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

Практическое задание № 1

по дисциплине «СППМиНПО»

УСТОЙЧИВЫЕ МЕТОДЫ ОЦЕНИВАНИЯ ПАРАМЕТРОВ СТАТИСТИЧЕСКИХ МОДЕЛЕЙ

Бригада 1	КАСИМОВ ТИМУР
Группа ПММ-21	БАРИЕВ РОДИОН
Вариант 7в	ЧЕРНЕНКО ДАНИЛА

Преподаватели ЛИСИЦИН Д. В.

Новосибирск, 2022

1. **Цель работы**

Изучить методы робастного оценивания параметра сдвига распределений случайных величин.

2. **Вариант**

Распределение Хьюбера со значением параметра $\nu = 0.05$.

3. **Содержание работы (уровень выполнения №2)**

1. Разработать программу, которая реализует:
2. Провести проверку генератора чистого распределения путем сравнения выборочных характеристик с их теоретическими значениями на выборках большого объема (N порядка $10^5 - 10^7$); 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 следующие оценки параметра сдвига:
    - среднее арифметическое;
    - выборочная медиана;
    - оценка максимального правдоподобия;
    - усеченное среднее с разными уровнями (как минимум три обязательных значения $0.05, 0.1, 0.15$);
    - обобщенные радикальные оценки с разными значениями параметра (как минимум три обязательных значения $0.1, 0.5, 1$).

    Использовать выборки, имеющие слеудющие виды распределений:
    - чистое распределение;
    - засоренное распределение с симетричным засорением (равные сдвиги у чистого и засоряющего распределений, масштаб у засоряющего больше в 2-3 раза, чем у чистого);
    - засоренное распределение с ассиметричным засорением (сдвиги у чистого и засоряющего распределений отличаются на 2-3 стандартных отклонения, масштаб у засоряющего распределения не меньше, чем у чистого).

    При выборе параметров засорения ориентироваться на график чистой, засоряющей и засоренной плотностей. Рекомендуемый уровень засорения $\varepsilon = 0.05 - 0.4$. Рекомендуемый объем выборки $N = 100 - 1000$. Сравнить устойчивость оценок для распределений указанных видов (минимум по три выборки для каждого набор значений параметров) по их отклонению от истинного значения, сопоставить результаты сравнения со свойствами фукнций влияния оценок.

4. Ход работы

Стандартное распределение Хьюбера имеет плотность

$$f(x,\nu) = \frac{1-\nu}{\sqrt{2\pi}} \begin{cases} \exp\{-\frac{x^2}{2}\}, |x| \le k \\ \exp\{\frac{1}{2}k^2 - k|x|\}, |x| > k \end{cases},$$

в нашем случае $k = 1.398, \nu = 0.05$.

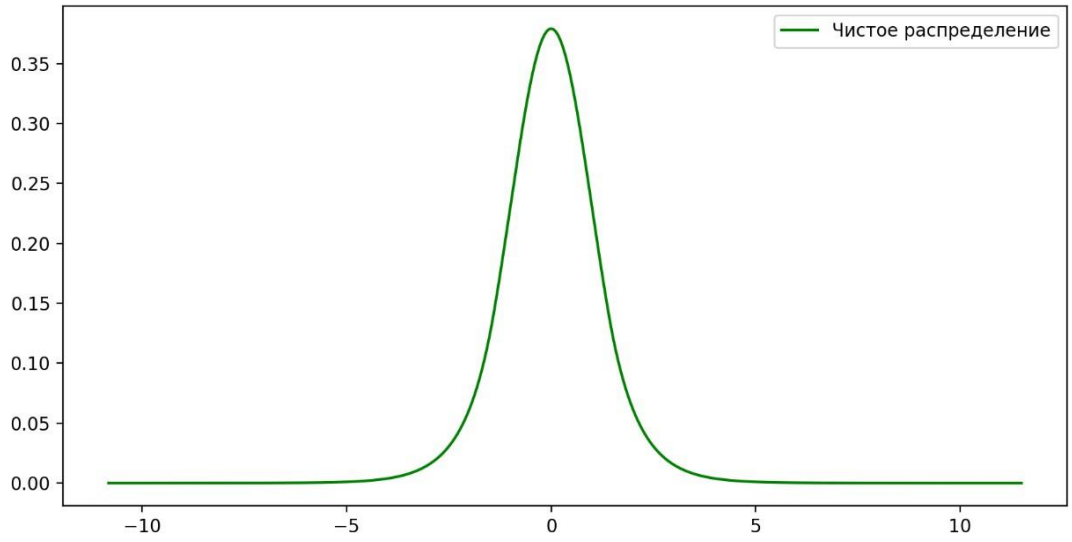
Для генерации данных использовался следующий алгоритм:

1. Вычислить константу $P = \frac{2(1-\nu)}{k} f(k,\nu)$ один раз для каждого значения ν.
2. Сгенерировать псевдослучайное число r_1, равномерно распределенное на интервале (0, 1). Если $r_1 \ge P$, перейти на шаг 3, иначе перейти на шаг 5.
3. Сгенерировать стандартное нормальное псевдослучайное число x_1.
4. Если x_1 принадлежит интервалу $[-k, k]$, то это искомое псевдослучайное число, иначе перейти на шаг 3.
5. Сгенерировать псевдослучайное число r_2, равномерно распределенное на интервале (0, 1). Вычислить $x_2 = k - \frac{lnr_2}{k}$.
6. Если $r_1 < \frac{P}{2^{\nu}}$ то x – искомое псевдослучайное число, иначе искомым псевдослучайным числом является $-x$.

Для проверки генераторы использовалась выборка размером 10^7 элементов.

Характеристика	Теоретически ожидаемое значение	Полученное значение
Дисперсия	1.41	1.36828
Коэффициент эксцесса	4.52	4.60256

График функции плотности имеет следующий вид:



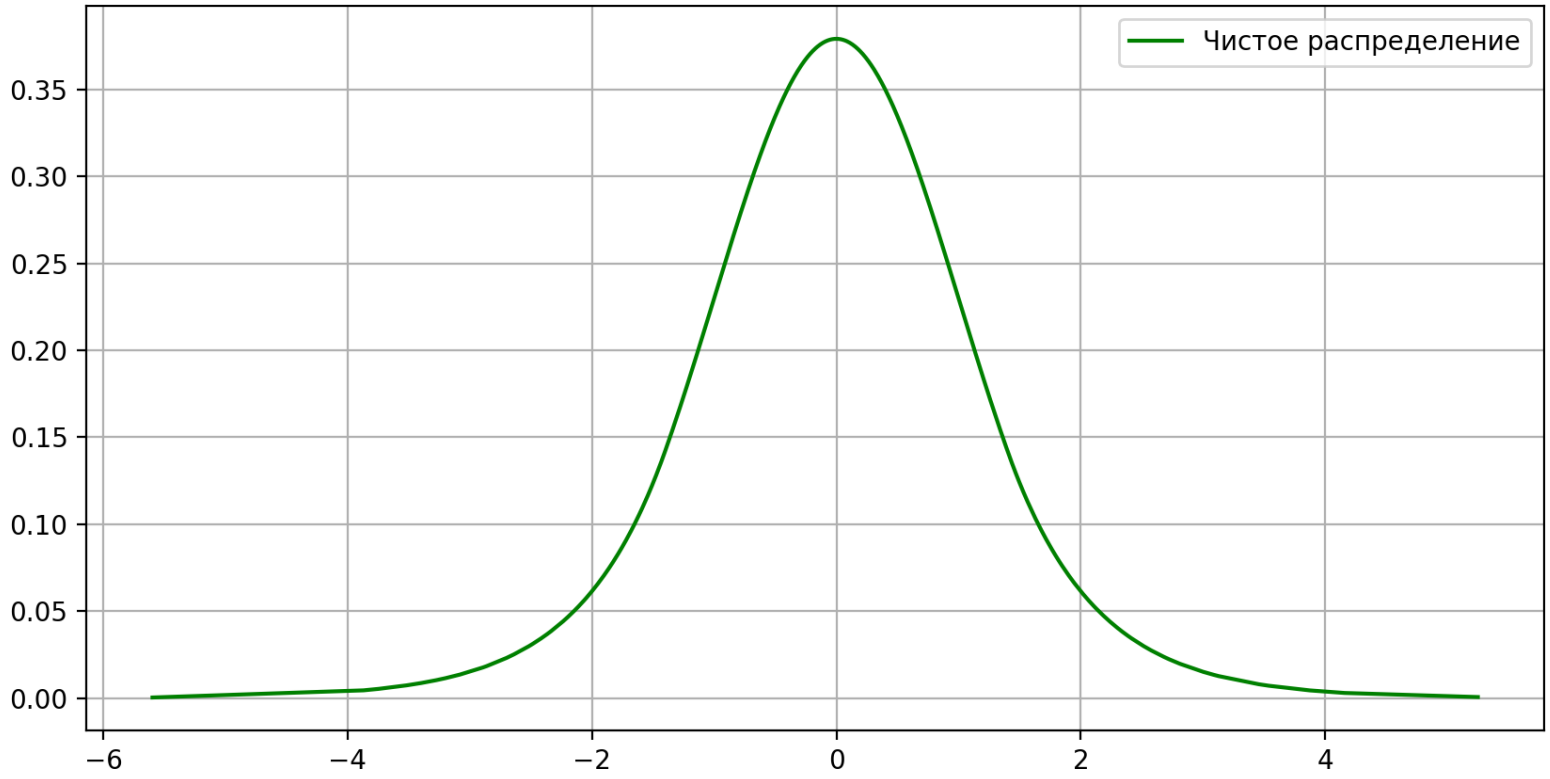
Прочие статистические характеристики:

- Среднее: 0.00026
- Медиана: 0.00023
- Ассиметрия: -0.00107
- Максимальное значение: 11.49666
- Минимальное значение: -10.81895
- ОМП: 0.00017
- Усеченное среднее (0.05): 0.00032
- Усеченное среднее (0.1): 0.00038
- Усеченное среднее (0.15): 0.00037

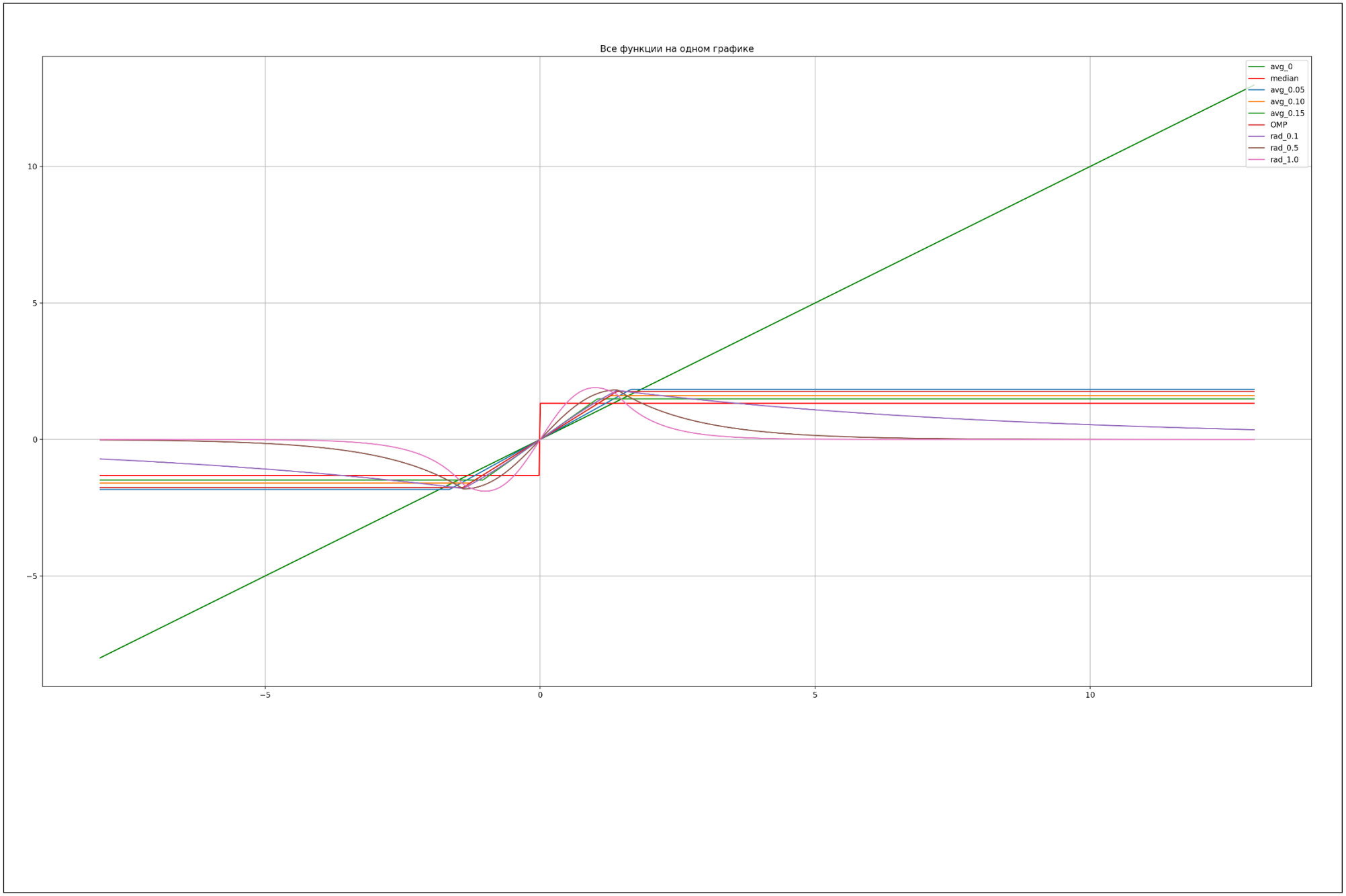
Оценки параметра сдвига для разных распределений

Объем выборки во всех выборках 1000 элементов. Истинное значение параметра сдвига во всех распределениях 0.

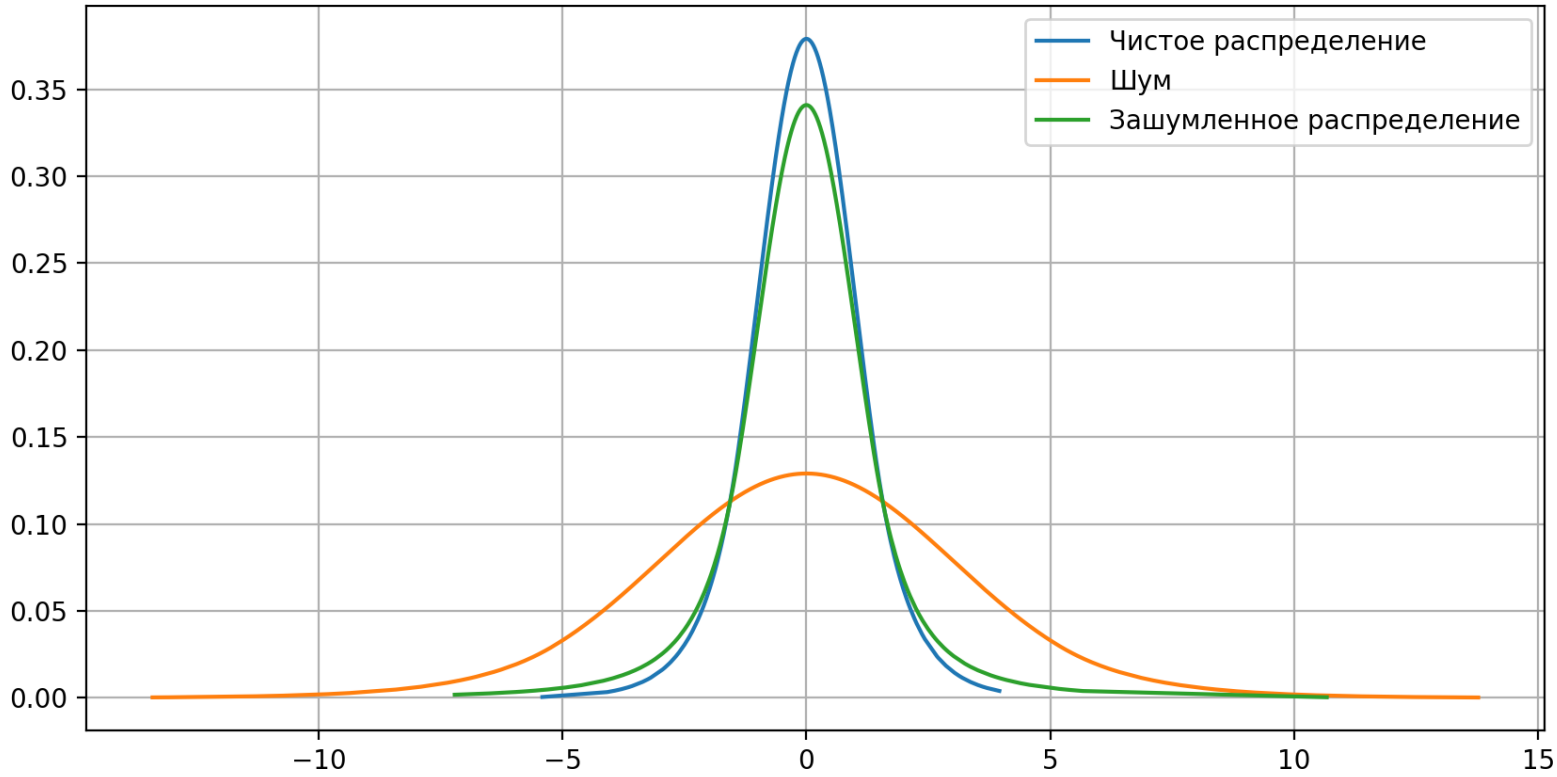
Чистое распределение ($\theta = 0, \lambda = 1, \varepsilon = 0$)



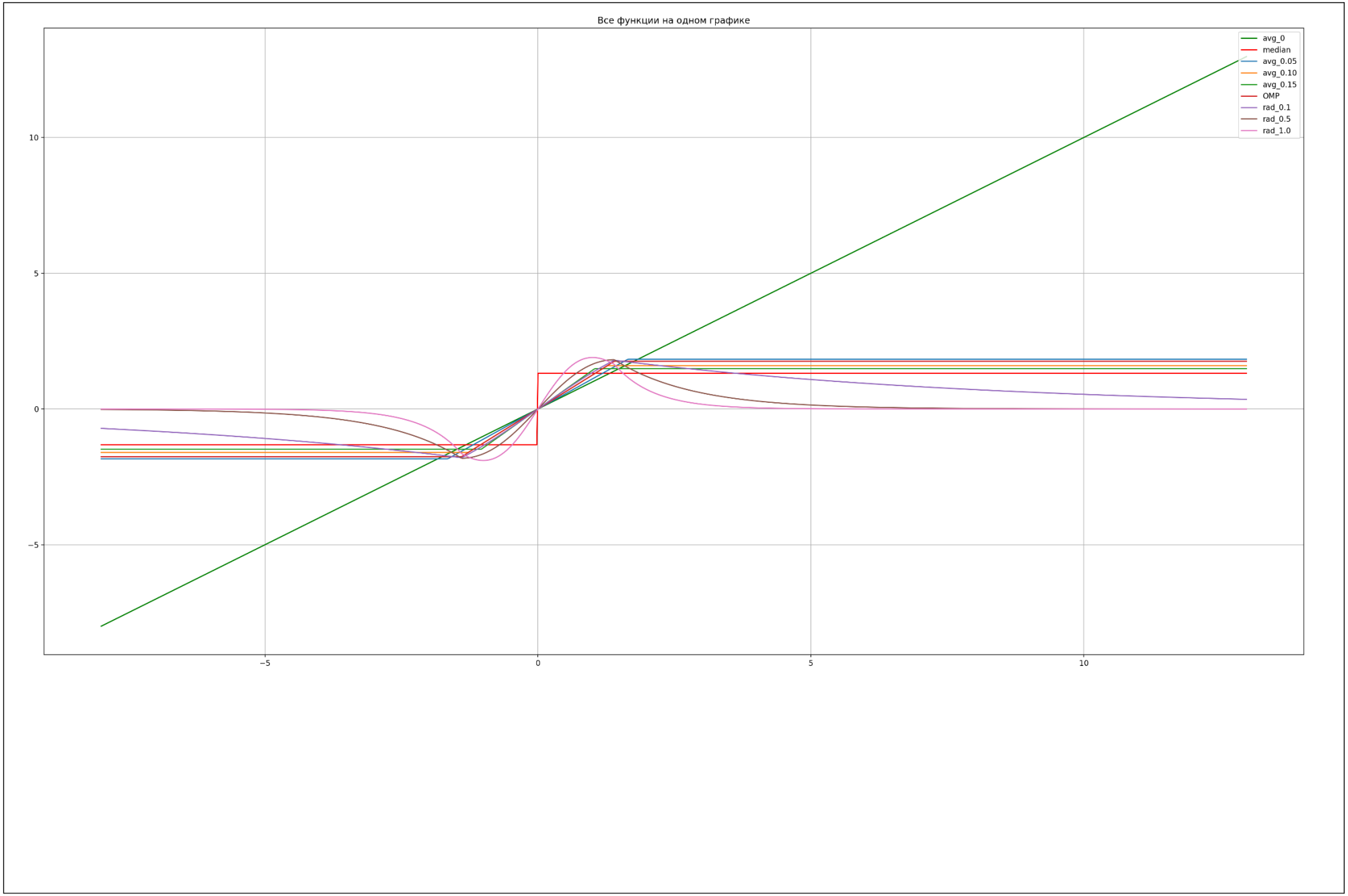
Номер выборки	1	2	3
Среднее арифметическое	-0.01702	-0.04922	-0.02416
Выборочная медиана	-0.01779	-0.05904	-0.03607
Дисперсия	1.31789	1.36590	1.33357
Ассиметрия	0.05445	-0.18584	-0.08010
Эксцесс	4.35242	4.55714	5.37652
Максимальное значение	5.24948	4.78132	4.53205
Минимальное значение	-5.59602	-5.08408	-5.76715
Оценка максимального правдоподобия	0.08743	0.06248	0.01076
Усеченное среднее (уровень 0.05)	-0.02414	-0.03673	-0.02978
Усеченное среднее (уровень 0.10)	-0.02496	-0.03619	-0.03183
Усеченное среднее (уровень 0.15)	-0.02079	-0.03298	-0.03022
Обобщенные радикальные оценки (параметр 0.1)	-0.02520	-0.03322	-0.03262
Обобщенные радикальные оценки (параметр 0.5)	-0.01999	-0.02738	-0.03420
Обобщенные радикальные оценки (параметр 1)	-0.00691	-0.02553	-0.02945


Все функции на одном графике
avg_0
median
avg_0.05
avg_0.10
avg_0.15
OMP
rad_0.1
rad_0.5
rad_1.0
10
5
0
−5
−5
0
5
10

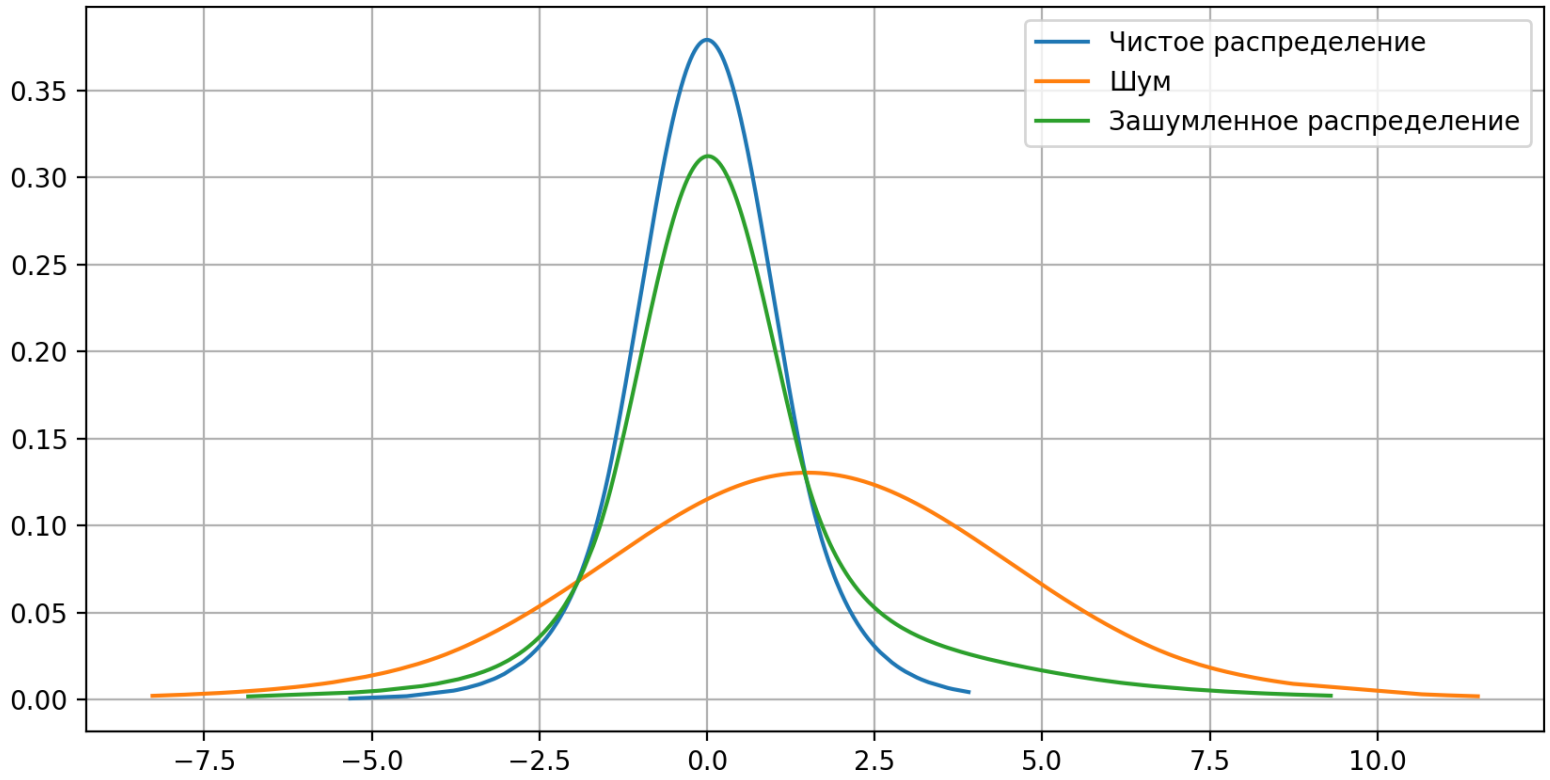
Распределение с симметричным засорением ($\theta = 0, \lambda = 3.03, \varepsilon = 0.15$)



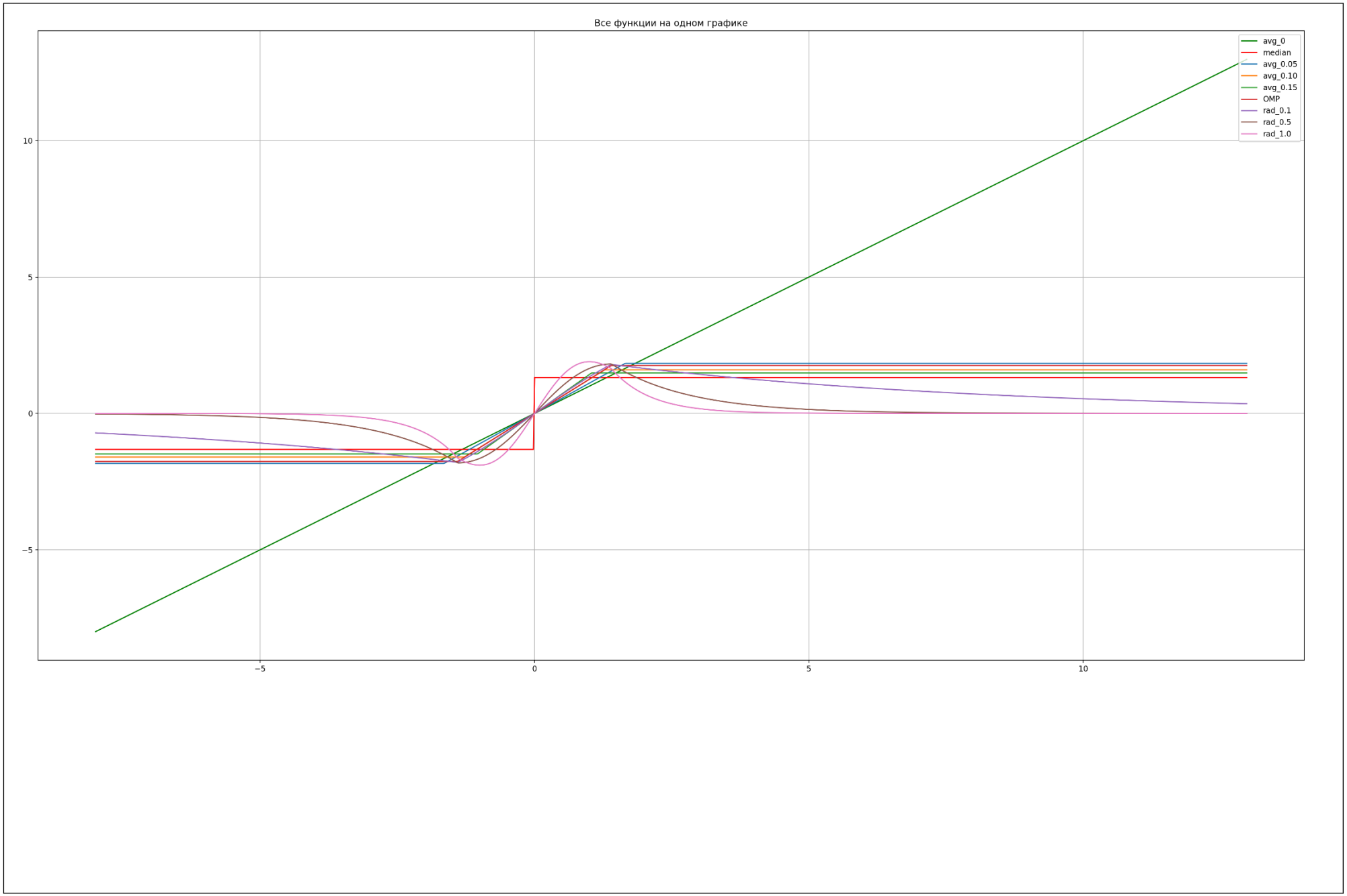
Номер выборки	1	2	3
Среднее арифметическое	-0.01924	-0.01217	-0.06131
Выборочная медиана	-0.00016	-0.01989	-0.01056
Дисперсия	2.58701	2.83885	2.96134
Ассиметрия	0.13041	-0.15220	-0.83855
Эксцесс	7.23873	7.27586	10.20051
Максимальное значение	10.66631	9.25153	7.11106
Минимальное значение	-7.21025	-8.88741	-13.74434
Оценка максимального правдоподобия	0.03174	-0.03341	-0.00368
Усеченное среднее (уровень 0.05)	-0.02147	-0.00253	-0.03016
Усеченное среднее (уровень 0.10)	-0.01677	-0.00657	-0.02489
Усеченное среднее (уровень 0.15)	-0.00914	-0.01633	-0.02150
Обобщенные радикальные оценки (параметр 0.1)	-0.00301	-0.02091	-0.01293
Обобщенные радикальные оценки (параметр 0.5)	0.00878	-0.03506	-0.00996
Обобщенные радикальные оценки (параметр 1)	0.01954	-0.04897	-0.01044


Все функции на одном графике
avg_0
median
avg_0.05
avg_0.10
avg_0.15
OMP
rad_0.1
rad_0.5
rad_1.0
10
5
0
−5
−5
0
5
10

Распределение с асимметричным засорением ($\theta = 1.5, \lambda = 3, \varepsilon = 0.25$)



Номер выборки	1	2	3
Среднее арифметическое	0.41527	0.39734	0.43892
Выборочная медиана	0.17690	0.15426	0.15784
Дисперсия	3.62617	4.28311	4.66174
Ассиметрия	1.06094	0.81454	0.91784
Эксцесс	6.20427	6.21311	7.84995
Максимальное значение	9.30508	12.27060	16.07702
Минимальное значение	-6.84261	-9.19362	-9.46030
Оценка максимального правдоподобия	0.22169	0.19876	0.17634
Усеченное среднее (уровень 0.05)	0.31104	0.31068	0.34990
Усеченное среднее (уровень 0.10)	0.25718	0.24184	0.27889
Усеченное среднее (уровень 0.15)	0.22876	0.20764	0.23385
Обобщенные радикальные оценки (параметр 0.1)	0.05126	0.00861	0.00583
Обобщенные радикальные оценки (параметр 0.5)	0.05523	0.02251	0.00069
Обобщенные радикальные оценки (параметр 1)	0.05986	0.03873	-0.00420


Все функции на одном графике
avg_0
median
avg_0.05
avg_0.10
avg_0.15
OMP
rad_0.1
rad_0.5
rad_1.0
10
5
0
−5
−5
0
5
10

5. Выводы

Для выборок с чистым распределением все оценки дают примерно одинаковый результат, т.к. наблюдения распределены симметрично относительно параметра сдвига.

Для выборок с симметричным засорением все оценки так же показывают себя хорошо: «выбросы» справа и слева от параметра сдвига основного распределения компенсируют друг друга при оценке параметра сдвига (из графиков влияния оценок видно, что справа и слева от сдвига функции вклад от наблюдений равен по модулю, но противоположен по знаку)

Для выборок с несимметричным засорением хуже всех показали себя выборочное среднее и ОМП, наиболее точная оценка сдвига получена обобщенной радикальной оценкой. Этот результат объясняется тем, что «выбросы» сосредоточены правее параметра сдвига, а в этой области у разных оценок разные функции влияния: у среднего арифметического функция влияния имеет наибольшее значение, а у обобщенных радикальных оценок напротив – в области «выбросов» функция влияния наименьшая по модулю.

6. Текст программы

```
# influence.py
import math
import typing as t

import numpy as np
from scipy import integrate

from src.density import density
from src.derivatives import derivative_for_influence_omp, deriva-
tive_for_influence_rad


def influence_omp(
     x: float,
    nu: float,
     k: float,
) -> float:
    v, err = integrate.quad(
        derivative_for_influence_omp,
        -np.inf,
        np.inf,
        args=(
            nu,
            k,
        ),
    )

    return x / v if abs(x) <= k else k * math.copysign(1, x) / v
```

```
def influence_rad(
     x: float,
    nu: float,
     k: float,
    delta: float,
) -> float:
    v, err = integrate.quad(
        derivative_for_influence_rad,
        -np.inf,
        np.inf,
        args=(
            k,
            delta,
        ),
    )
    v *= (1 - nu) / (math.sqrt(2 * math.pi))

    if abs(x) <= k:
        return (math.exp(-delta * x * x / 2) * x) / v

    return (math.exp(delta * (0.5 * k * k - k * abs(x))) * k *
math.copysign(1, x)) / v


def influence_function(
    data: t.List[float],
    n_steps: int,
    nu: float,
    k: float,
):
    avg_infl = []
    mediana_infl = []
    avg005_infl = []
    avg010_infl = []
    avg015_infl = []
    OMP_infl = []
    rad01_infl = []
    rad05_infl = []
    rad10_infl = []
    h_d = []

    # start = min(data) - 1
    # h = (max(data) + min(data)) / n_steps
    start = -8 # min(testing_data)-1.0
    # h = (max(testing_data) + 1.0 - (min(testing_data)-1.0)) / N2
    h = (13 - (-8)) / n_steps

    for i in range(n_steps):
        y = start + i * h

        h_d.append(y)
```

```
        avg_infl.append(y)
        mediana_infl.append(math.copysign(1, y) / (2 * density(0, nu=nu,
k=k)))

        tmp = 1 / (1 - 2 * 0.05)  # a = 0.05; k = 1.65

        if y <= -1.65:
            tmp *= -1.65
        else:
            if y >= 1.65:
                tmp *= 1.65
            else:
                tmp *= y
        avg005_infl.append(tmp)

        tmp = 1 / (1 - 2 * 0.1) # a = 0.1; k = 1.28

        if y <= -1.28:
            tmp *= -1.28
        else:
            if y >= 1.28:
                tmp *= 1.28
            else:
                tmp *= y
        avg010_infl.append(tmp)

        tmp = 1 / (1 - 2 * 0.15) # a = 0.15; k = 1.04

        if y <= -1.04:
            tmp *= -1.04
        else:
            if y >= 1.04:
                tmp *= 1.04
            else:
                tmp *= y
        avg015_infl.append(tmp)

        OMP_infl.append(influence_omp(
            y,
            nu,
            k,
        ))
        rad01_infl.append(influence_rad(y, nu, k, 0.1))
        rad05_infl.append(influence_rad(y, nu, k, 0.5))
        rad10_infl.append(influence_rad(y, nu, k, 1))

    return (
        avg_infl,
        mediana_infl,
        avg005_infl,
        avg010_infl,
```

```
        avg015_infl,
        OMP_infl,
        rad01_infl,
        rad05_infl,
        rad10_infl,
        h_d,
    )

# generators.py
import math
import random
import typing as t

from src.density import density, noisy_density
from src.utils import calc_p


def generate_random_value(
    *,
    k: float,
    p: float,
) -> float:
    r = random.uniform(0, 1)

    if r >= p:
        x = random.normalvariate(0, 1)

        while -k > x or x > k:
            x = random.normalvariate(0, 1)

        return x

    r2 = random.uniform(0, 1)
    x = k - math.log(r2) / k

    return x if r < p / 2 else -x


def generate_clean_data(
    n: int,
    nu: float,
    k: float,
) -> t.Tuple[t.List[float], t.List[float]]:
    p = calc_p(
        nu=nu,
        k=k,
    )
    clean_data = sorted([generate_random_value(k=k, p=p) for _ in
range(n)])
    data_density = [density(val, nu=nu, k=k) for val in clean_data]
```

```
    return (
        clean_data,
        data_density,
    )


def generate_noise(
    n: int,
    nu: float,
    k: float,
    theta: float,
    lmdb: float,
) -> t.Tuple[t.List[float], t.List[float]]:
    p = calc_p(
        nu=nu,
        k=k,
    )

    noise = sorted([theta + lmdb * generate_random_value(k=k, p=p) for _
in range(n)])
    noise_density = [density((val - theta) / lmdb, nu=nu, k=k) / lmdb for
val in noise]

    return (
        noise,
        noise_density,
    )


def generate_noisy_data(
    n: int,
    nu: float,
    k: float,
    noisy_nu: float,
    noisy_k: float,
    theta: float,
    lmdb: float,
    eps: float,
) -> t.Tuple[t.List[float], t.List[float]]:
    p = calc_p(
        nu=nu,
        k=k,
    )
    noisy_p = calc_p(
        nu=noisy_nu,
        k=noisy_k,
    )

    noisy_data = []

    for _ in range(n):
```

```
        r = random.uniform(
            0,
            1,
        )

        if r <= 1 - eps:
            noisy_data.append(generate_random_value(k=k, p=p))
        else:
            noisy_data.append(theta + lmdb * generate_ran-
dom_value(k=noisy_k, p=noisy_p))

    noisy_data.sort()

    noisy_data_density = [noisy_density(val, nu=nu, k=k, eps=eps,
theta=theta, lmdb=lmdb) for val in noisy_data]

    return (
        noisy_data,
        noisy_data_density,
    )
#  utils.py
import statistics
import typing as t
from pathlib import Path

from scipy import stats
from scipy.optimize import minimize

from src.const import Constants
from src.density import density
from src.losses import huber_loss


def calc_p(
    *,
    nu: float,
    k: float,
) -> float:
    return 2 * (1 - nu) / k * density(k, nu=nu, k=k)


def describe_distribution(
    data: t.List[float],
    nu: t.Optional[float] = None,
    k: t.Optional[float] = None,
    eps: t.Optional[float] = None,
    noise_mean: t.Optional[float] = None,
    noise_variance: t.Optional[float] = None,
    *,
    out_path: Path,
    with_estimations: bool = False,
```

```
) -> None:
    mean = statistics.mean(data)

    with open(out_path, 'w', encoding='utf-8') as out:
        out.write('Статистика,Значение\n')

        print(f'Среднее:    {mean:.5f}')
        out.write(f'Среднее,{mean:.5f}\n')

        print(f'Медиана:    {statistics.median(data):.5f}')
        out.write(f'Медиана,{statistics.median(data):.5f}\n')

        print(f'Дисперсия: {statistics.variance(data, mean):.5f}')
        out.write(f'Дисперсия,{statistics.variance(data, mean):.5f}\n')

        print(f'Асимметрия: {stats.skew(data):.5f}')
        out.write(f'Асимметрия,{stats.skew(data):.5f}\n')

        print(f'Эксцесс:    {stats.kurtosis(data, fisher=False):.5f}')
        out.write(f'Эксцесс,{stats.kurtosis(data, fisher=False):.5f}\n')

        print(f'Макс:       {max(data):.5f}')
        out.write(f'Макс,{max(data):.5f}\n')

        print(f'Мин:        {min(data):.5f}')
        out.write(f'Мин,{min(data):.5f}\n')

        for pc in Constants.proportion_cuts:
            print(f'Усеченное среднее (уровень усечения {pc}):
{stats.trim_mean(data, pc):.5f}')
            out.write(f'Усеченное среднее (уровень усечения
{pc}),{stats.trim_mean(data, pc):.5f}\n')

        print(f'ОМП: {stats.norm.fit(data)[0]:.5f}')
        out.write(f'ОМП,{stats.norm.fit(data)[0]:.5f}\n')

        if with_estimations:
            for delta in Constants.deltas:
                m_start = mean
                res = minimize(
                    huber_loss,
                    m_start,
                    args=(
                        data,
                        nu,
                        k,
                        delta,
                        eps,
                        noise_mean,
                        noise_variance,
                    ),
```

```
                    method='nelder-mead',
                    # options={'maxiter': 1},
                )
                cur_q = huber_loss(
                    res.x[0],
                    data,
                    nu,
                    k,
                    delta,
                    eps,
                    noise_mean,
                    noise_variance,
                )

                print(f'Обобщенная радикальная оценка (delta = {delta}, Q
= {cur_q}): {res.x[0]:.5f}')
                out.write(f'Обобщенная радикальная оценка (delta =
{delta}, Q = {cur_q}),{res.x[0]:.5f}\n')

# density.py
import math


def density(
    x: float,
    *,
    nu: float,
    k: float,
) -> float:
    if abs(x) <= k:
        return (1 - nu) / (math.sqrt(2 * math.pi)) * math.exp(-x * x / 2)

    return (1 - nu) / (math.sqrt(2 * math.pi)) * math.exp(1 / 2 * k * k -
k * abs(x))


def noisy_density(
    x: float,
    *,
    nu: float,
    k: float,
    eps: float,
    theta: float,
    lmdb: float,
) -> float:
    return (1 - eps) * density(x, nu=nu, k=k) + eps * density((x - theta)
/ lmdb, nu=nu, k=k) / lmdb

# losses.py
import typing as t
```

```
from src.density import density, noisy_density


def huber_loss(
    theta: float,
    data: t.List[float],
    nu: float,
    k: float,
    delta: float,
    eps: float,
    noise_mean: float,
    noise_variance: float,
) -> float:
    res_q = 0

    if eps is None:
        for val in data:
            res_q += -(1 / density(0, nu=nu, k=k) ** delta) * (den-
sity(val - theta, nu=nu, k=k) ** delta)
    else:
        for val in data:
            res_q += -(1 / noisy_density(0, nu=nu, k=k, eps=eps,
theta=noise_mean, lmdb=noise_variance) ** delta) * (
                      noisy_density(val - theta, nu=nu, k=k, eps=eps,
theta=noise_mean, lmdb=noise_variance) ** delta)

    return res_q

# const.py
from pathlib import Path

CWD = Path(__file__).parents[1]


class Constants:
    nu = 0.05
    k = 1.398

    noisy_nu = 0.02
    noisy_k = 1.717

    proportion_cuts = [
        0.05,
        0.1,
        0.15,
    ]
    deltas = [
        0.1,
        0.5,
        1,
    ]
```

```
    n_resamples = 3

    sym_eps = 0.1
    asym_eps = 0.25

# derivatives.py
import math


def derivative_for_influence_omp(
    x: float,
    nu: float,
    k: float,
) -> float:
    if abs(x) <= k:
        return (1 - nu) / (math.sqrt(2 * math.pi)) * math.exp(-x * x / 2)
* (x * x)

    return (1 - nu) / (math.sqrt(2 * math.pi)) * math.exp(0.5 * k * k - k
* abs(x)) * (k * k)


def derivative_for_influence_rad(
    x: float,
    k: float,
    delta: float,
) -> float:
    if abs(x) <= k:
        return math.exp(-(delta + 1) * x * x / 2) * (x * x)

    return math.exp((delta + 1) * (0.5 * k * k - k * abs(x))) * (k * k)
```