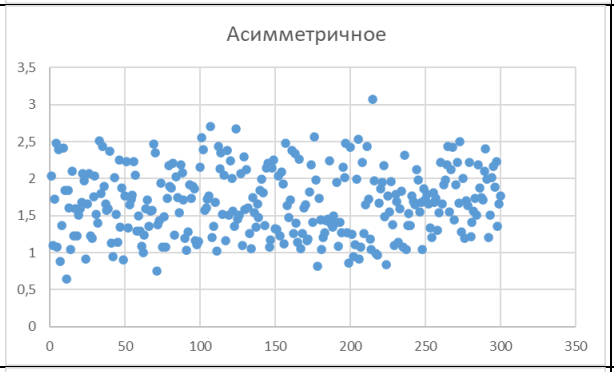
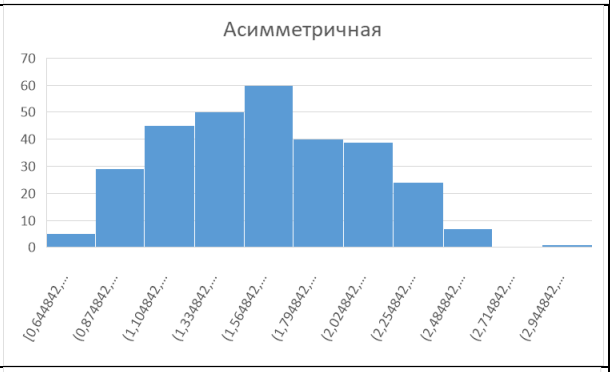
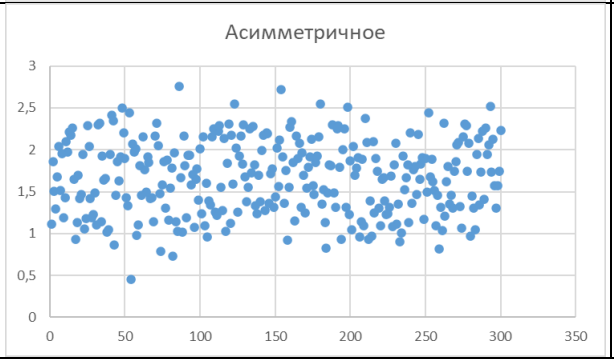
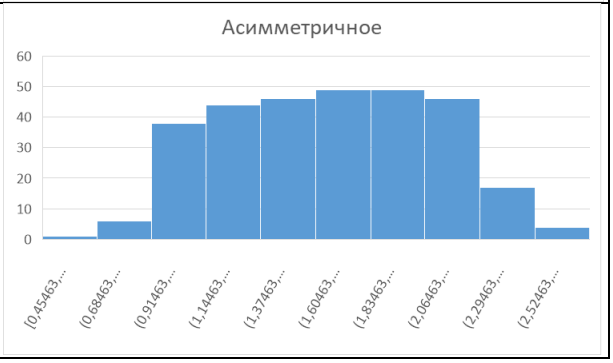


Оценки	Выборка 1	Выборка 2	Выборка 3	Истинное
Среднее арифметическое	6.799880e-01	6.498199e-01	7.066489e-01	0.8
Выборочная медиана	5.659886e-01	5.569663e-01	6.259975e-01	0.8
Усеченное среднее с разными уровнями усечения $0 \leq a \leq 0.5$ (0.05, 0.10, 0.15)	6.650715e-01 6.530625e-01 6.410262e-01	6.332987e-01 6.217225e-01 6.072108e-01	6.849744e-01 6.709671e-01 6.594188e-01	0.8
Оценка максимального правдоподобия	8.123443e-01	8.004893e-01	8.866460e-01	0.8
Обобщенные радикальные оценки с разными значениями параметра $\delta > 0$ (0.1, 0.5, 1)	8.123296e-01 8.122626e-01 8.121764e-01	8.004746e-01 8.004195e-01 8.003452e-01	8.866341e-01 8.865910e-01 8.865405e-01	0.8

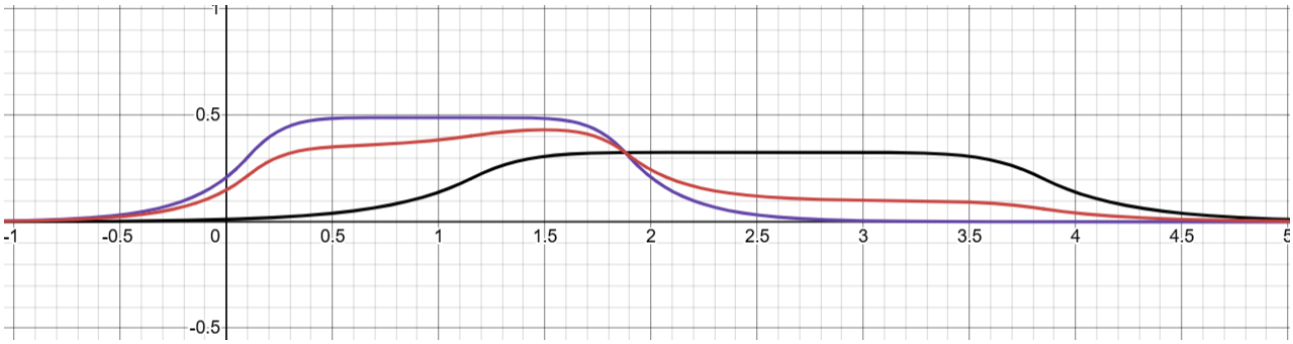
- **Для засоренного распределения с асимметричным засорением (сдвиги у чистого и засоряющего распределений отличаются на 2–4 стандартных отклонения, масштаб у засоряющего распределения не меньше, чем у чистого)**

$$\theta = 2.5, \quad \lambda = 1.5$$

График (значения наблюдений как функции от номеров наблюдений).

1		
2		
3		

Сравнение чистой, засоряющей и засоренной плотности

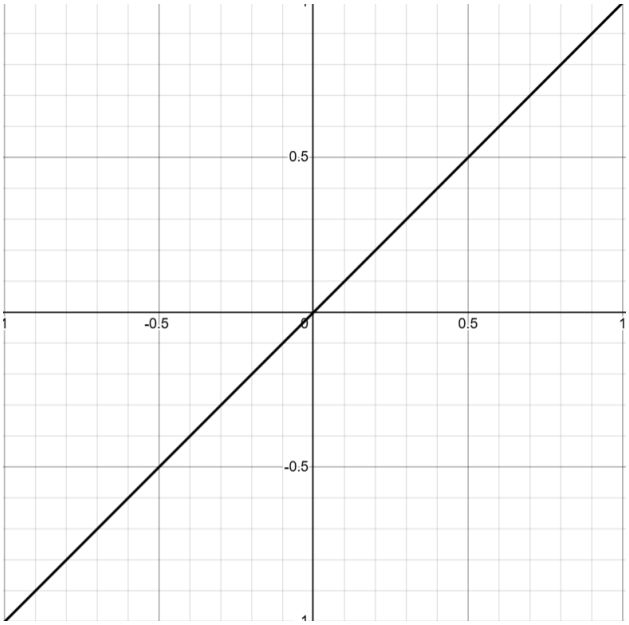


Фиолетовый – чистое, красное – засорённое. Чёрное – засоряющая.

Оценки	Выборка 1	Выборка 2	Выборка 3	Истинное
Среднее арифметическое	1.053332e+00	1.060558e+00	1.055169e+00	0,8
Выборочная медиана	1.055864e+00	1.070140e+00	1.042557e+00	0,8
Усеченное среднее с разными уровнями усечения $0 \leq a \leq 0.5$ (0.05, 0.10, 0.15)	1.046622e+00 1.042204e+00 1.043128e+00	1.055854e+00 1.054074e+00 1.053230e+00	1.049376e+00 1.045643e+00 1.047868e+00	0,8
Оценка максимального правдоподобия	1.089876e+00	1.124343e+00	1.095613e+00	0,8
Обобщенные радикальные оценки с разными значениями параметра $\delta > 0$ (0.1, 0.5, 1)	1.088659e+00 1.083677e+00 1.077402e+00	1.126735e+00 1.131711e+00 1.129691e+00	1.095344e+00 1.092450e+00 1.087281e+00	0,8

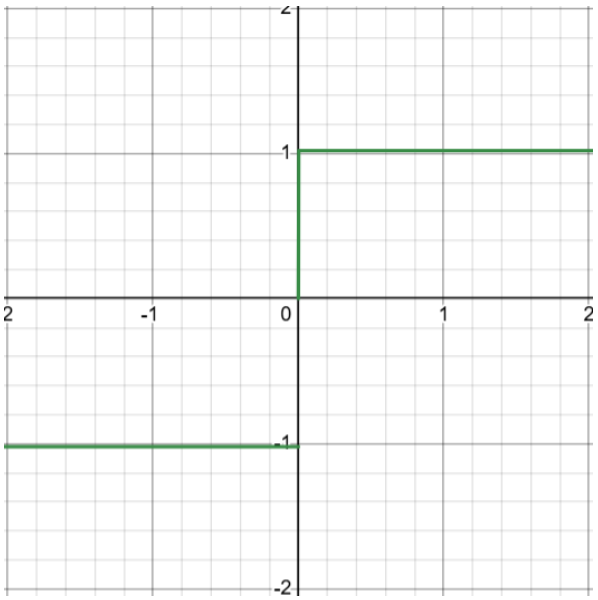
- **График с функциями влияния для всех использованных оценок.**
- Среднее арифметическое

$$IF(y) = y - \theta$$



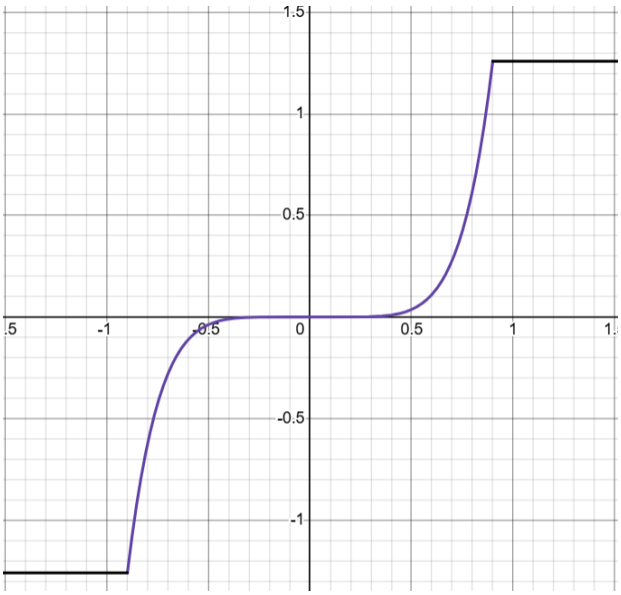
- Выборочная медиана

$$IF(y) = \frac{\lambda sign(y-\theta)}{2f(0)}$$



- Оценка максимального правдоподобия

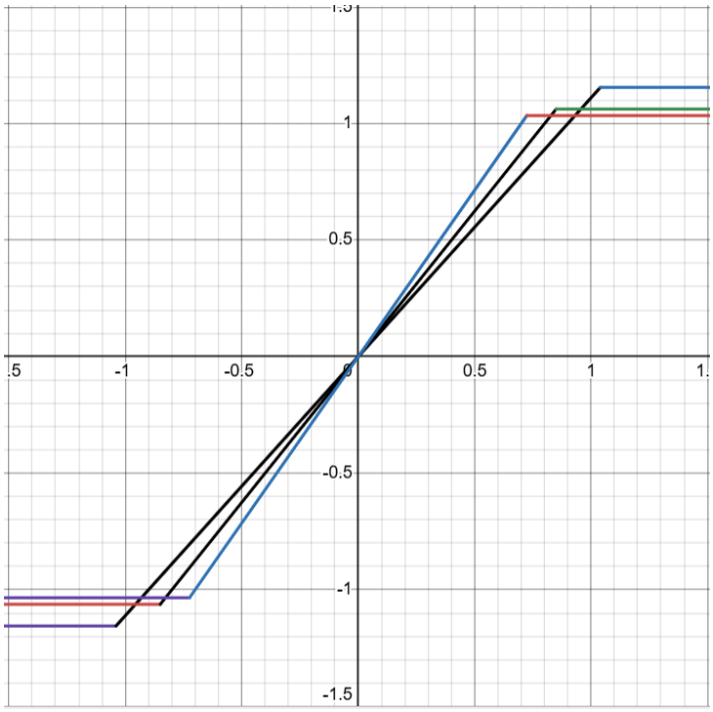
$$IF(y) = -\frac{\lambda\left(\frac{f'\left(\frac{y-\theta}{\lambda}\right)}{f\left(\frac{y-\theta}{\lambda}\right)}\right)}{\left(\int_{-\infty}^{\infty}\frac{[f'(z)]^2}{f(z)}dz\right)}$$



- Усеченное среднее с разными уровнями усечения $\alpha \in (0.05, 0.10, 0.15)$

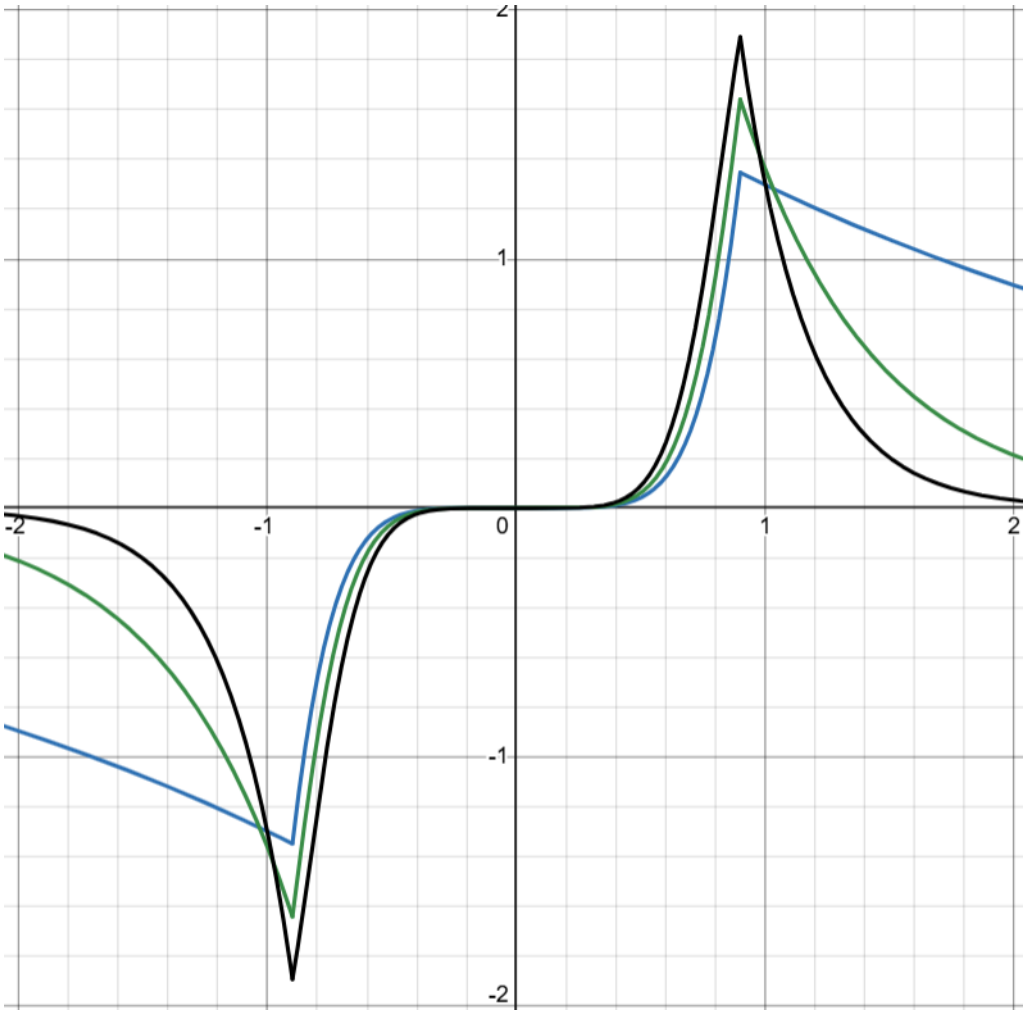
$$IF(y) = \frac{1}{1-2\alpha}\begin{cases} -q, & \frac{y-\theta}{\lambda} < -q \\ \frac{y-\theta}{\lambda}, & \left|\frac{y-\theta}{\lambda}\right| \le q \\ q, & \frac{y-\theta}{\lambda} > q \end{cases}$$

$$\hat{q} = \frac{\tilde{y}_{[(1-\alpha)N]+1} - \theta}{\lambda}$$



- Обобщенные радикальные оценки с разными значениями параметра $\delta \in (0.1, 0.5, 1)$

$$IF(y) = -\frac{\lambda\left(f'\left(\frac{y-\theta}{\lambda}\right) f^{\delta-1}\left(\frac{y-\theta}{\lambda}\right)\right)}{\left(\int_{-\infty}^{\infty}[f'(z)]^2 f^{\delta-1}(z)dz\right)}$$



4. Выводы

Все робастные и не робастные оценки работают одинаково для чистого распределения. При симметричном засорении робастные оценки показывают истинное значения параметра сдвига. При асимметричном засорения все рассмотренные оценки получились неустойчивыми.

Полученные значения сдвига, соответствует условию асимптотической несмещённости $E\psi(y, \theta) = 0$, получили что ноль функции влияния оценок $IF(0)$ совпадает с найденным значением сдвига.

5. Текст программы

```
#define _USE_MATH_DEFINES
#define _WRITE

#include <cstdlib>
```

```
#include <ctime>
#include <cstdio>
#include <cmath>
#include <fstream>
#include "spec_func.hpp"

using namespace std;

const int N = 300;
const double EPS = 0.30;                // clogging probability

const double PURE_TETTA = 1.0;          // shift for pure distribution
const double PURE_LAMBDA = 1.25;         // scale for pure distribution

const double SYM_TETTA = PURE_TETTA;    // shift for symmetrical clogging distribution
const double SYM_LAMBDA = 3.0;          // scale for symmetrical clogging distribution

const double ASYM_TETTA = 2.5;          // shift for asymmetrical clogging distribu-
tion
const double ASYM_LAMBDA = PURE_LAMBDA + 0.25;  // scale for asymmetrical clogging dis-
tribution

double ALPHA = 0.0;                     // trimmed mean parameter
double DELTA = 0.0;                     // radical function power coef

double CalcPureValue(double P, double v, double a)
{
	//printf("Calculating value..\n");
	double r1, r2, r3, r4, r5;
	double x1, x2, z;

	double one_seven = 1.0 / 7.0;
	double two_seven = 2.0 / 7.0;

	double seven_one_seven = pow(7, -one_seven);
	double seven_two_seven = pow(7, two_seven);

	double five_fourteen = 5.0 / 14.0;

	do {
		r1 = (double)rand() / RAND_MAX;
		r3 = (double)rand() / RAND_MAX;
	} while (r1 == 0 || r3 == 0 || r1 == 1 || r3 == 1);

	if (r1 <= P)
	{
		while (true)
		{
			do {
				r4 = (double)rand() / RAND_MAX;
				r5 = (double)rand() / RAND_MAX;
				r2 = (double)rand() / RAND_MAX;
			} while (r4 == 0 || r5 == 0 || r2 == 0 || r4 == 1 || r5 == 1 || r2
== 1);

			z = sqrt(-2 * log(r4)) * cos(2 * M_PI * r5);
			x1 = seven_one_seven * z;
			if (abs(x1) <= v) {
				//printf("Not Done\n");
				if (log(r2) <= -pow(abs(x1), 7) + seven_two_seven * x1 * x1 *
0.5 - five_fourteen)
				{
```

```
                                //printf("Done\n");
                                return x1;
                        }
                }
        }
    }

    x2 = v - (log(r3) / a);

    //printf("Done\n");

    if (r1 < (1.0 + P) * 0.5)
        return x2;
    return -x2;

}

double CalcDistributionFunc(double x, double v, double v7, double K, double a)
{
    double result = 0;
    if (abs(x) > v)
    {
        double tmp = -a * (abs(x) - v) - v7;
        result = exp(tmp);
    }
    else
        result = exp(-pow(abs(x), 7));
    return result / K;
}

double CalcMLEFunc(double* values, double v, double v7, double K, double a, double
lambda, double tetta)
{
    double result = 0;

    for (int i = 0; i < N; i++)
        result += -log(CalcDistributionFunc((values[i] - tetta) / lambda, v, v7,
K, a));

    return result / N;
}

double CalcRadicalFunc(double* values, double v, double v7, double K, double a, double
lambda, double tetta)
{
    double result = 0;

    double divider = pow(CalcDistributionFunc(0, v, v7, K, a), DELTA);

    for (int i = 0; i < N; i++)
        result += -pow(CalcDistributionFunc((values[i] - tetta) / lambda, v, v7,
K, a), DELTA) / divider;

    return result / N;
}

double CalcMLE(double* values, double v, double v7, double K, double a, double lambda,
double eps)
{
    double gld = 1.6180339887498948482;
    double n = -1;
    double m =  1;
```

```
	double x1;
	double x2;

	while (abs(m - n) > eps)
	{
		x1 = m - (m - n) / gld;
		x2 = n + (m - n) / gld;

		if (CalcMLEFunc(values, v, v7, K, a, lambda, x1) < CalcMLEFunc(values, v,
v7, K, a, lambda, x2))
			m = x2;
		else
			n = x1;
	}

	return (n + m) / 2;
}

double CalcRadical(double* values, double v, double v7, double K, double a, double
lambda, double eps)
{
	double gld = 1.6180339887498948482;
	double n = -1;
	double m =  1;

	double x1;
	double x2;

	while (abs(m - n) > eps)
	{
		x1 = m - (m - n) / gld;
		x2 = n + (m - n) / gld;

		if (CalcRadicalFunc(values, v, v7, K, a, lambda, x1) < CalcRadi-
calFunc(values, v, v7, K, a, lambda, x2))
			m = x2;
		else
			n = x1;
	}

	return (n + m) / 2;
}

double CalcMLEIF()
{
	double result = 0.0;

	return result;
}

double CalcRadicalIF()
{
	double result = 0.0;

	return result;
}

double CalcArithmeticMean(double *values)
{
	double result = 0;
	for (int i = 0; i < N; i++)
```

```
		result += values[i];
	return result / N;
}

double CalcDispersion(double* values, double y_)
{
	double result = 0;
	for (int i = 0; i < N; i++)
		result += (values[i] - y_) * (values[i] - y_);
	return result / N;
}

double CalcSampleMedian(double* values)
{
	double result = 0;
	if (N % 2 == 0)
		result = (values[N / 2] + values[(N / 2) - 1]) / 2.0;
	else
		result = values[N / 2];
	return result;
}

double CalcAsymmetryCoefficient(double* values, double y_, double D)
{
	double result = 0;
	for (int i = 0; i < N; i++)
		result += (values[i] - y_) * (values[i] - y_) * (values[i] - y_);
	return result / (N * pow(D, 1.5));
}

double CalcKurtosisCoefficient(double* values, double y_, double D)
{
	double result = 0.0;
	for (int i = 0; i < N; i++)
		result += (values[i] - y_) * (values[i] - y_) * (values[i] - y_) * (val-
ues[i] - y_);
	result /= (N * D * D);
	return result - 3.0;
}

double CalcTrimmedMean(double* values)
{
	double result = 0;
	int k = N * ALPHA;
	for (int i = k; i < N - k; i++)
		result += values[i];
	return result / (N - 2 * k);
}

int main()
{
#pragma region Constants

	double v = 0.9;
	double v2 = v * v;
	double v3 = v2 * v;
	double v4 = v3 * v;
	double v5 = v4 * v;
	double v6 = v5 * v;
	double v7 = v6 * v;

	double a = 7.0 * v6;
```

```
        double a2 = a * a;
        double a3 = a2 * a;
        double a4 = a3 * a;
        double a5 = a4 * a;

        double one_seven = 1.0 / 7.0;
        double two_seven = 2.0 / 7.0;
        double three_seven = 3.0 / 7.0;
        double five_seven = 5.0 / 7.0;
        double five_fourteen = 5.0 / 14.0;

        double seven_one_seven = pow(7, -one_seven);
        double seven_two_seven = pow(7, two_seven);

        double ev7 = exp(-v7);

        double K = two_seven * (igamma(one_seven, v7) + ev7 / v6);

        double variance = (2.0 / K) * ((igamma(three_seven, v7) * one_seven) + ev7 * ((2
/ a3) +

(2 * v / a2) +

(v2 / a)));

        double igamma2 = (2.0 / K) * ((igamma(five_seven, v7) * one_seven) + ev7 * ((24 /
a5) +

                                        (24 * v / a4) +

                                        (12 * v2 / a3) +

                                        (4 * v3 / a2) +

                                        (v4 / a)));
        igamma2 /= variance * variance;
        igamma2 -= 3.0;

        double P = 2.0 * igamma(one_seven, v7) / (7.0 * K);

        printf("Constants:\n");
        printf("v: %e\n", v);
        printf("variance: %e\n", variance);
        printf("igamma2: %e\n", igamma2);
        printf("P: %e\n", P);
        printf("K: %e\n\n", K);

#pragma endregion

        srand(time(nullptr));

        double* pure_values = new double[N];

        double* sym_clog_values = new double[N];
        double* sym_dirt_values = new double[N];

        double* asym_clog_values = new double[N];
        double* asym_dirt_values = new double[N];

        double r = 0.0;

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

```
	{
		pure_values[i] = (CalcPureValue(P, v, a) + PURE_TETTA) / PURE_LAMBDA;
	}

	for (int i = 0; i < N; i++)
	{
		sym_clog_values[i] = (CalcPureValue(P, v, a) + SYM_TETTA) / SYM_LAMBDA;

		r = (double)rand() / RAND_MAX;
		if (r < EPS)
			sym_dirt_values[i] = sym_clog_values[i];
		else
			sym_dirt_values[i] = pure_values[i];
	}

	for (int i = 0; i < N; i++)
	{
		asym_clog_values[i] = (CalcPureValue(P, v, a) + ASYM_TETTA) / ASYM_LAMBDA;

		r = (double)rand() / RAND_MAX;
		if (r < EPS)
			asym_dirt_values[i] = asym_clog_values[i];
		else
			asym_dirt_values[i] = pure_values[i];
	}

	qsort(pure_values, N, sizeof(double),
		[](const void* a, const void* b)
		{
			const double* x = (double*)a;
			const double* y = (double*)b;

			if (*x > *y)
				return 1;
			else if (*x < *y)
				return -1;

			return 0;
		}
	);

	qsort(sym_dirt_values, N, sizeof(double),
		[](const void* a, const void* b)
		{
			const double* x = (double*)a;
			const double* y = (double*)b;

			if (*x > *y)
				return 1;
			else if (*x < *y)
				return -1;

			return 0;
		}
	);

	qsort(asym_dirt_values, N, sizeof(double),
		[](const void* a, const void* b)
		{
			const double* x = (double*)a;
			const double* y = (double*)b;
```

```
                    if (*x > *y)
                        return 1;
                    else if (*x < *y)
                        return -1;

                    return 0;
                }
        );

        double y_, D, ac, kc, sm, tm, MLE, RAD;

#pragma region Pure

        printf("Pure distribution (shift: %e, scale %e):\n\n", PURE_TETTA, PURE_LAMBDA);

        y_ = CalcArithmeticMean(pure_values);

        printf("Arithmetic Mean: %e\n", y_);

        D = CalcDispersion(pure_values, y_);

        printf("Dispersion: %e\n", D);

        ac = CalcAsymmetryCoefficient(pure_values, y_, D);

        printf("Assymetry Coefficient: %e\n", ac);

        kc = CalcKurtosisCoefficient(pure_values, y_, D);

        printf("Kurtosis Coefficient: %e\n", kc);

        sm = CalcSampleMedian(pure_values);

        printf("Sample Median: %e\n", sm);

        ALPHA = 0.05;
        tm = CalcTrimmedMean(pure_values);
        printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

        ALPHA = 0.10;
        tm = CalcTrimmedMean(pure_values);
        printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

        ALPHA = 0.15;
        tm = CalcTrimmedMean(pure_values);
        printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

        MLE = CalcMLE(pure_values, v, v7, K, a, PURE_LAMBDA, 1e-5);

        printf("MLE: %e\n", MLE);

        DELTA = 0.1;
        RAD = CalcRadical(pure_values, v, v7, K, a, PURE_LAMBDA, 1e-5);
        printf("Radical (DELTA = %e): %e\n", DELTA, RAD);

        DELTA = 0.5;
        RAD = CalcRadical(pure_values, v, v7, K, a, PURE_LAMBDA, 1e-5);
        printf("Radical (DELTA = %e): %e\n", DELTA, RAD);

        DELTA = 1.0;
        RAD = CalcRadical(pure_values, v, v7, K, a, PURE_LAMBDA, 1e-5);
        printf("Radical (DELTA = %e): %e\n\n\n", DELTA, RAD);
```

```
#pragma endregion

#pragma region Sym Dirt

	printf("Symmetrical dirt distribution (shift: %e, scale %e):\n\n", SYM_TETTA,
SYM_LAMBDA);

	y_ = CalcArithmeticMean(sym_dirt_values);

	printf("Arithmetic Mean: %e\n", y_);

	D = CalcDispersion(sym_dirt_values, y_);

	printf("Dispersion: %e\n", D);

	ac = CalcAsymmetryCoefficient(sym_dirt_values, y_, D);

	printf("Assymetry Coefficient: %e\n", ac);

	kc = CalcKurtosisCoefficient(sym_dirt_values, y_, D);

	printf("Kurtosis Coefficient: %e\n", kc);

	sm = CalcSampleMedian(sym_dirt_values);

	printf("Sample Median: %e\n", sm);

	ALPHA = 0.05;
	tm = CalcTrimmedMean(sym_dirt_values);
	printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

	ALPHA = 0.10;
	tm = CalcTrimmedMean(sym_dirt_values);
	printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

	ALPHA = 0.15;
	tm = CalcTrimmedMean(sym_dirt_values);
	printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

	MLE = CalcMLE(sym_dirt_values, v, v7, K, a, SYM_LAMBDA, 1e-5);

	printf("MLE: %e\n", MLE);

	DELTA = 0.1;
	RAD = CalcRadical(sym_dirt_values, v, v7, K, a, SYM_LAMBDA, 1e-5);
	printf("Radical (DELTA = %e): %e\n", DELTA, RAD);

	DELTA = 0.5;
	RAD = CalcRadical(sym_dirt_values, v, v7, K, a, SYM_LAMBDA, 1e-5);
	printf("Radical (DELTA = %e): %e\n", DELTA, RAD);

	DELTA = 1.0;
	RAD = CalcRadical(sym_dirt_values, v, v7, K, a, SYM_LAMBDA, 1e-5);
	printf("Radical (DELTA = %e): %e\n\n\n", DELTA, RAD);

#pragma endregion

#pragma region Asym Dirt

	printf("Asymmetrical dirt distribution (shift: %e, scale %e):\n\n", ASYM_TETTA,
ASYM_LAMBDA);
```

```
	y_ = CalcArithmeticMean(asym_dirt_values);

	printf("Arithmetic Mean: %e\n", y_);

	D = CalcDispersion(asym_dirt_values, y_);

	printf("Dispersion: %e\n", D);

	ac = CalcAsymmetryCoefficient(asym_dirt_values, y_, D);

	printf("Assymetry Coefficient: %e\n", ac);

	kc = CalcKurtosisCoefficient(asym_dirt_values, y_, D);

	printf("Kurtosis Coefficient: %e\n", kc);

	sm = CalcSampleMedian(asym_dirt_values);

	printf("Sample Median: %e\n", sm);

	ALPHA = 0.05;
	tm = CalcTrimmedMean(asym_dirt_values);
	printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

	ALPHA = 0.10;
	tm = CalcTrimmedMean(asym_dirt_values);
	printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

	ALPHA = 0.15;
	tm = CalcTrimmedMean(asym_dirt_values);
	printf("Trimmed Mean (ALPHA = %e): %e\n", ALPHA, tm);

	MLE = CalcMLE(asym_dirt_values, v, v7, K, a, ASYM_LAMBDA, 1e-5);

	printf("MLE: %e\n", MLE);

	DELTA = 0.1;
	RAD = CalcRadical(asym_dirt_values, v, v7, K, a, ASYM_LAMBDA, 1e-5);
	printf("Radical (DELTA = %e): %e\n", DELTA, RAD);

	DELTA = 0.5;
	RAD = CalcRadical(asym_dirt_values, v, v7, K, a, ASYM_LAMBDA, 1e-5);
	printf("Radical (DELTA = %e): %e\n", DELTA, RAD);

	DELTA = 1.0;
	RAD = CalcRadical(asym_dirt_values, v, v7, K, a, ASYM_LAMBDA, 1e-5);
	printf("Radical (DELTA = %e): %e\n\n\n", DELTA, RAD);

#pragma endregion

#ifdef _WRITE
#pragma region WriteToFile

	ofstream file;

	file.open("data_pure.csv");

	for (int i = 0; i < N; i++)
		file << pure_values[i] << ' ' << endl;

	file.close();
```

```
	file.open("data_sym_clog.csv");

	for (int i = 0; i < N; i++)
		file << sym_clog_values[i] << ' ' << endl;

	file.close();

	file.open("data_sym_dirt.csv");

	for (int i = 0; i < N; i++)
		file << sym_dirt_values[i] << ' ' << endl;

	file.close();

	file.open("data_asym_clog.csv");

	for (int i = 0; i < N; i++)
		file << asym_clog_values[i] << ' ' << endl;

	file.close();

	file.open("data_asym_dirt.csv");

	for (int i = 0; i < N; i++)
		file << asym_dirt_values[i] << ' ' << endl;

	file.close();

#pragma endregion
#endif

	return 0;
	}
```