# Лабораторная работа №2

## Методы оценки статических характеристик, связанных с распределением пользователей на плоскости

### Выполнил Лобов Михаил НПМмд-02

#jupyter nbconvert --to markdown Project.ipynb

## Импортируем библиотеки

import numpy as np  
import matplotlib.pyplot as plt  
from scipy import stats

# Задание №1

## Создаем выборки Экспоненциального распределения и Нормального распределения

exp\_sample\_100 = np.random.exponential(scale=1, size=100)  
exp\_sample\_1000 = np.random.exponential(scale=1, size=1000)  
  
  
  
norm\_sample\_100 = np.random.normal(loc=0, scale=1, size=100)  
norm\_sample\_1000 = np.random.normal(loc=0, scale=1, size=1000)

## Мат ожидание и дисперсия

1. Мы считаем мат ожидание и дисперсию для созданных распределений
2. Задаем теоретическое мат ожидание и дисперсию для теоретических значений у данных распределений
   1. Мат ожидание и дисперсия у экспоненциальног распределения будет 1 и 1 соответсвенно, потому что мы такие задали при генерации
   2. На практике случайные значнеия будут отличаться от теоретических, т.к. мы рассматриваем их на несовершенной выборке

### Все статистические вычисления выполняются с помощью встроенных функций numpy

def calculate\_mean\_variance(sample, distribution\_name):  
 sample\_mean = np.mean(sample)  
 sample\_variance = np.var(sample, ddof=1)   
  
 if distribution\_name == "exponential":  
 theoretical\_mean = 1   
 theoretical\_variance = 1   
 elif distribution\_name == "normal":  
 theoretical\_mean = 0   
 theoretical\_variance = 1   
  
   
 print(f"Математическое ожидание ({distribution\_name}): {theoretical\_mean}")  
 print(f"Наблюдаемое среднее ({distribution\_name}, n={len(sample)}): {sample\_mean:.4f}")  
 print(f"Теоретическая дисперсия ({distribution\_name}): {theoretical\_variance}")  
 print(f"Наблюдаемая дисперсия ({distribution\_name}, n={len(sample)}): {sample\_variance:.4f}")  
 print("")

## Вызываем функцию вычисляющее среднее для каждого заданного распределения

calculate\_mean\_variance(exp\_sample\_100, "exponential")  
calculate\_mean\_variance(exp\_sample\_1000, "exponential")  
calculate\_mean\_variance(norm\_sample\_100, "normal")  
calculate\_mean\_variance(norm\_sample\_1000, "normal")

Математическое ожидание (exponential): 1  
Наблюдаемое среднее (exponential, n=100): 0.9101  
Теоретическая дисперсия (exponential): 1  
Наблюдаемая дисперсия (exponential, n=100): 0.5744  
  
Математическое ожидание (exponential): 1  
Наблюдаемое среднее (exponential, n=1000): 0.9969  
Теоретическая дисперсия (exponential): 1  
Наблюдаемая дисперсия (exponential, n=1000): 0.9619  
  
Математическое ожидание (normal): 0  
Наблюдаемое среднее (normal, n=100): -0.0815  
Теоретическая дисперсия (normal): 1  
Наблюдаемая дисперсия (normal, n=100): 1.2893  
  
Математическое ожидание (normal): 0  
Наблюдаемое среднее (normal, n=1000): 0.0251  
Теоретическая дисперсия (normal): 1  
Наблюдаемая дисперсия (normal, n=1000): 0.9782

## Тоже самое для квантилей

def calculate\_quantiles(sample, distribution\_name):  
 sample\_quantiles = np.quantile(sample, [0.5, 0.99])  
  
 if distribution\_name == "exponential":  
 theoretical\_quantiles = stats.expon.ppf([0.5, 0.99], scale=1)  
 elif distribution\_name == "normal":  
 theoretical\_quantiles = stats.norm.ppf([0.5, 0.99], loc=0, scale=1)  
  
 print(f"Наблюдаемые квантили ({distribution\_name}, n={len(sample)}): {sample\_quantiles}")  
 print(f"Теоретические квантили ({distribution\_name}): {theoretical\_quantiles}")  
 print("\n")

calculate\_quantiles(exp\_sample\_100, "exponential")  
calculate\_quantiles(exp\_sample\_1000, "exponential")  
calculate\_quantiles(norm\_sample\_100, "normal")  
calculate\_quantiles(norm\_sample\_1000, "normal")

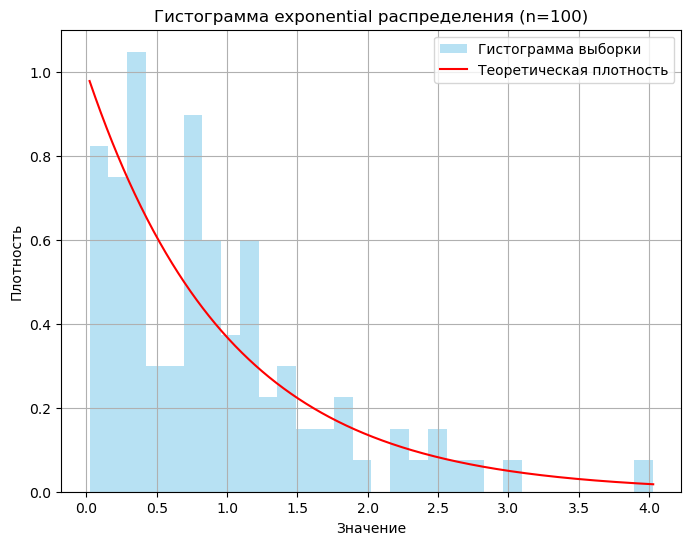
Наблюдаемые квантили (exponential, n=100): [0.79599225 3.10469233]  
Теоретические квантили (exponential): [0.69314718 4.60517019]  
  
  
Наблюдаемые квантили (exponential, n=1000): [0.69396014 4.1181586 ]  
Теоретические квантили (exponential): [0.69314718 4.60517019]  
  
  
Наблюдаемые квантили (normal, n=100): [-0.11287732 2.37344646]  
Теоретические квантили (normal): [0. 2.32634787]  
  
  
Наблюдаемые квантили (normal, n=1000): [-1.7143188e-03 2.2685730e+00]  
Теоретические квантили (normal): [0. 2.32634787]

## Построение графиков

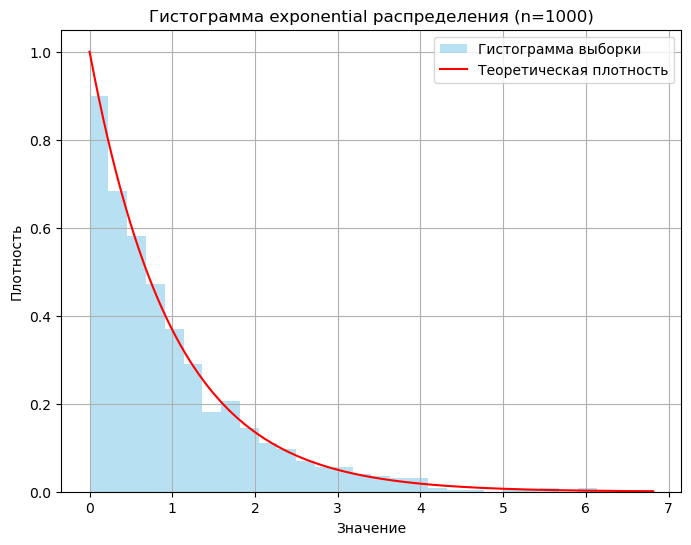
Код для построения графиков взят с [Сайта](https://notebook.community/jaropolk2/python_statistics/stochastic_variables)!

def plot\_histogram(sample, distribution\_name, n):  
 plt.figure(figsize=(8,6))  
 plt.hist(sample, bins=30, density=True, alpha=0.6, color="skyblue", label="Гистограмма выборки")  
  
 x = np.linspace(min(sample), max(sample), 1000)  
 if distribution\_name == "exponential":  
 pdf = stats.expon.pdf(x, scale=1)  
 elif distribution\_name == "normal":  
 pdf = stats.norm.pdf(x, loc=0, scale=1)  
 plt.plot(x, pdf, "r", label="Теоретическая плотность")  
  
 plt.title(f"Гистограмма {distribution\_name} распределения (n={n})")  
 plt.xlabel("Значение")  
 plt.ylabel("Плотность")  
 plt.legend()  
 plt.grid(True)  
 plt.show()

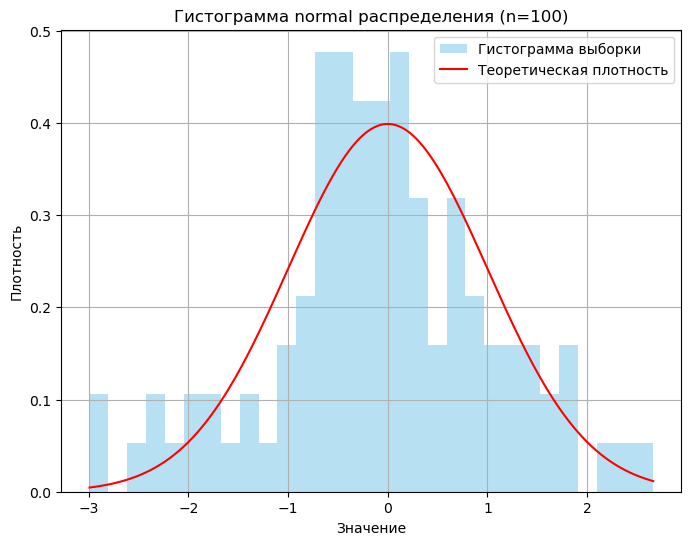
plot\_histogram(exp\_sample\_100, "exponential", 100)  
plot\_histogram(exp\_sample\_1000, "exponential", 1000)  
plot\_histogram(norm\_sample\_100, "normal", 100)  
plot\_histogram(norm\_sample\_1000, "normal", 1000)



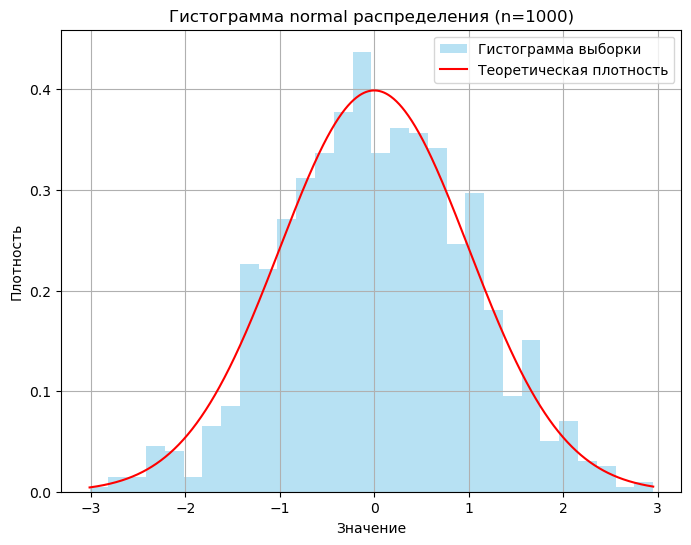
png



png



png

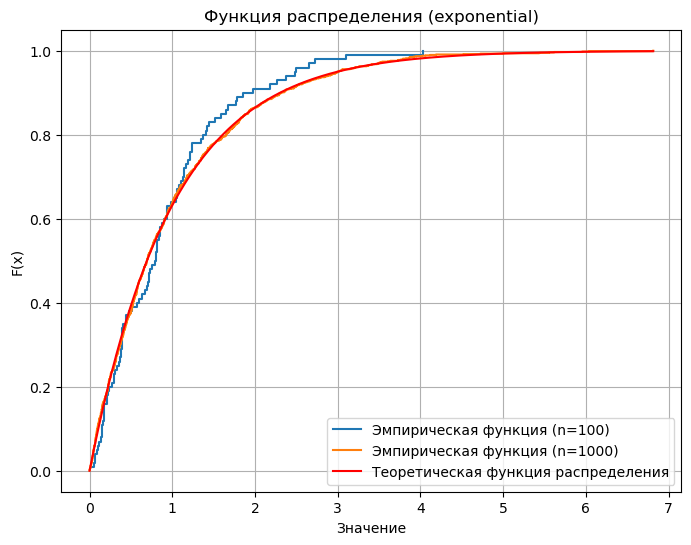


png

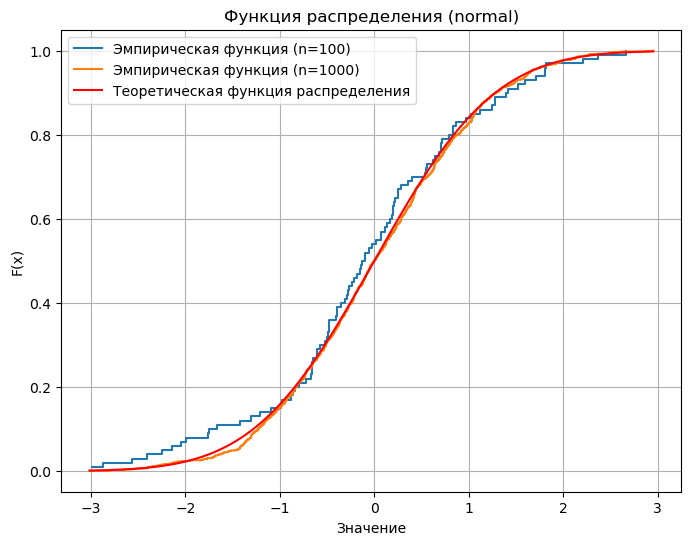
## Функция распределения случайной величины на основе выборки в сравнении с теоретической

def plot\_cdf(sample\_100, sample\_1000, distribution\_name):  
 sorted\_sample\_100 = np.sort(sample\_100)  
 sorted\_sample\_1000 = np.sort(sample\_1000)  
  
 ecdf\_100 = np.arange(1, len(sorted\_sample\_100)+1) / len(sorted\_sample\_100)  
 ecdf\_1000 = np.arange(1, len(sorted\_sample\_1000)+1) / len(sorted\_sample\_1000)  
  
 plt.figure(figsize=(8,6))  
 plt.step(sorted\_sample\_100, ecdf\_100, where="post", label="Эмпирическая функция (n=100)")  
 plt.step(sorted\_sample\_1000, ecdf\_1000, where="post", label="Эмпирическая функция (n=1000)")  
  
 x = np.linspace(min(np.min(sorted\_sample\_100), np.min(sorted\_sample\_1000)),  
 max(np.max(sorted\_sample\_100), np.max(sorted\_sample\_1000)), 1000)  
 if distribution\_name == "exponential":  
 cdf = stats.expon.cdf(x, scale=1)  
 elif distribution\_name == "normal":  
 cdf = stats.norm.cdf(x, loc=0, scale=1)  
 plt.plot(x, cdf, "r", label="Теоретическая функция распределения")  
  
 plt.title(f"Функция распределения ({distribution\_name})")  
 plt.xlabel("Значение")  
 plt.ylabel("F(x)")  
 plt.legend()  
 plt.grid(True)  
 plt.show()

plot\_cdf(exp\_sample\_100, exp\_sample\_1000, "exponential")  
plot\_cdf(norm\_sample\_100, norm\_sample\_1000, "normal")



png

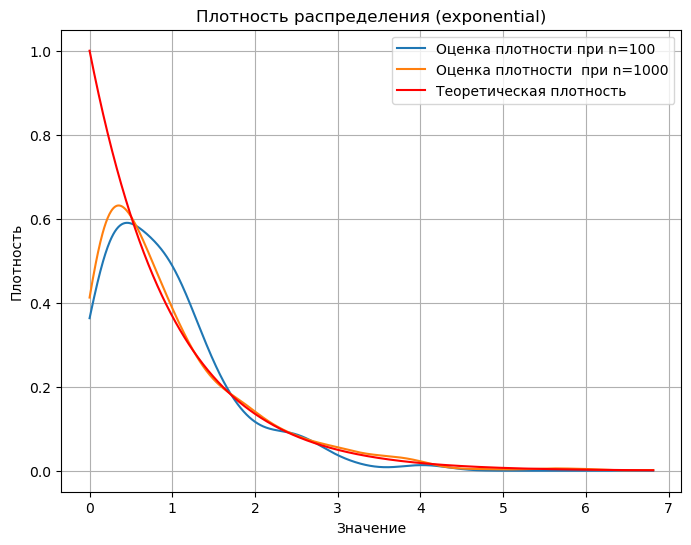


png

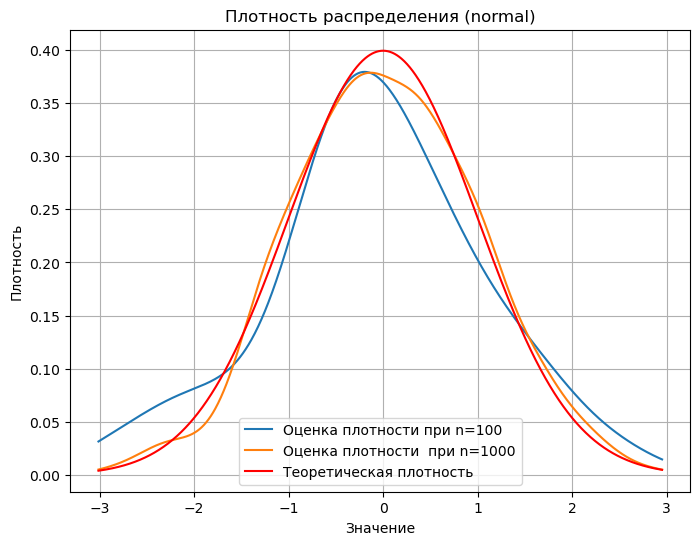
## Плотность распределения случайной величины на основе выборки по сравнению с $$\textcolor{red}{Теоретическим значением}$$

def plot\_density(sample\_100, sample\_1000, distribution\_name):  
 from scipy.stats import gaussian\_kde  
  
 density\_100 = gaussian\_kde(sample\_100)  
 density\_1000 = gaussian\_kde(sample\_1000)  
  
 x = np.linspace(min(np.min(sample\_100), np.min(sample\_1000)),  
 max(np.max(sample\_100), np.max(sample\_1000)), 1000)  
  
 plt.figure(figsize=(8,6))  
 plt.plot(x, density\_100(x), label="Оценка плотности при n=100")  
 plt.plot(x, density\_1000(x), label="Оценка плотности при n=1000")  
  
 if distribution\_name == "exponential":  
 pdf = stats.expon.pdf(x, scale=1)  
 elif distribution\_name == "normal":  
 pdf = stats.norm.pdf(x, loc=0, scale=1)  
 plt.plot(x, pdf, 'r', label="Теоретическая плотность")  
  
 plt.title(f"Плотность распределения ({distribution\_name})")  
 plt.xlabel("Значение")  
 plt.ylabel("Плотность")  
 plt.legend()  
 plt.grid(True)  
 plt.show()

plot\_density(exp\_sample\_100, exp\_sample\_1000, "exponential")  
plot\_density(norm\_sample\_100, norm\_sample\_1000, "normal")



png



png

# Задание №2

*Сгенерировать три выборки размера 100, 1000 и 10000 для случайных расстояний между двумя точками, равномерно распределенные в прямоугольнике со сторонами 10 и 30. Получить среднее значение расстояния между точками, построить функцию распределения вероятностей и плотности вероятностей случайных расстояний. Показать разницу между соответствующими функциями на одном графике.*

## Создание 2 случайных точек и вычисление расстояни между ними

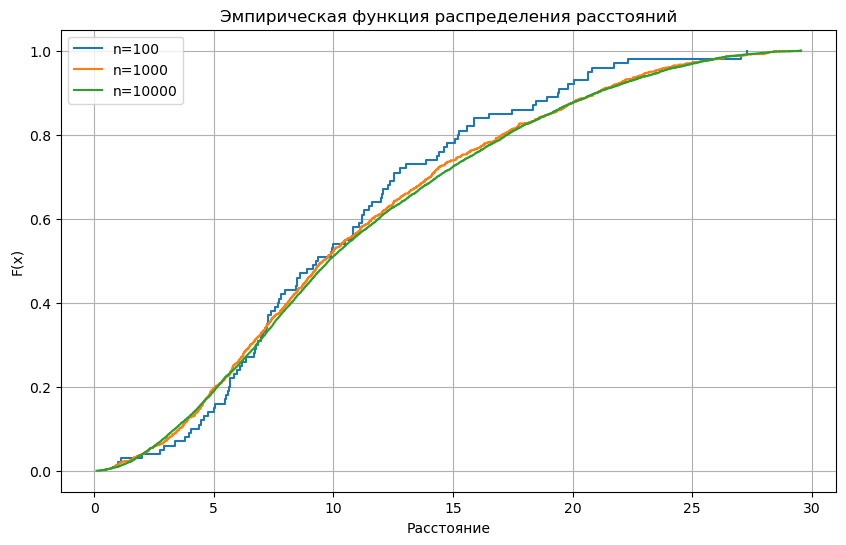
def generate\_points(sample\_size, width=10, height=30):  
 x1 = np.random.uniform(0, width, sample\_size)  
 y1 = np.random.uniform(0, height, sample\_size)  
 x2 = np.random.uniform(0, width, sample\_size)  
 y2 = np.random.uniform(0, height, sample\_size)  
 distances = np.sqrt((x1 - x2)\*\*2 + (y1 - y2)\*\*2)  
 return distances

## Задаем 3 расстояния из задания и в каждом случае создаем точки, между котороми будем считать расстояние

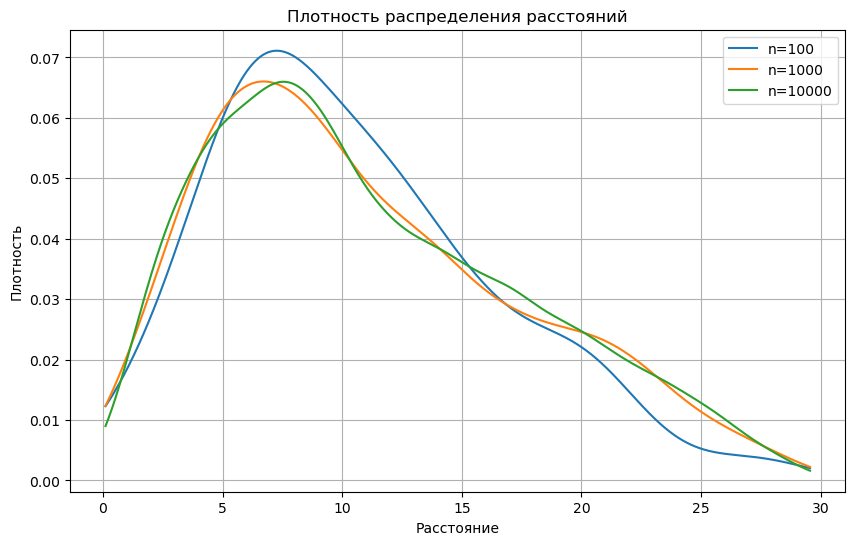
sizes = [100, 1000, 10000]  
distance\_samples = {}  
  
for size in sizes:  
 distances = generate\_points(size, width=10, height=30)  
 distance\_samples[size] = distances

from scipy.stats import gaussian\_kde  
def plot\_cdf(samples\_dict):  
 plt.figure(figsize=(10,6))  
 for size, distances in samples\_dict.items():  
 sorted\_distances = np.sort(distances)  
 ecdf = np.arange(1, len(sorted\_distances)+1) / len(sorted\_distances)  
 plt.step(sorted\_distances, ecdf, where="post", label=f"n={size}")  
   
 plt.title("Эмпирическая функция распределения расстояний")  
 plt.xlabel("Расстояние")  
 plt.ylabel("F(x)")  
 plt.legend()  
 plt.grid(True)  
 plt.show()  
  
def plot\_pdf(samples\_dict):  
 plt.figure(figsize=(10,6))  
 x\_min = min([min(distances) for distances in samples\_dict.values()])  
 x\_max = max([max(distances) for distances in samples\_dict.values()])  
 x = np.linspace(x\_min, x\_max, 1000)  
   
 for size, distances in samples\_dict.items():  
 density = gaussian\_kde(distances)  
 plt.plot(x, density(x), label=f'n={size}')  
   
 plt.title("Плотность распределения расстояний")  
 plt.xlabel("Расстояние")  
 plt.ylabel("Плотность")  
 plt.legend()  
 plt.grid(True)  
 plt.show()

plot\_cdf(distance\_samples)  
plot\_pdf(distance\_samples)



png



png