ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| ст. преподаватель |  |  |  | В.А. Миклуш |
| --- | --- | --- | --- | --- |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 |
| --- |
| **ОЦЕНКА ГЕНЕРАЛЬНЫХ ПАРАМЕТРОВ СЛУЧАЙНЫХ ВЕЛИЧИН** |
| по курсу: Теория информации, данные, знания |
|  |

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ ГР. № | 4128 |  |  |  | В. А. Воробьев |
| --- | --- | --- | --- | --- | --- |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

***Цель работы****:* Нахождение основных статистических характеристик распределения случайных величин, построение функции распределения, определение закона распределения в среде математического редактора.

**Задание**

1. Найти числовые параметры распределения.

2. Найти закон распределения.

3. Провести степенную аппроксимацию произвольно выбранных данных

Вариант задания - **№5**

**Ход работы**

В ходе работы была создана программа на языке Python, выполняющая все поставленные задачи. Исходный код программы представлен в Приложении.

**1 Числовые параметры распределения**

* + Объем выборки: 500
  + X минимальный: -8.01379391414643
  + X максимальный: 6.144979813287666
  + Выборочное математическое ожидание (среднее арифметическое всех элементов выборки)

Формула для МО:

Мх: -0.45437341526625824

* + Число степеней свободы выборки (количество значений или наблюдений в выборке, которые могут быть изменены независимо друг от друга без изменения ее структуры).

f: 499

* + Выборочная дисперсия (среднее квадратов отклонений каждого элемента выборки от выборочного среднего)

Формулы для выборочной дисперсии:

Dx: 5.362700285478515

* + CКО(корень дисперсии), Sx: 2.315750479969402
  + Асимметрия (отражает степень и направление отклонения распределения от симметрии).

Формула для асимметрии:

Ax: -0.07735713294746685

* + Эксцесс (характеризует остроту или плоскость вершины распределения).

Формула для эксцесса:

Ех: -0.14901143278322104

* + Медиана (значение, которое разделяет выборку на две равные части), Medx: -0.5205897905121429
  + Размах (разница между максимальным и минимальным значениями выборки), R: 14.158773727434095
  + Дисперсия ассиметрии (измеряет степень изменчивости асимметрии в различных выборках), Da: 0.011880810942726873
  + Дисперсия эксцесса (показывает степень изменчивости эксцесса в различных выборках), De: 0.00011549620221635089
  + Доверительные интервалы — это интервалы, которые с высокой вероятностью содержат истинное значение параметра генеральной совокупности. Ниже представлены доверительные интервалы при q = 0.95.

1. Для генерального математического ожидания

-0.657847686888222 <= mx <= -0.25089914364429444

1. Для генеральной дисперсии:

4.754863872733365 <= Dx <= 6.0956708362501315

1. Для генеральной асимметрии:

-0.291940529496238 <= ax <= 0.1372262636013043

1. Для генерального эксцесса:

-0.573063147083164 <= ex <= 0.27504028151672194

Результат вычисление числовых параметров изображен на рисунке 1.

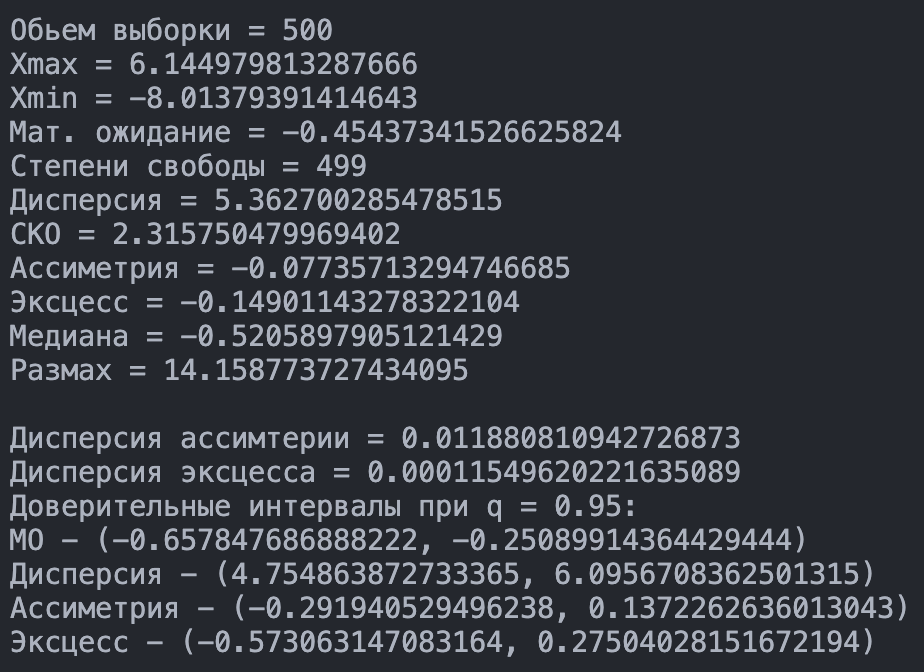


Рисунок 1 – Числовые параметры распределения

**2 Найти закон распределения**

1. Проверим нашу выборку на нормальность с помощью: по простейшего теста и теста Жарка-Бера. Для проверки возьмём один уровень значимости q = 5%.
   1. Простейший тест: принимаем основную гипотезу.
   2. Тест Жарка-Бера: принимаем основную гипотезу.

Результат прохождения тестов был выведен на консоль и представлен на рисунке 2.

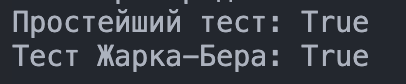


Рисунок 2 – Результат проверки выборки

1. Строим гистограмму плотности распределения для заданного массива исходных данных в сравнении с теоретическими законами распределения плотности случайных величин. Результат представлен на рисунке 3.

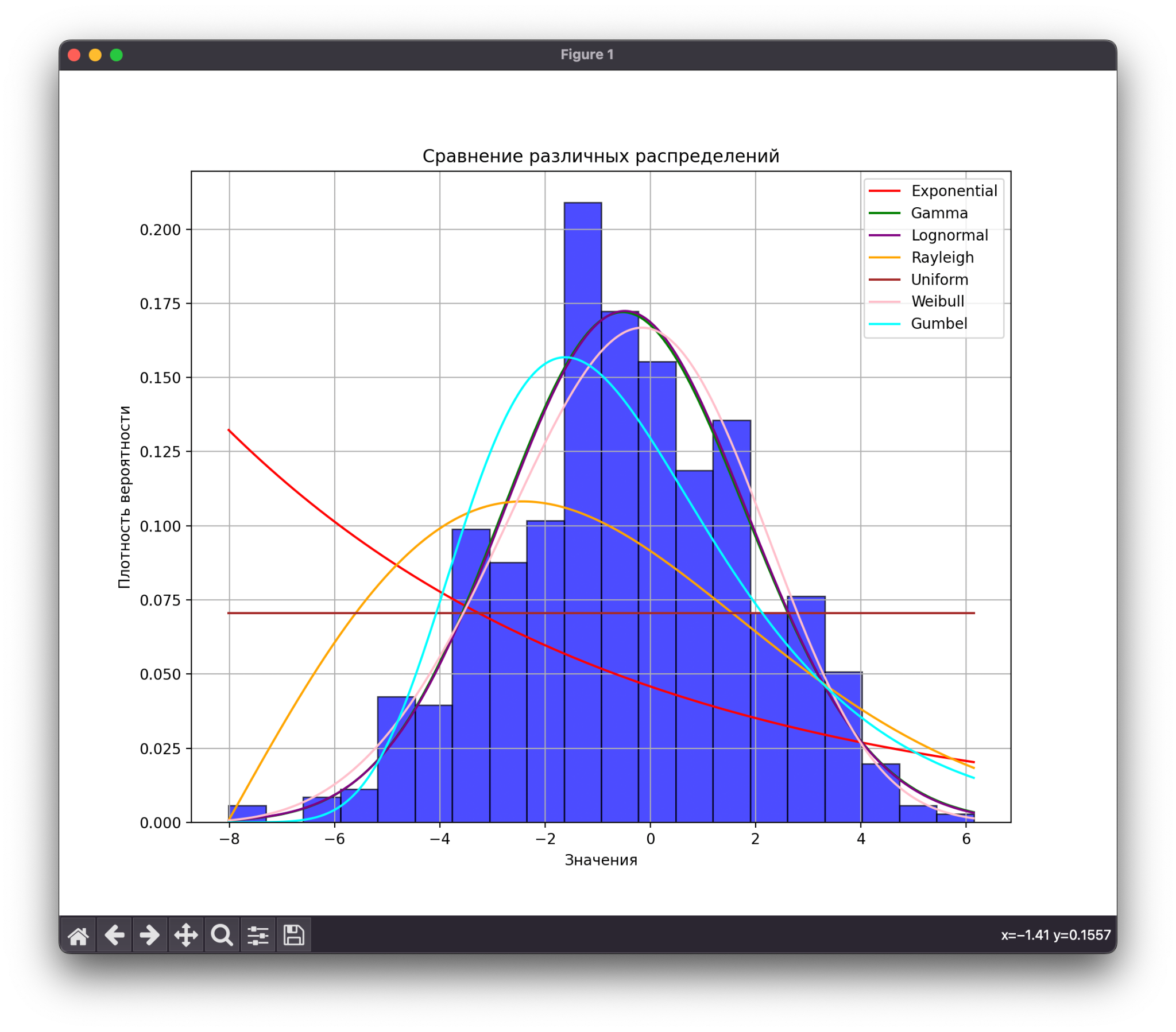


Рисунок 3 – Эмпирическая и теоретические плотности распределения

1. Строим графики функции распределения. Результат представлен на рисунке 4.

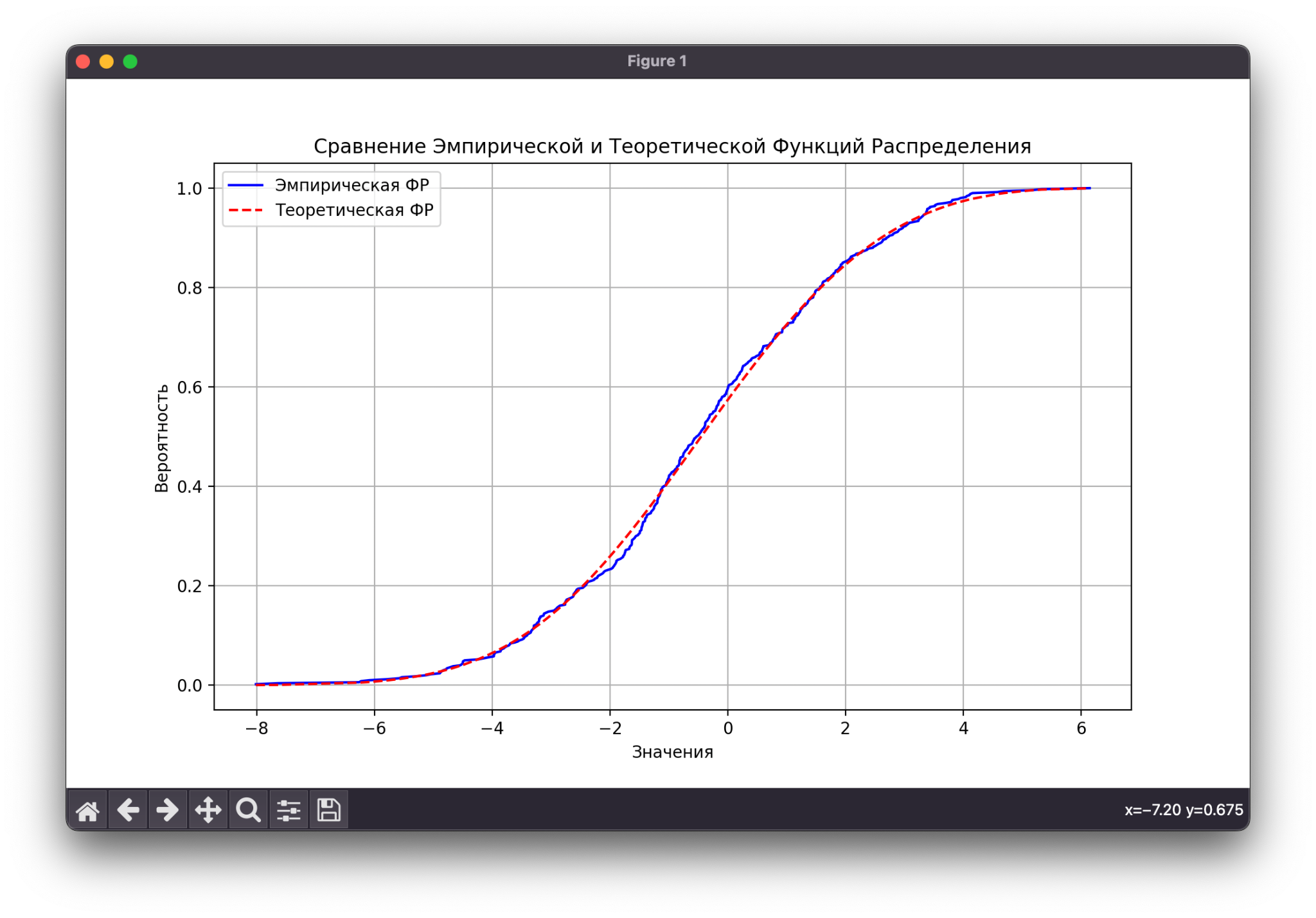


Рисунок 4 – Эмпирическая и теоретические плотности распределения

1. Проверка непрерывных распределений по критерию Колмогорова = 0.93435435244320 у нормального распределния.

**3 Степенная аппроксимацию произвольно выбранных данных**

Степенная аппроксимация данных представляет собой метод приближения данных к функции степенного закона. Этот метод может быть использован для приближения нелинейных данных, где зависимость между переменными может быть описана степенной функцией. Проведем ее для функции гармонических колебаний с шумом.

Результат аппроксимации представлены на рисунках 5, 6 и 7.

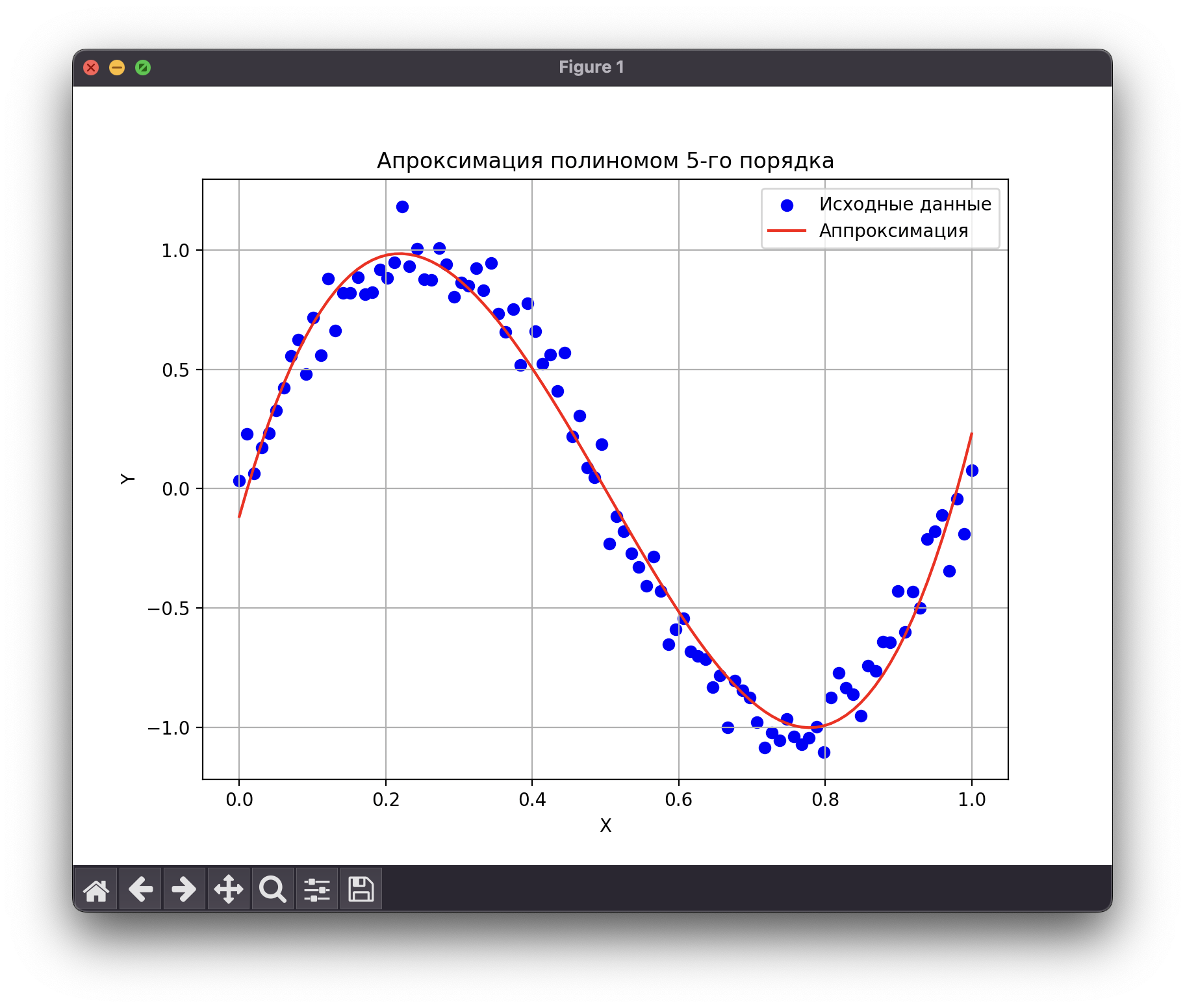


Рисунок 5 – Аппроксимация полиномом 5-го порядка

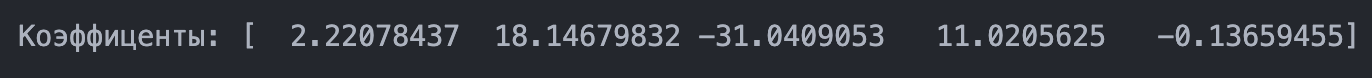


Рисунок 6 – Коэффициенты полинома

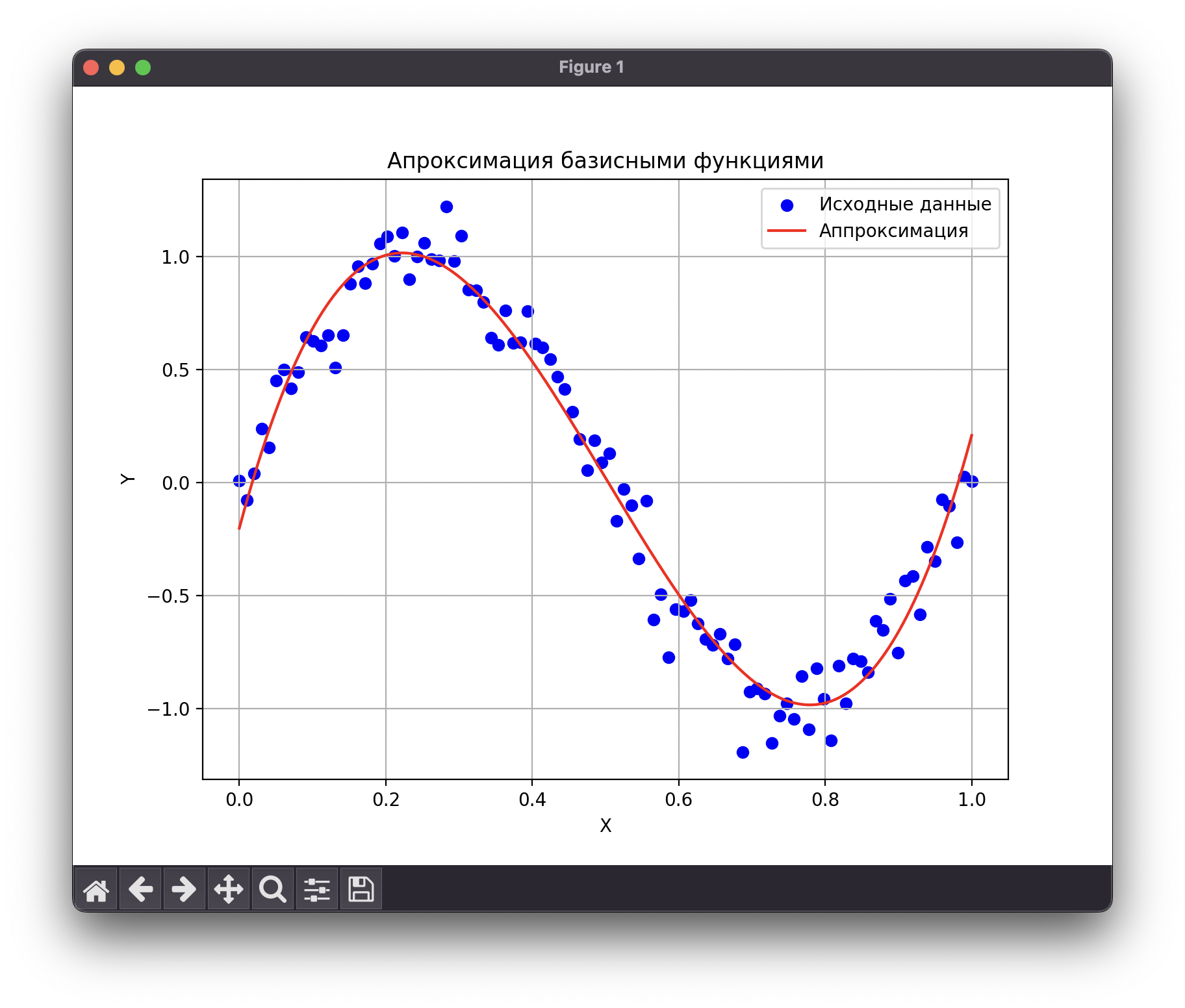


Рисунок 7 – Аппроксимация базисными функциями

**ВЫВОД**

В ходе выполнения лабораторной работы, на основе исходных данных, соответствующих варианту с использованием языка программирования Python, была создана программа, вычисляющая необходимые статистические значения и построены гистограмма распределения, функции распределения и степенная аппроксимация данных гармонической функции при помощи базисных функций и полинома 5-ой степени.

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ КОД

ploter.py

from statsmodels.distributions.empirical\_distribution import ECDF

from scipy.stats import genextreme

from typing import List

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import gamma, expon, lognorm, rayleigh, uniform, weibull\_min, gumbel\_r

def plot\_data\_and\_approximation(x\_values, y\_values, approx\_points):

"""

Строит график точек исходной выборки и аппроксимированной функции.

Параметры:

- x\_values (np.ndarray): Массив значений x исходной выборки.

- y\_values (np.ndarray): Массив значений y исходной выборки.

- approx\_points (list): Список точек аппроксимированной кривой в формате (x, y).

"""

plt.figure(figsize=(8, 6))

# График исходной выборки

plt.scatter(x\_values, y\_values, label='Исходные данные', color='blue')

# График аппроксимированной функции

x\_approx, y\_approx = zip(\*approx\_points)

plt.plot(x\_approx, y\_approx, label='Аппроксимация', color='red')

# Настройки графика

plt.title('Аппроксимация полиномом')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.grid(True)

# Отображение графика

plt.show()

def plot\_histogram(values, bins=32):

"""

Строит гистограмму распределения случайной величины.

Параметры:

- values (List[float]): Список значений случайной величины.

- bins (int): Количество интервалов гистограммы (по умолчанию 10).

"""

plt.figure(figsize=(8, 6))

# Строим гистограмму

plt.hist(values, bins=bins, color='blue', edgecolor='black', alpha=0.7)

# Настройки графика

plt.title('Гистограмма распределения случайной величины')

plt.xlabel('Значения')

plt.ylabel('Частота')

plt.grid(True)

# Отображение гистограммы

plt.show()

def plot\_all\_distributions(data: List[float]):

"""

Строит график гистограммы и плотностей вероятности различных распределений.

Параметры:

- data (List[float]): Массив значений случайных величин.

"""

plt.figure(figsize=(10, 8))

# Строим гистограмму

plt.hist(data, bins=20, density=True, alpha=0.7,

color='blue', edgecolor='black')

# Оцениваем параметры и строим плотности вероятности

x = np.linspace(min(data), max(data), 1000)

# Экспоненциальное распределение

exp\_params = expon.fit(data)

exp\_pdf = expon.pdf(x, \*exp\_params)

plt.plot(x, exp\_pdf, label='Exponential', color='red')

# Гамма-распределение

gamma\_params = gamma.fit(data)

gamma\_pdf = gamma.pdf(x, \*gamma\_params)

plt.plot(x, gamma\_pdf, label='Gamma', color='green')

# Логнормальное распределение

lognorm\_params = lognorm.fit(data)

lognorm\_pdf = lognorm.pdf(x, \*lognorm\_params)

plt.plot(x, lognorm\_pdf, label='Lognormal', color='purple')

# Рэлеевское распределение

rayleigh\_params = rayleigh.fit(data)

rayleigh\_pdf = rayleigh.pdf(x, \*rayleigh\_params)

plt.plot(x, rayleigh\_pdf, label='Rayleigh', color='orange')

# Равномерное распределение

uniform\_params = uniform.fit(data)

uniform\_pdf = uniform.pdf(x, \*uniform\_params)

plt.plot(x, uniform\_pdf, label='Uniform', color='brown')

# Вейбулловское распределение

weibull\_params = weibull\_min.fit(data)

weibull\_pdf = weibull\_min.pdf(x, \*weibull\_params)

plt.plot(x, weibull\_pdf, label='Weibull', color='pink')

gumbel\_params = gumbel\_r.fit(data)

gumbel\_pdf = gumbel\_r.pdf(x, \*gumbel\_params)

plt.plot(x, gumbel\_pdf, label='Gumbel', color='cyan')

# Настройки графика

plt.title('Сравнение различных распределений')

plt.xlabel('Значения')

plt.ylabel('Плотность вероятности')

plt.legend()

plt.grid(True)

# Отображение графика

plt.show()

def plot\_empirical\_vs\_theoretical\_cdf(data: List[float], theoretical\_distribution='genextreme'):

"""

Строит график эмпирической и теоретической функций распределения.

Параметры:

- data (List[float]): Массив значений случайных величин.

- theoretical\_distribution (str): Название теоретического распределения ('genextreme' по умолчанию).

"""

plt.figure(figsize=(10, 6))

# Эмпирическая функция распределения (ЭФР)

ecdf = ECDF(data)

x\_values = np.sort(data)

y\_values\_empirical = ecdf(x\_values)

# Теоретическая функция распределения (ТФР)

if theoretical\_distribution == 'genextreme':

params = genextreme.fit(data)

y\_values\_theoretical = genextreme.cdf(x\_values, \*params)

# Добавьте другие теоретические распределения, если необходимо

# Строим графики

plt.plot(x\_values, y\_values\_empirical,

label='Эмпирическая ФР', color='blue')

plt.plot(x\_values, y\_values\_theoretical,

label='Теоретическая ФР', linestyle='--', color='red')

# Настройки графика

plt.title('Сравнение Эмпирической и Теоретической Функций Распределения')

plt.xlabel('Значения')

plt.ylabel('Вероятность')

plt.legend()

plt.grid(True)

# Отображение графика

plt.show()

distribution\_params.py

from scipy.stats import skew

from typing import List

from typing import List, Tuple

from scipy.stats import chi2, t, skewnorm

from typing import List, Optional

import numpy as np

def sample\_size(data: List[float]) -> int:

"""

Возвращает объем выборки.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- size (int): Объем выборки.

"""

return len(data)

def calculate\_Xmin(data: List[float]) -> Optional[float]:

"""

Возвращает минимальное значение в выборке.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- Xmin (float): Минимальное значение в выборке, или None, если выборка пуста.

"""

if not data:

return None

return min(data)

def calculate\_Xmax(data: List[float]) -> Optional[float]:

"""

Возвращает максимальное значение в выборке.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- Xmax (float): Максимальное значение в выборке, или None, если выборка пуста.

"""

if not data:

return None

return max(data)

def calculate\_mean(data: List[float]) -> Optional[float]:

"""

Возвращает выборочное математическое ожидание.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- mean (float): Выборочное математическое ожидание, или None, если выборка пуста.

"""

if not data:

return None

return sum(data) / len(data)

def degrees\_of\_freedom(data: List[float]) -> int:

"""

Возвращает число степеней свободы выборки.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- df (int): Число степеней свободы.

"""

return len(data) - 1

def calculate\_variance(data: List[float]) -> Optional[float]:

"""

Возвращает выборочную дисперсию.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- variance (float): Выборочная дисперсия, или None, если выборка пуста.

"""

if not data:

return None

mean = calculate\_mean(data)

return sum((x - mean) \*\* 2 for x in data) / (len(data) - 1)

def calculate\_std\_dev(data: List[float]) -> Optional[float]:

"""

Возвращает стандартное отклонение.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- std\_dev (float): Стандартное отклонение, или None, если выборка пуста.

"""

if not data:

return None

mean = calculate\_mean(data)

variance = calculate\_variance(data)

return variance \*\* 0.5

def calculate\_skewness(data: List[float]) -> Optional[float]:

"""

Возвращает коэффициент асимметрии.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- skewness (float): Коэффициент асимметрии, или None, если выборка пуста.

"""

if not data:

return None

mean = calculate\_mean(data)

std\_dev = calculate\_std\_dev(data)

return sum((x - mean) \*\* 3 for x in data) / (len(data) \* std\_dev \*\* 3)

def calculate\_kurtosis(data: List[float]) -> Optional[float]:

"""

Возвращает коэффициент эксцесса.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- kurtosis (float): Коэффициент эксцесса, или None, если выборка пуста.

"""

if not data:

return None

mean = calculate\_mean(data)

std\_dev = calculate\_std\_dev(data)

return sum((x - mean) \*\* 4 for x in data) / (len(data) \* std\_dev \*\* 4) - 3

def calculate\_median(data: List[float]) -> Optional[float]:

"""

Возвращает медиану.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- median (float): Медиана, или None, если выборка пуста.

"""

if not data:

return None

sorted\_data = sorted(data)

n = len(sorted\_data)

if n % 2 == 0:

# Если число элементов четное, медиана - среднее двух центральных элементов

middle1 = sorted\_data[n // 2 - 1]

middle2 = sorted\_data[n // 2]

return (middle1 + middle2) / 2

else:

# Если число элементов нечетное, медиана - центральный элемент

return sorted\_data[n // 2]

def calculate\_range(data: List[float]) -> Optional[float]:

"""

Возвращает размах.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- range (float): Размах, или None, если выборка пуста.

"""

if not data:

return None

return max(data) - min(data)

def confidence\_interval\_mean(data: List[float], confidence\_level: float = 0.95) -> Tuple[float, float]:

"""

Возвращает доверительный интервал для генерального математического ожидания.

Параметры:

- data (List[float]): Массив значений случайных величин.

- confidence\_level (float): Уровень доверия (по умолчанию 0.95).

Возвращает:

- confidence\_interval (Tuple[float, float]): Доверительный интервал.

"""

if not data:

return None

mean\_value = calculate\_mean(data)

std\_dev\_value = calculate\_std\_dev(data)

sample\_size\_value = sample\_size(data)

if mean\_value is None or std\_dev\_value is None or sample\_size\_value is None:

return None

# Вычисляем стандартную ошибку среднего

standard\_error = std\_dev\_value / (sample\_size\_value \*\* 0.5)

# Вычисляем критическое значение t-распределения

df = degrees\_of\_freedom(data)

if df is None:

return None

t\_value = t.ppf((1 + confidence\_level) / 2, df)

# Вычисляем доверительный интервал

margin\_of\_error = t\_value \* standard\_error

lower\_bound = mean\_value - margin\_of\_error

upper\_bound = mean\_value + margin\_of\_error

return lower\_bound, upper\_bound

def confidence\_interval\_variance(data: List[float], confidence\_level: float = 0.95) -> Tuple[float, float]:

"""

Возвращает доверительный интервал для генеральной дисперсии.

Параметры:

- data (List[float]): Массив значений случайных величин.

- confidence\_level (float): Уровень доверия (по умолчанию 0.95).

Возвращает:

- confidence\_interval (Tuple[float, float]): Доверительный интервал.

"""

if not data:

return None

variance\_value = calculate\_variance(data)

sample\_size\_value = sample\_size(data)

if variance\_value is None or sample\_size\_value is None:

return None

# Вычисляем критические значения хи-квадрат распределения

df = degrees\_of\_freedom(data)

if df is None:

return None

chi2\_lower = chi2.ppf((1 - confidence\_level) / 2, df)

chi2\_upper = chi2.ppf((1 + confidence\_level) / 2, df)

# Вычисляем доверительный интервал

lower\_bound = (sample\_size\_value - 1) \* variance\_value / chi2\_upper

upper\_bound = (sample\_size\_value - 1) \* variance\_value / chi2\_lower

return lower\_bound, upper\_bound

def confidence\_interval\_skewness(data: List[float], confidence\_level: float = 0.95) -> Tuple[float, float]:

"""

Возвращает доверительный интервал для генеральной асимметрии.

Параметры:

- data (List[float]): Массив значений случайных величин.

- confidence\_level (float): Уровень доверия (по умолчанию 0.95).

Возвращает:

- confidence\_interval (Tuple[float, float]): Доверительный интервал.

"""

if not data:

return None

skewness\_value = calculate\_skewness(data)

sample\_size\_value = sample\_size(data)

if skewness\_value is None or sample\_size\_value is None:

return None

# Вычисляем стандартную ошибку асимметрии

std\_error\_skewness = (6 \* sample\_size\_value \* (sample\_size\_value - 1) / (

(sample\_size\_value - 2) \* (sample\_size\_value + 1) \* (sample\_size\_value + 3))) \*\* 0.5

# Вычисляем критическое значение t-распределения

df = degrees\_of\_freedom(data)

if df is None:

return None

t\_value = t.ppf((1 + confidence\_level) / 2, df)

# Вычисляем доверительный интервал

lower\_bound = skewness\_value - t\_value \* std\_error\_skewness

upper\_bound = skewness\_value + t\_value \* std\_error\_skewness

return lower\_bound, upper\_bound

def confidence\_interval\_kurtosis(data: List[float], confidence\_level: float = 0.95) -> Tuple[float, float]:

"""

Возвращает доверительный интервал для генерального эксцесса.

Параметры:

- data (List[float]): Массив значений случайных величин.

- confidence\_level (float): Уровень доверия (по умолчанию 0.95).

Возвращает:

- confidence\_interval (Tuple[float, float]): Доверительный интервал.

"""

if not data:

return None

kurtosis\_value = calculate\_kurtosis(data)

sample\_size\_value = sample\_size(data)

if kurtosis\_value is None or sample\_size\_value is None:

return None

# Вычисляем стандартную ошибку эксцесса

std\_error\_kurtosis = (24 \* sample\_size\_value \* (sample\_size\_value - 2) \* (sample\_size\_value - 3) / (

(sample\_size\_value + 1) \* (sample\_size\_value + 1) \* (sample\_size\_value + 3) \* (sample\_size\_value + 5))) \*\* 0.5

# Вычисляем критическое значение t-распределения

df = degrees\_of\_freedom(data)

if df is None:

return None

t\_value = t.ppf((1 + confidence\_level) / 2, df)

# Вычисляем доверительный интервал

lower\_bound = kurtosis\_value - t\_value \* std\_error\_kurtosis

upper\_bound = kurtosis\_value + t\_value \* std\_error\_kurtosis

return lower\_bound, upper\_bound

def skewness\_variance(data):

n = len(data)

return 6 \* (n - 1) / (n + 1) / (n + 3)

def kurtosis\_variance(values: List[float]) -> float:

"""

Вычисляет дисперсию эксцесса для массива значений.

Параметры:

- values (List[float]): Массив значений случайных величин.

Возвращает:

- excess\_variance (float): Дисперсия эксцесса.

"""

# 24 \* n \* (n - 2) \* (n - 3) / (n + 1)\*\*2 / (n + 3) / (n + 5)

excess\_variance = np.var(

values, ddof=1) \* np.var(values, ddof=2) / (len(values) \* (len(values) - 1))

return excess\_variance

reader.py

from typing import List

def read\_random\_variables(path: str) -> List[float]:

"""

Считывает файл с случайными величинами и возвращает массив значений.

Параметры:

- path (str): Путь к файлу.

Возвращает:

- values (List[float]): Массив значений случайных величин.

"""

try:

with open(path, 'r') as file:

values = [float(value) for value in file.read().split()]

return values

except FileNotFoundError:

print(f"Файл по пути {path} не найден.")

return []

except Exception as e:

print(f"Произошла ошибка при чтении файла: {e}")

return []

approximation.py

import numpy as np

def approximate\_polynomial(x\_values, y\_values, degree=3):

"""

Аппроксимирует данные полиномом.

Параметры:

- x\_values (np.ndarray): Массив значений x.

- y\_values (np.ndarray): Массив значений y.

- degree (int): Степень полинома для аппроксимации (по умолчанию 3).

Возвращает:

- approx\_points (list): Список точек аппроксимированной кривой в формате (x, y).

"""

coefficients = np.polyfit(x\_values, y\_values, degree)

approx\_polynomial = np.poly1d(coefficients)

# Генерируем точки для построения аппроксимированной кривой

x\_range = np.linspace(min(x\_values), max(x\_values), 100)

approx\_points = list(zip(x\_range, approx\_polynomial(x\_range)))

return (approx\_points, coefficients)

def approximate\_basis\_functions(x\_values, y\_values, degree):

"""

Аппроксимирует исходные данные базисными функциями.

Параметры:

- x\_values (np.ndarray): Массив значений x исходной выборки.

- y\_values (np.ndarray): Массив значений y исходной выборки.

- degree (int): Степень многочлена для аппроксимации.

Возвращает:

- approx\_points (list): Список точек аппроксимированной кривой в формате (x, y).

"""

# Построение матрицы базисных функций

basis\_matrix = np.vander(x\_values, degree + 1)

# Нахождение коэффициентов аппроксимации

coefficients, \_, \_, \_ = np.linalg.lstsq(basis\_matrix, y\_values, rcond=None)

# Генерация точек аппроксимированной кривой для отображения графика

x\_approx = np.linspace(min(x\_values), max(x\_values), 1000)

basis\_matrix\_approx = np.vander(x\_approx, degree + 1)

y\_approx = np.dot(basis\_matrix\_approx, coefficients)

# Сбор точек в список

approx\_points = list(zip(x\_approx, y\_approx))

return approx\_points

def generate\_random\_data(size: int = 100, noise\_factor: float = 0.1):

"""

Генерирует произвольные данные для аппроксимации.

Параметры:

- size (int): Размер массива данных (по умолчанию 100).

- noise\_factor (float): Фактор шума (по умолчанию 0.1).

Возвращает:

- x\_values (np.ndarray): Массив значений x.

- y\_values (np.ndarray): Массив значений y (кривая с шумом).

"""

x\_values = np.linspace(0, 1, size)

true\_curve = np.sin(2 \* np.pi \* x\_values)

noise = noise\_factor \* np.random.randn(size)

y\_values = true\_curve + noise

return x\_values, y\_values

distribution\_law.py

from scipy.stats import ks\_2samp

from scipy.stats import weibull\_min

from scipy.stats import uniform

from scipy.stats import rayleigh

from scipy.stats import lognorm

from scipy.stats import expon

from scipy.stats import gamma

from typing import List, Callable, Any

from scipy.stats import shapiro, jarque\_bera

import numpy as np

from distribution\_params import \*

def normality\_test\_q(data: List[float], alpha: float = 0.05) -> bool:

"""

Проверяет гипотезу о нормальном распределении генеральной совокупности при помощи простейшего теста q.

Параметры:

- data (List[float]): Массив значений случайных величин.

- alpha (float): Уровень значимости (по умолчанию 0.05).

Возвращает:

- result (bool): True, если гипотеза о нормальном распределении не отвергается, иначе False.

"""

\_, p\_value = shapiro(data)

return p\_value > alpha

def normality\_test\_jarque\_bera(data: List[float], alpha: float = 0.05) -> bool:

"""

Проверяет гипотезу о нормальном распределении генеральной совокупности при помощи теста Жарка-Бера.

Параметры:

- data (List[float]): Массив значений случайных величин.

- alpha (float): Уровень значимости (по умолчанию 0.05).

Возвращает:

- result (bool): True, если гипотеза о нормальном распределении не отвергается, иначе False.

"""

\_, p\_value = jarque\_bera(data)

return p\_value > alpha

def calculate\_histogram(data: List[float], bins: int = 10) -> Tuple[np.ndarray, np.ndarray]:

"""

Рассчитывает гистограмму распределения и возвращает данные для построения.

Параметры:

- data (List[float]): Массив значений случайных величин.

- bins (int): Количество интервалов (по умолчанию 10).

Возвращает:

- histogram\_data (Tuple[np.ndarray, np.ndarray]): Кортеж из двух массивов: значения интервалов и соответствующие частоты.

"""

histogram\_values, bin\_edges = np.histogram(data, bins=bins, density=False)

return bin\_edges[:-1], histogram\_values

def fit\_gamma\_distribution(data: List[float]) -> Tuple[float, float]:

"""

Оценивает параметры гамма-распределения.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- shape (float): Параметр формы (k) гамма-распределения.

- scale (float): Параметр масштаба (θ) гамма-распределения.

"""

shape, \_, scale = gamma.fit(data)

return shape, scale

def fit\_exponential\_distribution(data: List[float]) -> float:

"""

Оценивает параметр экспоненциального распределения.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- scale (float): Параметр масштаба (λ) экспоненциального распределения.

"""

\_, scale = expon.fit(data)

return scale

def fit\_gamma\_distribution(data: List[float]) -> Tuple[float, float]:

"""

Оценивает параметры гамма-распределения.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- shape (float): Параметр формы (k) гамма-распределения.

- scale (float): Параметр масштаба (θ) гамма-распределения.

"""

shape, \_, scale = gamma.fit(data)

return shape, scale

def fit\_lognormal\_distribution(data: List[float]) -> Tuple[float, float]:

"""

Оценивает параметры логнормального распределения.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- s (float): Параметр формы логнормального распределения.

- scale (float): Параметр масштаба логнормального распределения.

"""

s, \_, scale = lognorm.fit(data)

return s, scale

def fit\_rayleigh\_distribution(data: List[float]) -> float:

"""

Оценивает параметр рэлеевского распределения.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- scale (float): Параметр масштаба рэлеевского распределения.

"""

\_, scale = rayleigh.fit(data)

return scale

def fit\_uniform\_distribution(data: List[float]) -> Tuple[float, float]:

"""

Оценивает параметры равномерного распределения.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- loc (float): Параметр левой границы равномерного распределения.

- scale (float): Разница между правой и левой границами равномерного распределения.

"""

loc, scale = uniform.fit(data)

return loc, scale

def fit\_weibull\_distribution(data: List[float]) -> Tuple[float, float]:

"""

Оценивает параметры вейбуловского распределения.

Параметры:

- data (List[float]): Массив значений случайных величин.

Возвращает:

- c (float): Параметр формы вейбуловского распределения.

- loc (float): Параметр сдвига вейбуловского распределения.

- scale (float): Параметр масштаба вейбуловского распределения.

"""

c, loc, scale = weibull\_min.fit(data)

return c, loc, scale

def get\_distribution\_points(data: List[float], theoretical\_distribution\_func: Callable[[float], float], params: Tuple[float, ...], bins: int = 100) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:

"""

Возвращает точки для построения графика эмпирической и теоретической (например, PDF) распределений.

Параметры:

- data (List[float]): Массив значений случайных величин.

- theoretical\_distribution\_func (Callable[[float], float]): Функция теоретического распределения.

- params (Tuple[float, ...]): Параметры теоретического распределения.

- bins (int): Количество интервалов для гистограммы (по умолчанию 100).

Возвращает:

- empirical\_points (Tuple[np.ndarray, np.ndarray]): Кортеж из двух массивов: значения и частоты эмпирического распределения.

- theoretical\_points (Tuple[np.ndarray, np.ndarray]): Кортеж из двух массивов: значения и вероятности теоретического распределения.

"""

# Рассчитываем гистограмму для эмпирического распределения

empirical\_values, \_ = np.histogram(data, bins=bins, density=True)

empirical\_points = np.histogram(data, bins=bins)

# Рассчитываем значения теоретического распределения

theoretical\_values = [theoretical\_distribution\_func(

x, \*params) for x in empirical\_points[1]]

return empirical\_points[1][:-1], empirical\_values, theoretical\_values

# Пример использования:

# Пусть theoretical\_distribution\_func - это функция PDF гамма-распределения

# gamma\_params = fit\_gamma\_distribution(random\_values)

# gamma\_pdf = lambda x, shape, scale: gamma.pdf(x, shape, scale=scale)

# # Получаем точки для построения графика

# empirical\_points, empirical\_values, theoretical\_values = get\_distribution\_points(random\_values, gamma\_pdf, gamma\_params)

def kolmogorov\_smirnov\_test(data: List[float], theoretical\_distribution\_func: Callable[[float, \*Tuple[float, ...]], float], params: Tuple[float, ...]) -> Tuple[float, float]:

"""

Проверяет непрерывное распределение на соответствие теоретическому с использованием критерия Колмогорова.

Параметры:

- data (List[float]): Массив значений случайных величин.

- theoretical\_distribution\_func (Callable[[float], float]): Функция теоретического распределения.

- params (Tuple[float, ...]): Параметры теоретического распределения.

Возвращает:

- ks\_statistic (float): Значение статистики критерия Колмогорова.

- p\_value (float): p-значение для критерия Колмогорова.

"""

theoretical\_values = [

theoretical\_distribution\_func(x, \*params) for x in data]

ks\_statistic, p\_value = ks\_2samp(data, theoretical\_values)

return ks\_statistic, p\_value

def get\_cdf\_points(data: List[float], theoretical\_distribution\_func: Callable[[float, \*Tuple[Any, ...]], float], params: Tuple[float, ...], bins: int = 100) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:

"""

Возвращает точки для построения графика эмпирической и теоретической функций распределения (CDF).

Параметры:

- data (List[float]): Массив значений случайных величин.

- theoretical\_distribution\_func (Callable[[float], float]): Функция теоретического распределения (CDF).

- params (Tuple[float, ...]): Параметры теоретического распределения.

- bins (int): Количество интервалов для гистограммы (по умолчанию 100).

Возвращает:

- empirical\_points (Tuple[np.ndarray, np.ndarray]): Кортеж из двух массивов: значения и вероятности эмпирической функции распределения (CDF).

- theoretical\_points (Tuple[np.ndarray, np.ndarray]): Кортеж из двух массивов: значения и вероятности теоретической функции распределения (CDF).

"""

# Рассчитываем эмпирическую функцию распределения (ECDF)

empirical\_values, empirical\_points = np.histogram(

data, bins=bins, density=True)

empirical\_cumulative = np.cumsum(empirical\_values)

# Рассчитываем значения теоретической функции распределения

theoretical\_values = [theoretical\_distribution\_func(

x, \*params) for x in empirical\_points]

theoretical\_cumulative = np.cumsum(theoretical\_values)

return empirical\_points[:-1], empirical\_cumulative, theoretical\_cumulative

main.py

from matplotlib import pyplot as plt

import reader as reader

import distribution\_params as dparam

import distribution\_law as dlaw

import approximation as aprx

import ploter as ploter

FILE\_PATH = "1.txt"

rnd\_values = reader.read\_random\_variables(FILE\_PATH)

print("Числовые параметры распределения:")

print()

print(f"Sample size = {dparam.sample\_size(rnd\_values)}")

print(f"Xmax = {dparam.calculate\_Xmax(rnd\_values)}")

print(f"Xmin = {dparam.calculate\_Xmin(rnd\_values)}")

print(f"Mean = {dparam.calculate\_mean(rnd\_values)}")

print(f"Freedom = {dparam.degrees\_of\_freedom(rnd\_values)}")

print(f"Variance = {dparam.calculate\_variance(rnd\_values)}")

print(f"RMS = {dparam.calculate\_std\_dev(rnd\_values)}")

print(f"Skewness = {dparam.calculate\_skewness(rnd\_values)}")

print(f"Kurtosis = {dparam.calculate\_kurtosis(rnd\_values)}")

print(f"Median = {dparam.calculate\_median(rnd\_values)}")

print(f"Range = {dparam.calculate\_range(rnd\_values)}")

print()

print(f"Дисперсия ассимтерии = {dparam.skewness\_variance(rnd\_values)}")

print(f"Дисперсия эксцесса = {dparam.kurtosis\_variance(rnd\_values)}")

print(f"Interval mean = {dparam.confidence\_interval\_mean(rnd\_values)}")

print(f"Interval variance = {dparam.confidence\_interval\_variance(rnd\_values)}")

print(f"Interval skewness = {dparam.confidence\_interval\_skewness(rnd\_values)}")

print(f"Interval kurtosis = {dparam.calculate\_kurtosis(rnd\_values)}")

print()

print()

print("Закон распределения:")

ploter.plot\_histogram(rnd\_values)

print(f"Простейший тест: {dlaw.normality\_test\_q(rnd\_values)}")

print(f"Тест Жарка-Бера: {dlaw.normality\_test\_jarque\_bera(rnd\_values)}")

print(f"Гистограмма распредления: {dlaw.calculate\_histogram(rnd\_values)}")

ploter.plot\_all\_distributions(rnd\_values)

ploter.plot\_empirical\_vs\_theoretical\_cdf(rnd\_values)

x\_data, y\_data = aprx.generate\_random\_data()

plt.scatter(x\_data, y\_data, label="Original Data")

plt.title("Original Data")

plt.xlabel("X values")

plt.ylabel("Y values")

plt.legend()

plt.show()

power\_coefficients, coeffs = aprx.approximate\_polynomial(x\_data, y\_data, 3)

ploter.plot\_data\_and\_approximation(x\_data, y\_data, power\_coefficients)

print()

print(f"Coefs: {coeffs}")

approx\_points = aprx.approximate\_basis\_functions(x\_data, y\_data, 4)

ploter.plot\_data\_and\_approximation(x\_data, y\_data, approx\_points)