МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической и прикладной информатики

ПРАКТИЧЕСКОЕ ЗАДАНИЕ №1

по дисциплине: Современные проблемы прикладной математики и наукоемкого программного обеспечения

на тему: Устойчивые методы оценивания параметров статистических моделей

Вариант 4б

Факультет: ФПМИ

Группа: ПММ-21

Выполнил: Сухих А.С.

Проверил: д.т.н Лисицин Д.В.

Дата выполнения: 26.12.22

Новосибирск 2022

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

Задание:

Косинусно-экспоненциальное распределение со значением параметра формы v=0.1. Уровень выполнения №2 (средний).

1. Разработать программу, которая реализует:

– генерацию наборов данных с чистым распределением и засоренным распределением;

– вычисление среднего арифметического, выборочной медианы, дисперсии, коэффициентов асимметрии и эксцесса;

– вычисление усеченного среднего (с уровнями 0.05, 0.10, 0.15), обобщенных радикальных оценок (со значениями параметра 0.1, 0.5, 1) и оценки максимального правдоподобия параметра сдвига модельного распределения;

2. Провести проверку генератора чистого распределения путем сравнения выборочных характеристик с их теоретическими значениями на выборках большого объема (N порядка 10^5–10^7);

3. Для выборок с разными видами распределений вычислить оценки параметра сдвига. Использовать выборки, имеющие следующие виды распределений:

– чистое распределение;

– засоренное распределение с симметричным засорением;

– засоренное распределение с асимметричным засорением.

Рекомендуемый уровень засорения $\varepsilon = 0.05 - 0.4$. Рекомендуемый объем выборки $N = 100 - 1000$. Сравнить устойчивость оценок для распределений указанных видов по их отклонению от истинного значения, сопоставить результаты сравнения со свойствами функций влияния оценок.

Ход работы:

1. Генерация распределения

Стандартное косинусно-экспоненциальное распределение имеет плотность

$$f(x,\nu)=\begin{cases} b_2\cos^2(b_1 x), & |x|\le 1 \\ b_3\exp(-b_4|x|), & |x|>1 \end{cases}$$

где $0\le\nu<1$, константы $b_1, \ldots, b_4$ – неотрицательные числа, которые зависят от ν и определяются из уравнений.

$$1+b_1\operatorname{tg} b_1-\frac{\cos^2 b_1}{\nu}=0,\quad b_4=2b_1\operatorname{tg} b_1,\quad b_3=\frac{\nu b_4}{2}\exp b_4,\quad b_2=\frac{b_4}{2+b_4}.$$

Для текущего варианта определены параметры $v = 0.1$, $b_1 = 1.03$.

Для моделирования случайной величины можно использовать следующий алгоритм

Шаг 1. Сгенерировать псевдослучайное число r_1. Если $r_1\ge\nu$, перейти на шаг 2, иначе перейти на шаг 4.

Шаг 2. Сгенерировать псевдослучайные числа r_2, r_3 и вычислить $x_1=2r_2-1$.

Шаг 3. Если $r_3\le\cos^2(b_1 x)$, то x_1 – искомое псевдослучайное число, иначе перейти на шаг 2.

Шаг 4. Сгенерировать псевдослучайное число r_4, вычислить $x_2=1-\frac{\ln r_4}{b_4}$.

Шаг 5. Если $r_1<\nu/2$, то x_2 – искомое псевдослучайное число, иначе искомым псевдослучайным числом является $-x_2$.

Проверим реализованный алгоритм генерации случайной величины на выборке объемом $N = 10^7$. Сравним значения дисперсии и коэффициентов

эксцесса, заданных для параметра формы $\nu = 0.1$ и вычисленного на основании данных из выборки.

Параметр	Теоретическое значение	Вычисленное
Дисперсия	0.39	0.38838
Коэффициент эксцесса	3.36	3.33043

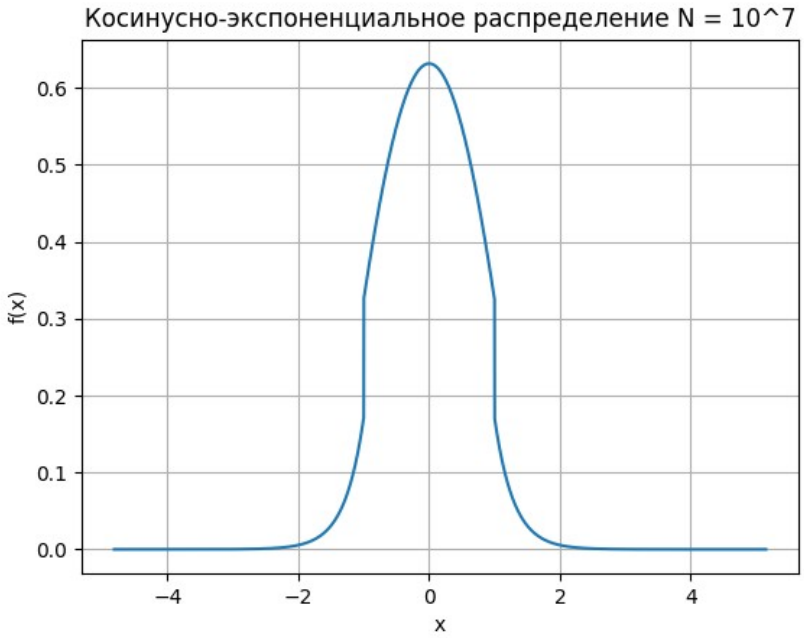

Рисунок 1 — функция плотности чистого распределения

2. Оценки параметра сдвига по чистому распределению

Объем выборки $N = 1000$. Оценка производилась по трем выборкам. Оценивается параметр сдвига при стандартном распределении (параметр сдвига равен 0 и масштаба равен 1).

Оценка		Выборка 1	Выборка 2	Выборка 3
Среднее арифметическое		-0.00394	0.022376	-0.019455
Медиана		0.01311	0.014106	-0.01391
Дисперсия		0.40573	0.3534	0.38583
Асимметрия		0.03658	-0.0124	-0.06316
Эксцесс		3.31943	3.07348	3.32355
ОМП		-0.00552	0.02433	-0.0169
Усеченное среднее	0.05	-0.0031	0.02321	-0.016764
	0.1	-0.00214	0.023	-0.01783
	0.15	-0.00178	0.020807	-0.01874
Обобщенные	0.1	-0.00434	0.024334	-0.0169

радикальные оценки	0.5	-0.00286	0.024683	-0.01684
	1	-0.00286	0.024334	-0.01684

3. Оценки параметра сдвига по симметрично засоренному распределению

Объем выборки N = 1000. Параметр сдвига у чистого и засоряющего распределения равны нулю. Параметр масштаба у засоряющего распределения равен 3. Уровень засорения 0.2

Оценка		Выборка 1	Выборка 2	Выборка 3
Среднее арифметическое		0.02847	0.04912	0.01136
Медиана		0.02175	0.02374	0.00473
Дисперсия		1.13956	0.99152	0.8704
Асимметрия		0.13851	0.17983	0.36343
Эксцесс		7.39095	7.0224	7.48752
ОМП		0.02568	0.05372	0.00907
Усеченное среднее	0.05	0.01778	0.04024	0.00232
	0.1	0.0196	0.03426	0.00607
	0.15	0.02269	0.02941	0.00926
Обобщенные радикальные оценки	0.1	0.02571	0.05019	0.00909
	0.5	0.02569	0.05016	0.00909
	1	0.0258	0.05015	0.01533

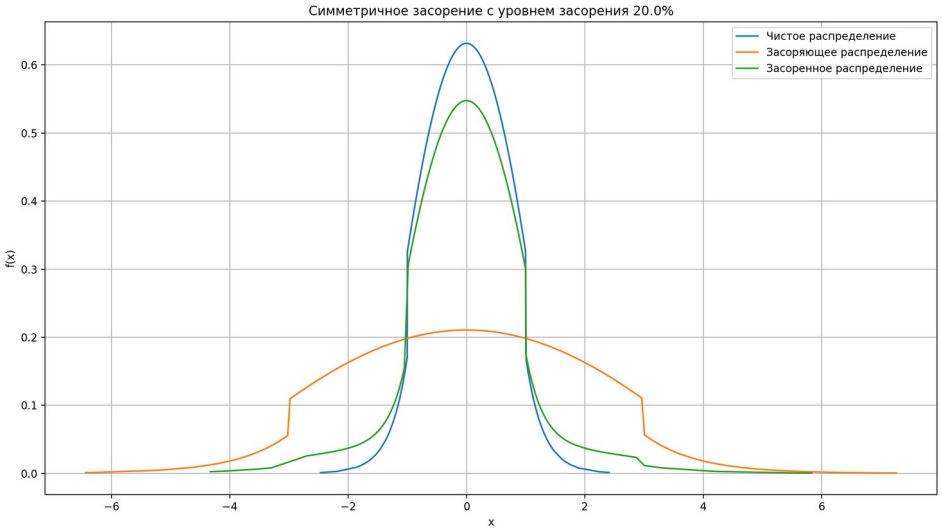

Рисунок 2 — функция плотности симметрично засоренного распределения

3. Оценки параметра сдвига по асимметрично засоренному распределению

Объем выборки $N = 1000$. Параметр масштаба у чистого и засоряющего распределения равны 1. Параметр сдвига у засоряющего распределения равен 2 стандартным отклонениям. Уровень засорения 0.2

Оценка		Выборка 1	Выборка 2	Выборка 3
Среднее арифметическое		0.2618	0.27417	0.17381
Медиана		0.23329	0.22876	0.09631
Дисперсия		0.5916	0.66495	0.59193
Асимметрия		0.2885	0.32947	0.3494
Эксцесс		3.02551	3.72713	3.12301
ОМП		0.26061	0.29723	0.18077
Усеченное среднее	0.05	0.24829	0.26105	0.1563
	0.1	0.2362	0.24907	0.14243
	0.15	0.2291	0.23755	0.1326
Обобщенные радикальные оценки	0.1	0.26057	0.28263	0.17313
	0.5	0.20596	0.25992	0.17306
	1	0.17682	0.25992	0.17306

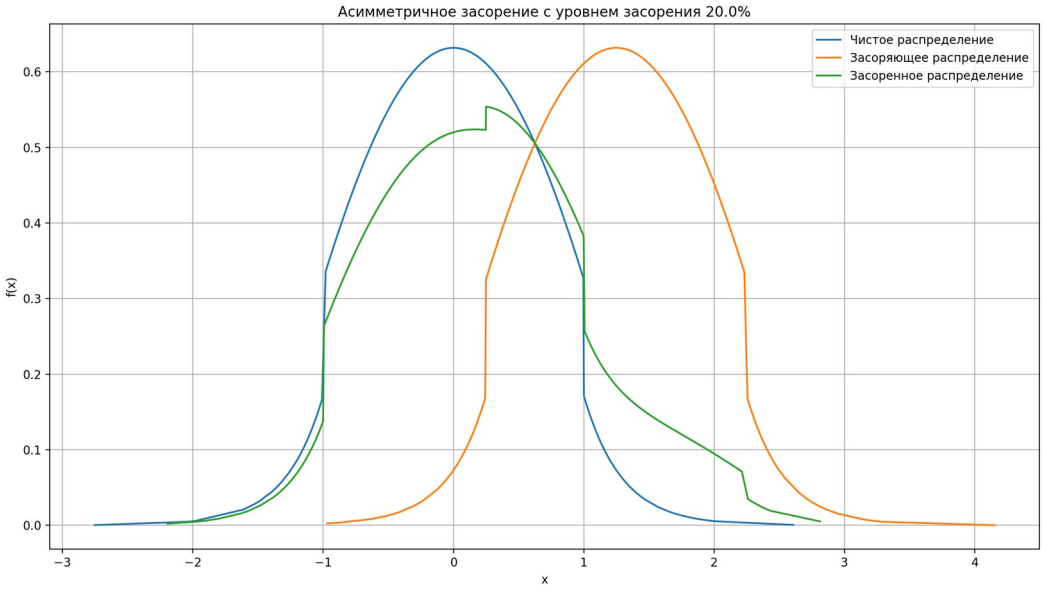

Рисунок 3 — функция плотности асимметрично засоренного распределения

4. Функции влияния примененных оценок

На рисунке 4 приведены графики функций влияния среднего арифметического, выборочной медианы и усеченного среднего с различными значениями параметров

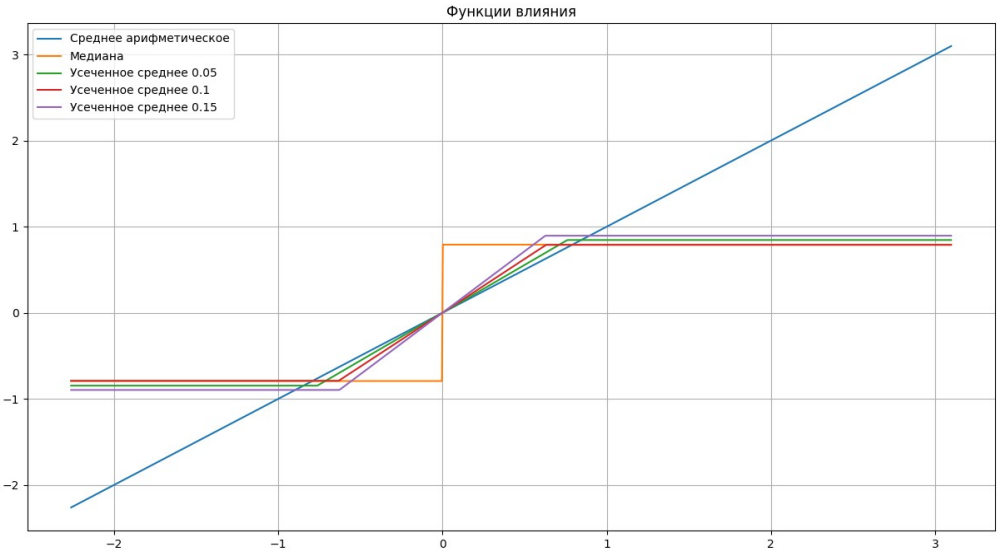

Рисунок 4 — функции влияния примененных оценок

Не удалось построить функции влияния для оценки максимального правдоподобия и обобщенных радикальных оценок, так как в процессе вычисления функций влияния плотность вероятность в некоторой точке принимало значение 0 и происходило деление на ноль.

Вывод: В ходе лабораторной работы была разработана программа для генерации наборов данных согласно закону косинусно-экспоненциального распределения и для вычисления оценок параметров сдвига. Расхождение между значениями дисперсии и коэффициента эксцесса на большом объеме выборки и теоретическим незначительно (в пределах сотых).

При оценке параметра сдвига по чистому распределению присутствуют отклонения от истинного значения 0. Наиболее точными оценками оказались усеченное среднее с уровнем усечения 0,15 и обобщенная радикальная оценка с параметром робастности 1. Менее точной показала себя оценка максимального правдоподобия и среднее арифметическое.

При оценке параметра по симметрично засоренному распределению можно увидеть заметный рост коэффициента эксцесса, характеризующего «тяжесть» хвостов распределения. Наиболее точной оценкой в определении параметра сдвига при симметричном засорении оказалась выборочная медиана. Причину можно наглядно увидеть по функции влияния выборочной медианы — засоренное распределение имеет больше выбросов на существенном отдалении от математического ожидания, чем вблизи к нему. А по функции влияния медианы видно, что вблизи наибольшее влияние на медиану оказывают выбросы вблизи мат.ожидания, но по мере удаления от него эта оценка имеет наименьшую чувствительность.

При оценке параметра по асимметрично засоренному распределению наблюдается существенный рост коэффициента асимметрии по сравнению с симметричным засорением, что связано с выбросами в области правого хвоста, вызванного засоряющим распределением. Наибольшую точность показали обобщенные радикальные оценки с параметром робастности 1 и медиана.

ПРИЛОЖЕНИЯ

```
import math
import random as rand
import numpy
import statistics

import numpy as np
import scipy.stats as stats
from scipy.optimize import minimize as scipy_minimize
from scipy import integrate


V_FORM = 0.1
B1 = 1.03
DISPERSION = 0.39
EXCESS = 3.36

B4 = 2 * B1 * math.tan(B1)
B3 = V_FORM * B4 / 2 * math.exp(B4)
B2 = B4 / (2 + B4)

TRIM_MEAN_PARAMETERS = (0.05, 0.1, 0.15)
ROBUSTNESS_PARAMETERS = (0.1, 0.5, 1)

STANDARD_SCALE = 1
STANDARD_SHIFT = 0

results_folder = "./estimation_results/"


def cos_exp_dist():
    """
    Генерация случайной величины на основе косинусно-эконенциального распределения
    :return: значение случайной величины
    """
    r1 = rand.random()
    if r1 < V_FORM:
        r4 = rand.random()
        x2 = 1 - math.log(r4) / B4
        if r1 < V_FORM / 2:
            return x2
        else:
            return -x2
    else:
        while True:
            r2 = rand.random()
            r3 = rand.random()
            x1 = 2 * r2 - 1
            if r3 <= math.pow(math.cos(B1 * x1), 2):
                return x1


def cos_exp_density(x):
    if abs(x) <= 1:
        return B2 * math.cos(B1 * x)
    else:
        return B3 * math.exp(-B4 * abs(x))


def cos_exp_density_derivative(x):
    if abs(x) <= 1:
        return -B1 * B2 * math.sin(2 * B1 * x)
    else:
        return -B4 * B3 * math.exp(-B4 * abs(x))


def mle_loss_function(shift, data, scale):
    q = 0
    for x in data:
        q += -math.log(cos_exp_density((x - shift) / scale))
    return q


def radical_loss_function(shift, data, scale, delta):
    q = 0
    for x in data:
```

```
        q += -1 / pow(cos_exp_density(0), delta) * pow(cos_exp_density((x - shift) / scale),
delta)
    return q


def generate_data(n: int, shift: float = STANDARD_SHIFT, scale: float = STANDARD_SCALE):
    data = [shift + scale * cos_exp_dist() for x in range(n)]
    data.sort()
    return data, [cos_exp_density((x - shift) / scale) / scale for x in data]


def generate_noisy_data(n: int, shift: float, scale, noise_level: float):
    if 0.5 <= noise_level <= 0:
        raise RuntimeError("Incorrect noise level")
    noisy_data = []
    for i in range(n):
        r = rand.random()
        if r <= 1 - noise_level:
            noisy_data.append(cos_exp_dist())
        else:
            noisy_data.append(shift + scale * cos_exp_dist())
    noisy_data.sort()
    noisy_density = [((1 - noise_level) * cos_exp_density(x) + noise_level * cos_exp_density((x -
shift) / scale) / scale) for x in noisy_data]
    return noisy_data, noisy_density


def calculate_estimations(data, scale, estimate_shift=False):
    estimations = dict()
    estimations['mean'] = statistics.mean(data)
    print(f"Среднее: {estimations['mean']}")
    estimations['median'] = statistics.median(data)
    print(f"Медиана: {estimations['median']}")
    estimations['variance'] = statistics.variance(data, estimations['mean'])
    print(f"Дисперсия: {estimations['variance']}")
    estimations['skewness'] = stats.skew(data)
    print(f"Коэффициент асимметрии: {estimations['skewness']}")
    estimations['kurtosis'] = stats.kurtosis(data, fisher=False)
    print(f"Коэффициент эксцесса: {estimations['kurtosis']}")


    estimations['trim_mean'] = []
    for param in TRIM_MEAN_PARAMETERS:
        estimations['trim_mean'].append(stats.trim_mean(data, param))
    print(f"Усеченное среднее: {estimations['trim_mean']}")

    if estimate_shift:
        result = scipy_minimize(mle_loss_function, estimations['mean'], (data, scale),
method='Nelder-Mead')
        if result.success:
            estimations['mle'] = result.x[0]
        else:
            print("MLE estimation error: ", result)
        print(f"ОМП: {estimations['mle']}")

        estimations['radical_estimates'] = list()
        for param in ROBUSTNESS_PARAMETERS:
            result = scipy_minimize(radical_loss_function, estimations['mean'], (data, scale,
param), method='Nelder-Mead')
            estimations['radical_estimates'].append(result.x[0])

        print(f"Обобщенные радикальные оценки: {estimations['radical_estimates']}")
    return estimations


def mle_integrate_function(x):
    return math.pow(cos_exp_density_derivative(x), 2) / cos_exp_density(x)

def rad_integrate_function(x, delta=1):
    return math.pow(cos_exp_density_derivative(x), 2) * pow(cos_exp_density(x), delta-1)

def calculate_influence_functions_data(data, shift, scale):
    mean_influence = [y - shift for y in data]
    median_influence = [scale * math.copysign(1, y - shift) / (2 * cos_exp_density(0)) for y in
data]

    trim_mean_influence = list()
    for param in TRIM_MEAN_PARAMETERS:
```

```
        k = numpy.quantile(generate_data(1000), (1-param))
        trim_param_inf = list()
        for y in data:
            value = (y - shift) / scale
            if value <= -k:
                inf = -k
            elif value >= k:
                inf = k
            else:
                inf = value
            inf *= 1 / (1 - 2 * param)
            trim_param_inf.append(inf)

        trim_mean_influence.append(trim_param_inf)

    # v = integrate.quad(mle_integrate_function, -np.inf, np.inf)
    # mle_influence = [(-scale * cos_exp_density_derivative((y-shift)/scale)
    #                   / cos_exp_density((y-shift)/scale))
    #                  / v[0] for y in data]

    # v = integrate.quad(rad_integrate_function, -np.inf, np.inf,
args=(ROBUSTNESS_PARAMETERS[0],))
    # rad_influence = [-scale * cos_exp_density_derivative((y - shift) / scale)
    #        * pow(cos_exp_density((y - shift) / scale), ROBUSTNESS_PARAMETERS[0]-1)
    #        / v[0] for y in data]

    return mean_influence, median_influence, trim_mean_influence

def estimations_to_file(estimations: dict, output_file: str):
    with open(results_folder + output_file, 'w') as out_file:
        for estimation in estimations:
            if estimation in ('trim_mean', 'radical_estimates'):
                out_file.write(f"{estimation}:\n")
                for i, param in enumerate(TRIM_MEAN_PARAMETERS if estimation == 'trim_mean' else
ROBUSTNESS_PARAMETERS):
                    out_file.write(f"{param} -> {estimations[estimation][i]}; ")
                out_file.write('\n')
            else:
                out_file.write(f"{estimation}: {estimations[estimation]}\n".capitalize())
```