Санкт-Петербургский Политехнический Университет им. Петра Великого

Институт прикладной математики и механики

Кафедра прикладной математики

Лабораторная работа №4
Эмпирические функции и ядерные оценки

3 курс, группа 33631/2

Студент Д. А. Плаксин

Преподаватель Баженов А. Н.

Санкт-Петербург
2019 г.

Содержание

1. Список иллюстраций **3**

2. Постановка задачи **4**

3. Теория **4**

4. Реализация **4**

5. Результаты **5**
5.1. Эмпирические функции распределения 5
5.2. Ядерные функции 7

6. Выводы **14**

7. Список литературы **15**

8. Приложения **15**

1 Список иллюстраций

1 Эмпирическая функция для нормального стандартного распределения . . 5
2 Эмпирическая функция для стандартного распределения Лапласа 5
3 Эмпирическая функция для стандартного распределения Коши 6
4 Эмпирическая функция для распределения Пуассона 6
5 Эмпирическая функция для равномерного распределения 7
6 Ядерная функция плотности для нормального распределения, n = 20 . . . 7
7 Ядерная функция плотности для нормального распределения, n = 60 . . . 8
8 Ядерная функция плотности для нормального распределения, n = 100 . . 8
9 Ядерная функция плотности для распределения Лапласа, n = 20 9
10 Ядерная функция плотности для распределения Лапласа, n = 60 9
11 Ядерная функция плотности для распределения Лапласа, n = 100 10
12 Ядерная функция плотности для распределения Коши, n = 20 10
13 Ядерная функция плотности для распределения Коши, n = 60 11
14 Ядерная функция плотности для распределения Коши, n = 100 11
15 Ядерная функция плотности для распределения Пуассона, n = 20 12
16 Ядерная функция плотности для распределения Пуассона, n = 60 12
17 Ядерная функция плотности для распределения Пуассона, n = 100 13
18 Ядерная функция плотности для равномерного распределения, n = 20 . . 13
19 Ядерная функция плотности для равномерного распределения, n = 60 . . 14
20 Ядерная функция плотности для равномерного распределения, n = 100 . 14

2 Постановка задачи

Для, приведённых ниже, пяти распределений сгенерировать выборки объёмом 20, 60, 100, для каждой выборки построить эмпирические функции распределения и ядерные оценки плотности распределения на отрезке $[-4,4]$.

Распределения [4]:

1. Стандартное нормальное распределение
2. Стандартное распределение Коши
3. Распределение Лапласа с коэффициентом масштаба $\sqrt{2}$ и нулевым коэффициентом сдвига.
4. Равномерное распределение на отрезке $\left[-\sqrt{3},\sqrt{3}\right]$
5. Распределение Пуассона со значением матожидания равным двум.

3 Теория

Эмпирическая функция распределения [5], построенная по выборке $X = (X_1,\dots,X_n)$ есть случайная функция $F_n(y)$, определённая на $\mathbb{R}$:

$$F_n(y) = \sum_{i=1}^{n} I\left(X_i < y\right) \text{ где } I\left(X_i < y\right) = \begin{cases} 1, & X_i < y \\ 0, & \text{иначе} \end{cases} \tag{1}$$

$X = (X_1,\dots,X_n)$ есть одномерная выборка одинаково распределённых элементов, с плотностью распределения f.

Ядерная оценка плотности [6]:

$$f_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) \tag{2}$$

где K является ядром, а $h > 0$ является сглаживающим параметром, и называется шириной полосы.

В данной работе в качестве ядра была выбрана плотность вероятности стандартного нормального распределения [7]:

$$K(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \tag{3}$$

4 Реализация

Для генерации выборки был использован *Python* 3.7: модуль *random* библиотеки *numpy* [1] для генерации случайных чисел с различными распределениями.

Обработка функций распределений была сделана с помощью модуля *scipy* [3].

5 Результаты

5.1 Эмпирические функции распределения

Рис. 1: Эмпирическая функция для нормального стандартного распределения

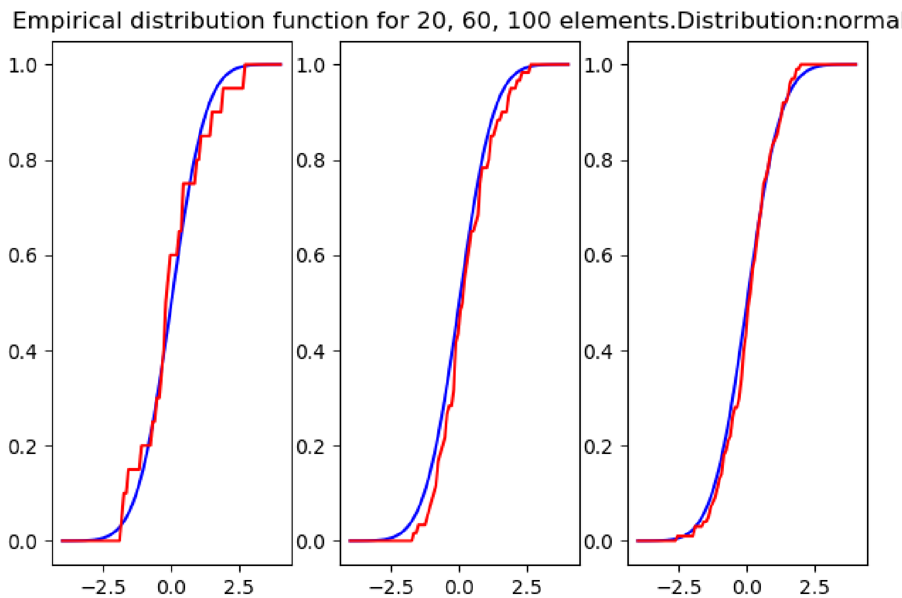


Рис. 2: Эмпирическая функция для стандартного распределения Лапласа

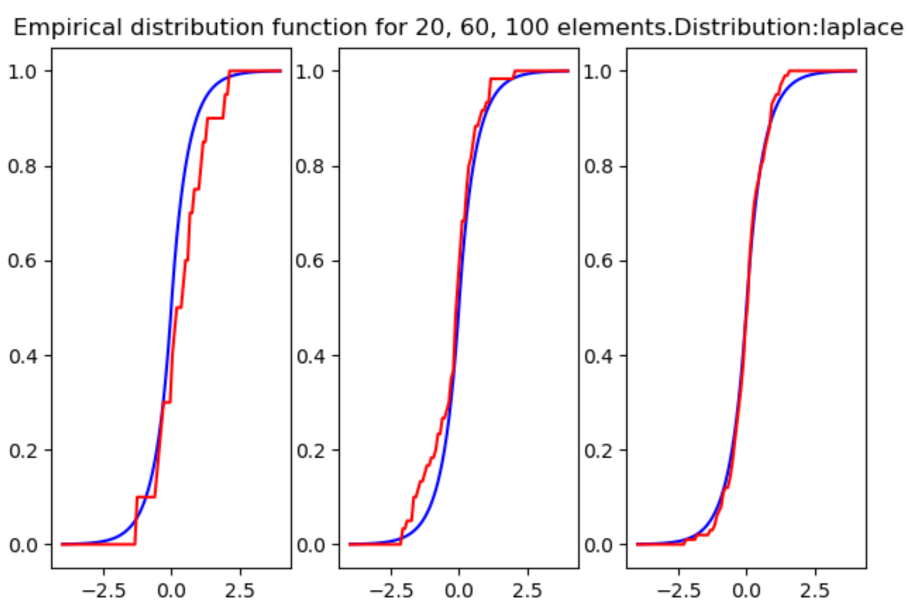


Рис. 3: Эмпирическая функция для стандартного распределения Коши

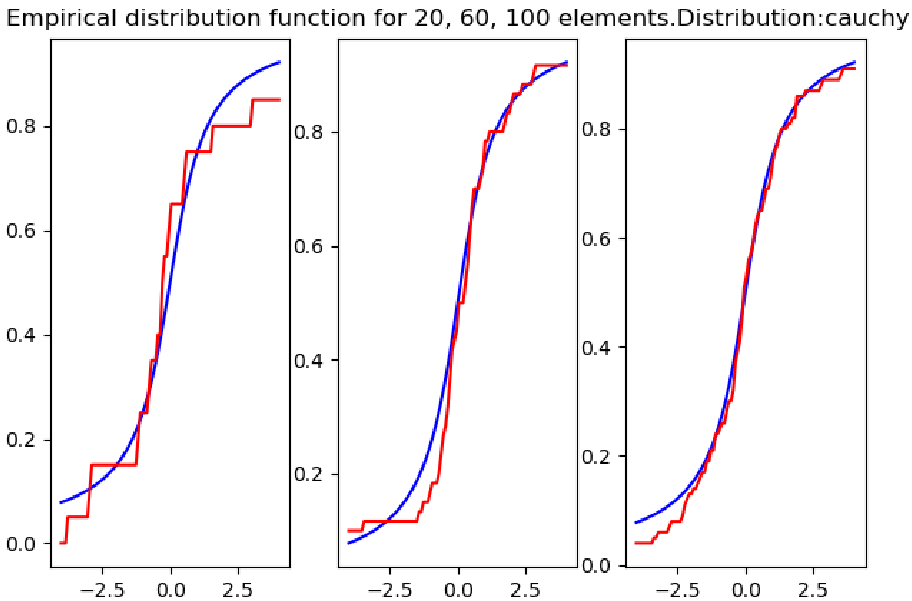


Рис. 4: Эмпирическая функция для распределения Пуассона

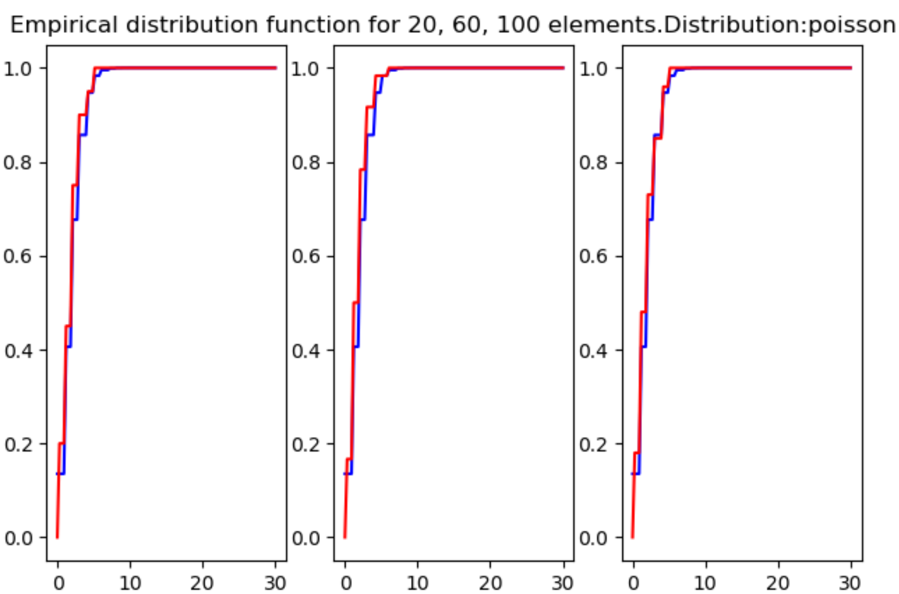
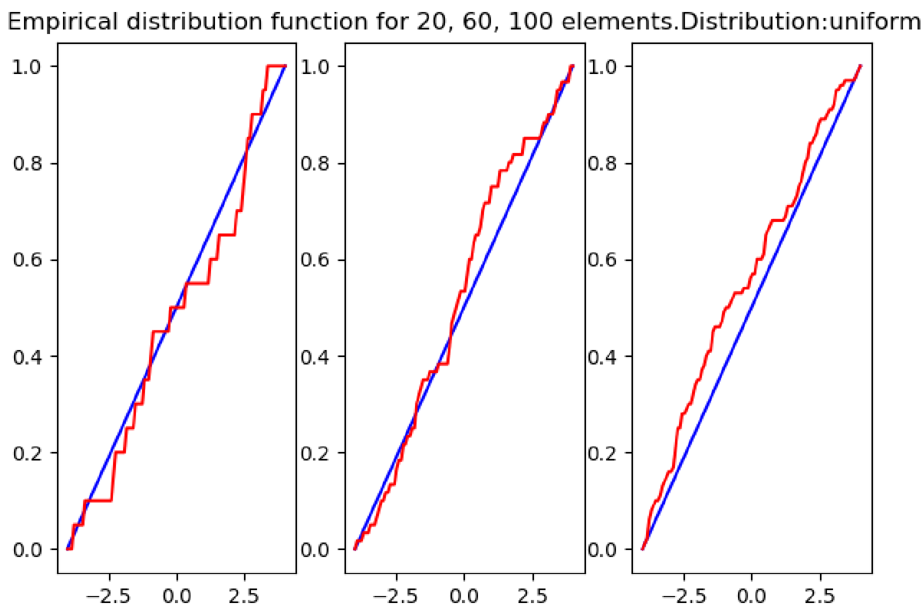


Рис. 5: Эмпирическая функция для равномерного распределения



5.2 Ядерные функции

Рис. 6: Ядерная функция плотности для нормального распределения, n = 20

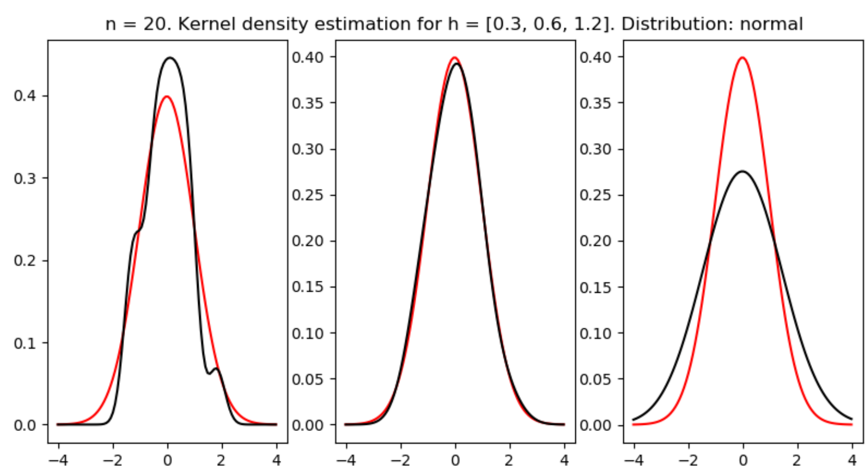


Рис. 7: Ядерная функция плотности для нормального распределения, n = 60

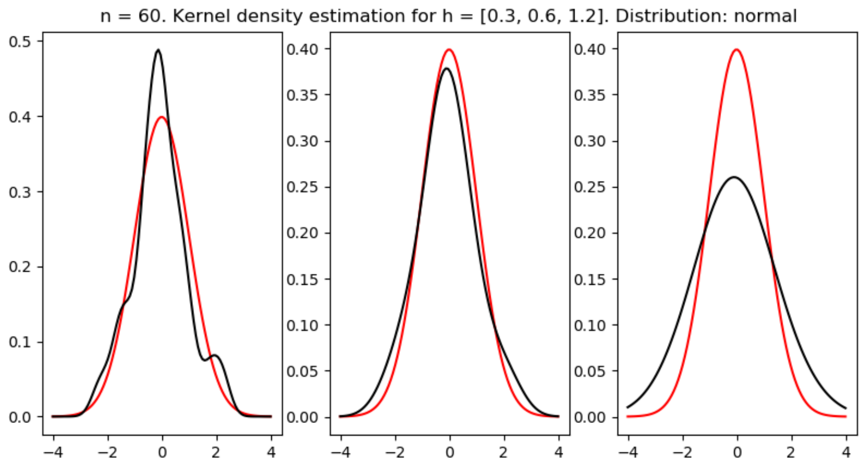


Рис. 8: Ядерная функция плотности для нормального распределения, n = 100

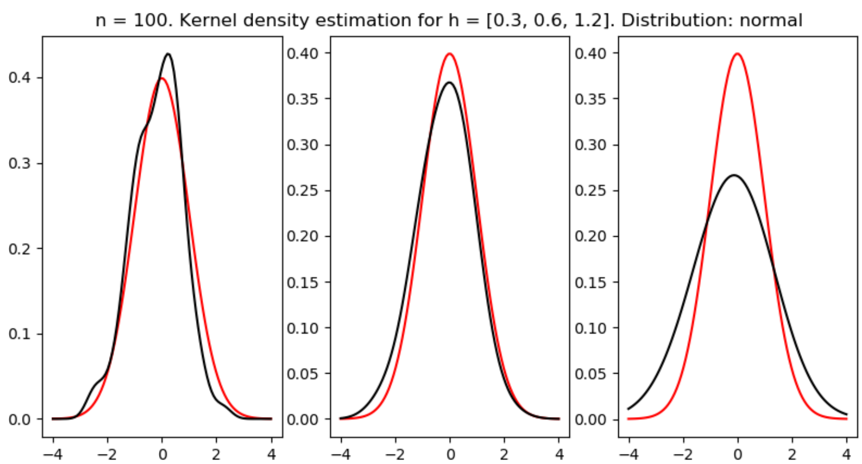


Рис. 9: Ядерная функция плотности для распределения Лапласа, n = 20

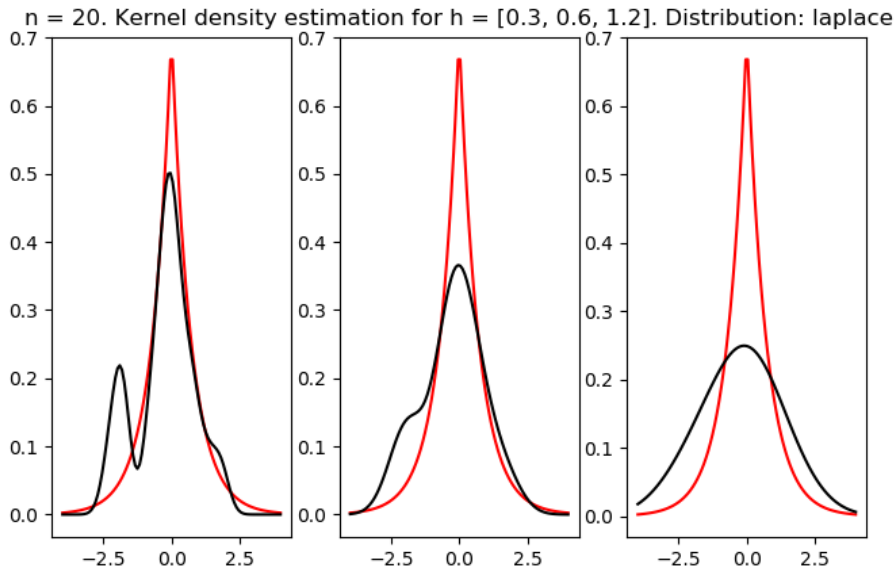


Рис. 10: Ядерная функция плотности для распределения Лапласа, n = 60

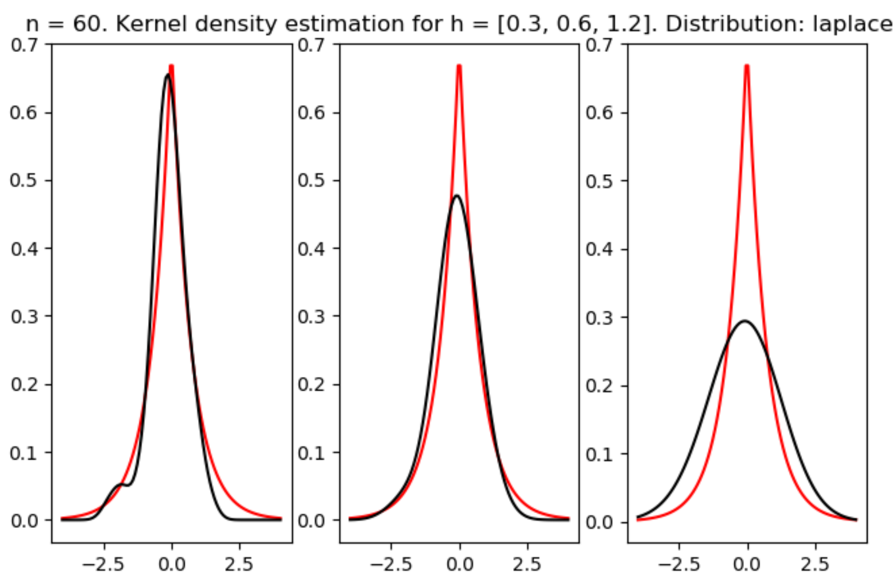


Рис. 11: Ядерная функция плотности для распределения Лапласа, n = 100

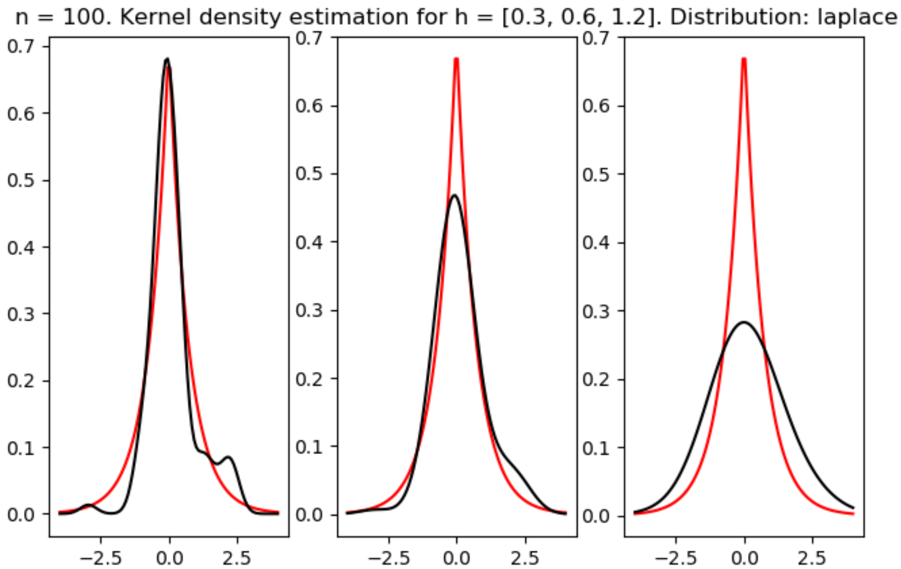


Рис. 12: Ядерная функция плотности для распределения Коши, n = 20

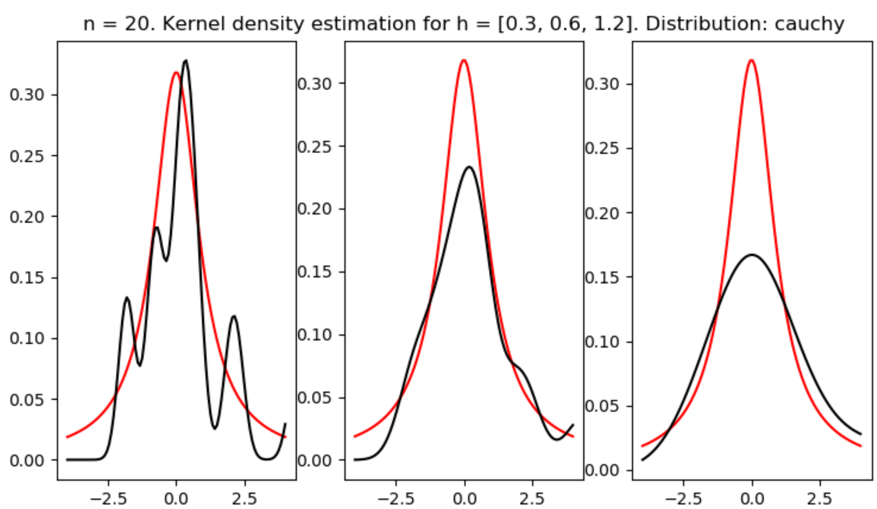


Рис. 13: Ядерная функция плотности для распределения Коши, n = 60

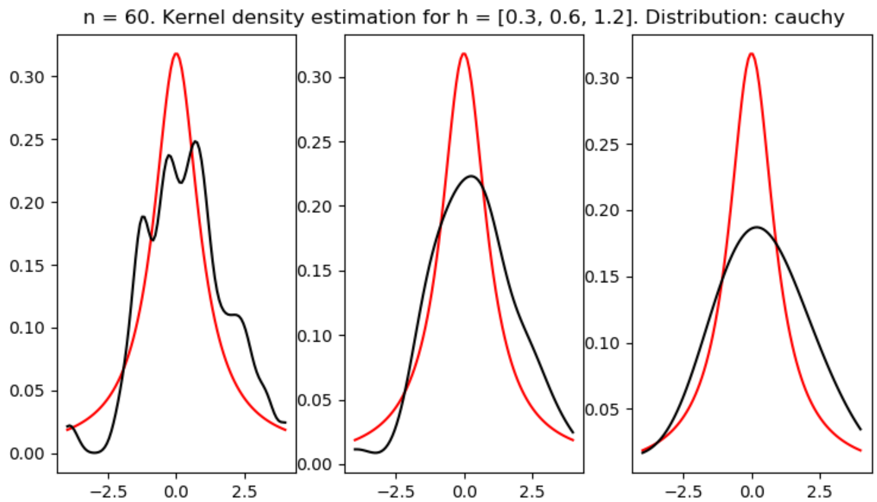


Рис. 14: Ядерная функция плотности для распределения Коши, n = 100

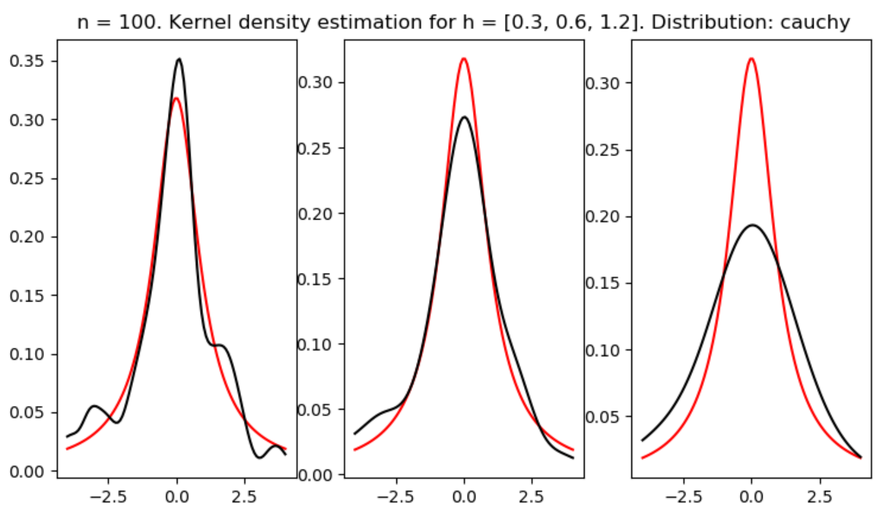


Рис. 15: Ядерная функция плотности для распределения Пуассона, n = 20

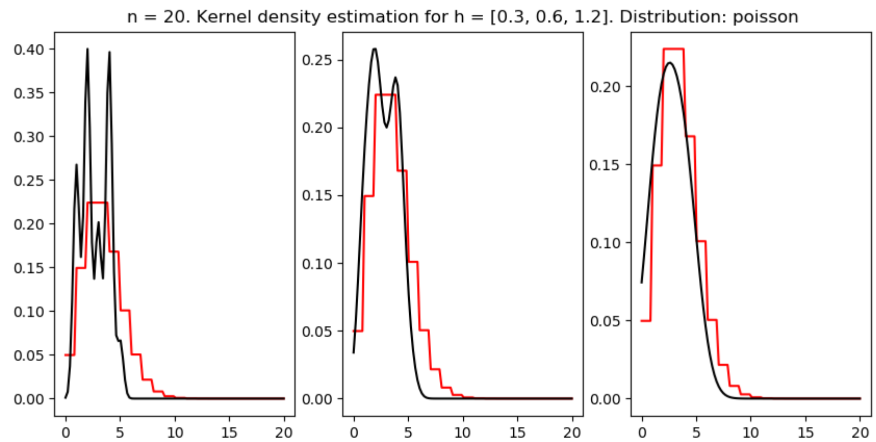


Рис. 16: Ядерная функция плотности для распределения Пуассона, n = 60

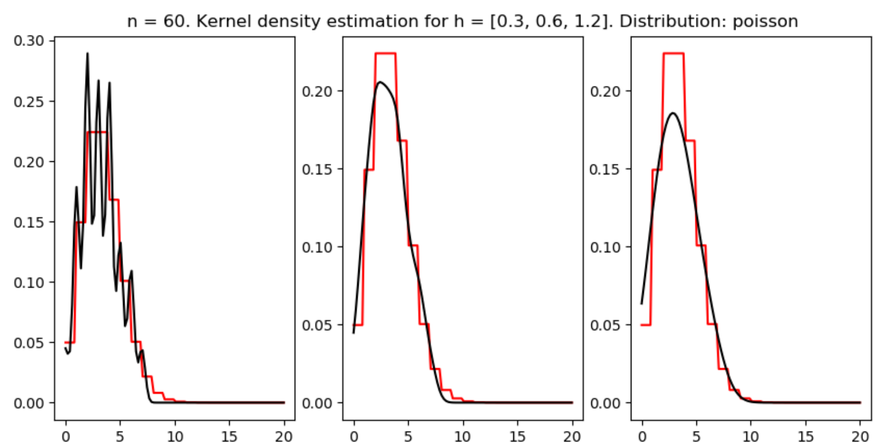


Рис. 17: Ядерная функция плотности для распределения Пуассона, n = 100

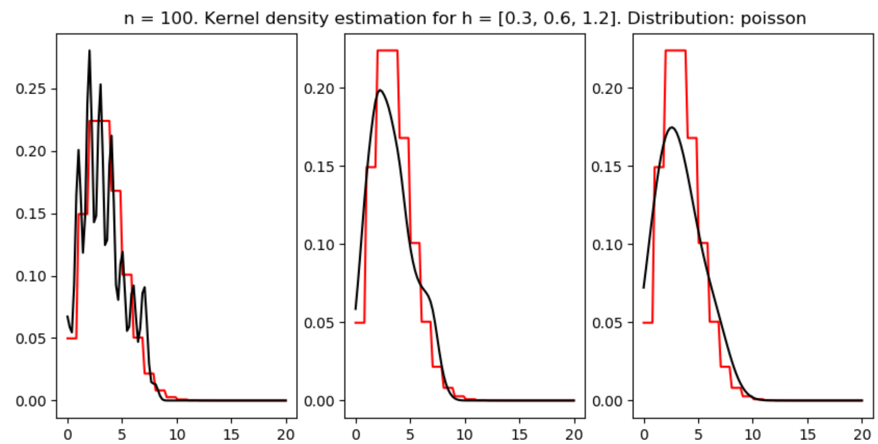


Рис. 18: Ядерная функция плотности для равномерного распределения, n = 20

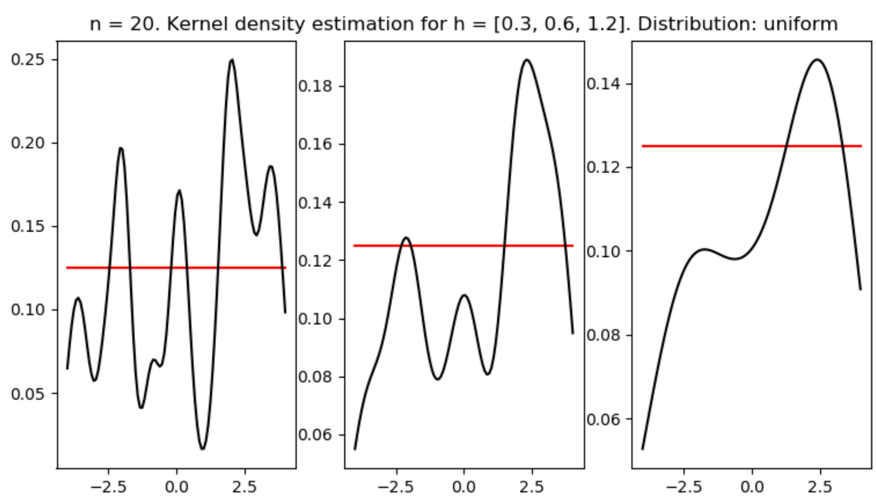


Рис. 19: Ядерная функция плотности для равномерного распределения, n = 60

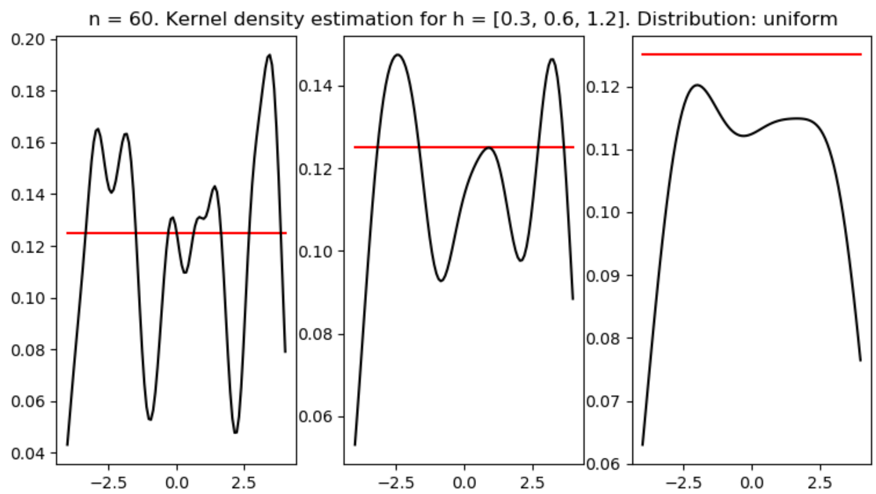
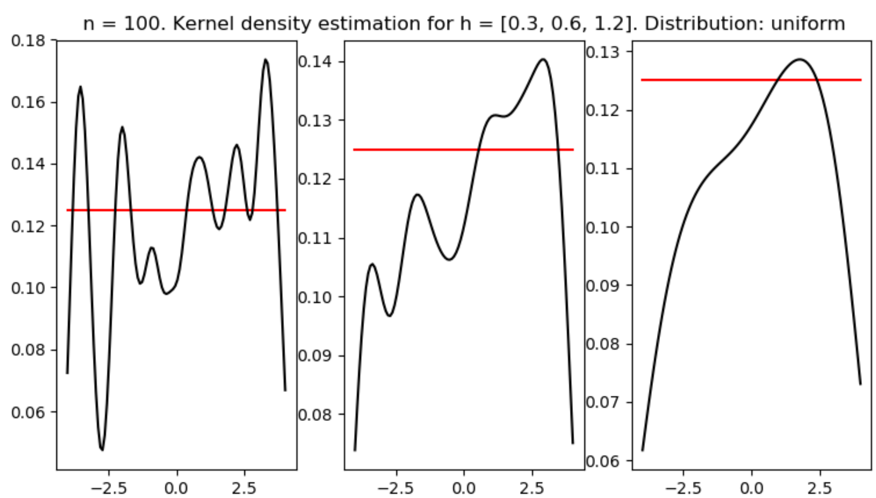


Рис. 20: Ядерная функция плотности для равномерного распределения, n = 100



6 Выводы

Эмпирическая функция лучше приближает эталонную функцию на больших выборках.

Наилучшее приближение функции распределения ядерной функции получено при наибольшей ширине окна. При фиксированной ширине окна точнее приблизить функцию распределения позволяет увеличение выборки.

7 Список литературы

[1] Модуль numpy - https://physics.susu.ru/vorontsov/language/numpy.html

[2] Модуль matplotlib - https://matplotlib.org/users/index.html

[3] Модуль scipy - https://docs.scipy.org/doc/scipy/reference/

[4] Формулы распределений - https://vk.com/doc184549949_491827451

[5] https://nsu.ru/mmf/tvims/chernova/ms/lec/node4.html

[6] https://www.mql5.com/ru/articles/396

[7] http://users.stat.umn.edu/ helwig/notes/den-Notes.pdf

8 Приложения

Код отчёта: https://github.com/MisterProper9000/MatStatLabs/blob/master/MatStatLab4/MatStatLab4.tex

Код лаборатрной: https://github.com/MisterProper9000/MatStatLabs/blob/master/MatStatLab4/MatStatLab4.py

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.stats import laplace
from scipy.stats import uniform
from scipy.stats import poisson


POISSON_PARAM = 2
UNIFORM_FRONT = 4


def normalized_distribution(x):
    return (1 / np.sqrt(2 * np.pi)) * np.exp(-x * x / 2)


def laplace_distribution(x):
    return (1 / np.sqrt(2)) * np.exp(-np.sqrt(2) * np.abs(x))


def uniform_distribution(x):
    flag = (x <= UNIFORM_FRONT)
    return 1 / (2 * UNIFORM_FRONT) * flag


def cauchy_distribution(x):
    return 1 / (np.pi * (1 + x * x))


def poisson_distribution(x):
    n = x.size
    res = []
    for i in range(n):
        res.append(0)

    for i in range(n):
        res[i] = (1 / np.power(np.e, 3)) / np.math.factorial(int(x[i])) * np.
    power(3, int(x[i]))
    return res

```

```
func_density_dict = {
    'normal':   normalized_distribution,
    'laplace':  laplace_distribution,
    'uniform':  uniform_distribution,
    'cauchy':   cauchy_distribution,
    'poisson':  poisson_distribution,
}


def cumulative_laplace(x):
    return laplace.cdf(x, 0, 1/np.sqrt(3))


def cumulative_poisson(x):
    return poisson.cdf(x, POISSON_PARAM)


def cumulative_cauchy(x):
    return (1/np.pi) * np.arctan(x) + 0.5


def cumulative_uniform(x):
    return uniform.cdf(x, -4, 8)


func_cumulative_dict = {
    'normal': norm.cdf,
    'laplace': cumulative_laplace,
    'uniform': cumulative_uniform,
    'cauchy': cumulative_cauchy,
    'poisson': cumulative_poisson
}


def generate_laplace(x):
    return np.random.laplace(0, 1/np.sqrt(3), x)


def generate_uniform(x):
    return np.random.uniform(-UNIFORM_FRONT, UNIFORM_FRONT, x)


def generate_poisson(x):
    return np.random.poisson(POISSON_PARAM, x)


generate_dict = {
    'normal':   np.random.standard_normal,
    'laplace':  generate_laplace,
    'uniform':  generate_uniform,
    'cauchy':   np.random.standard_cauchy,
    'poisson':  generate_poisson,
}


def empirical_function(sample, x):
    counter_array = []
    n = len(sample)
    m = len(x)
    for i in range(m):
        counter_array.append(0)

    for i in range(m):
        for j in range(n):
            if sample[j] < x[i]:
```

```
                counter_array[i] = counter_array[i] + 1
        counter_array[i] = counter_array[i] / n
    return counter_array


def kernel_function(sample, h, x):
    res_array = []
    n = len(sample)
    m = len(x)

    for i in range(m):
        res_array.append(0)

    for i in range(m):
        for j in range(n):
            res_array[i] += normalized_distribution((x[i] - sample[j]) / h)
        res_array[i] = res_array[i] / (n * h)

    return res_array


def draw_empirical(sample, func, sector):
    if sector == 3:
        plt.title('Empirical distribution function for 20, 60, 100 elements. Distribution:' + func)
    plt.subplot(130+sector)
    if func == 'poisson':
        xx = np.linspace(0, 30, 100)
    else:
        xx = np.linspace(-4, 4, 100)
    plt.plot(xx, func_cumulative_dict[func](xx), 'b')
    plt.plot(xx, empirical_function(sample, xx), 'r')


def research_empirical(distribution_type):
    plt.figure("distribution " + distribution_type)
    num = 20
    sector = 1
    for i in range(3):
        sample = generate_dict[distribution_type](num)
        draw_empirical(sample, distribution_type, sector)
        num += 40
        sector += 1
    plt.show()


def draw_kernel(sample, func, sector, h):
    if sector == 3:
        plt.title('n = ' + str(len(sample)) + '. Kernel density estimation for h = [0.3, 0.6, 1.2]. Distribution: ' + func)
    plt.subplot(130+sector)
    if func == 'poisson':
        xx = np.linspace(0, 20, 100)
    else:
        xx = np.linspace(-4, 4, 100)
    plt.plot(xx, func_density_dict[func](xx), 'r')
    plt.plot(xx, kernel_function(sample, h,  xx), 'black')


def research_kernel(distribution_type):
    num = 20
    sector = 1
    h = 0.3
    for i in range(3):
        sample = generate_dict[distribution_type](num)
        plt.figure("distribution " + distribution_type + ", sample size: " +
```

```
    str(len(sample)))
        for j in range(3):
            draw_kernel(sample, distribution_type, sector, h)
            sector += 1
            h *= 2
        sector = 1
        num += 40
        h = 0.3
    plt.show()

research_empirical('normal')
research_empirical('laplace')
research_empirical('uniform')
research_empirical('cauchy')
research_empirical('poisson')

research_kernel('normal')
research_kernel('laplace')
research_kernel('uniform')
research_kernel('cauchy')
#research_kernel('poisson')
```