ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет программной инженерии и компьютерной техники

Дисциплина:
«Вычислительная математика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
Вариант 10.

Выполнил:
Студент гр. P32151
Понамарев Степан Андреевич

Проверил:
Машина Екатерина Алексеевна

Санкт-Петербург
2023г.

1. **Цель лабораторной работы**: найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.
2. **Задание лабораторной работы**:

Вычислительная реализация задачи

Вычислительная часть лабораторной работы должна быть представлена только в отчете.

Задание:

1. Сформировать таблицу табулирования заданной функции на указанном интервале (см. табл. 1)
2. Построить линейное и квадратичное приближения по 11 точкам заданного интервала;
3. Найти среднеквадратические отклонения для каждой аппроксимирующей функции. Ответы дать с тремя знаками после запятой;
4. Выбрать наилучшее приближение;
5. Построить графики заданной функции, а также полученные линейное и квадратичное приближения;
6. Привести в отчете **подробные вычисления**.

3. **Рабочие формулы**:
   1. $y = \frac{18x}{x^4+10}$, при $x \in [0,4]$ $h = 0{,}4$ – *уравнение по заданию.*
   2. $S = \sum_{i=1}^{n} \varepsilon_i^2$ – *мера отклонения.*
   3. $\delta = \sqrt{\frac{\sum_{i=1}^{11}[\phi(x_i)-y_i]^2}{n}}$ – *СКО.*
   4. $n = 11$ – *число точек.*
4. **Вычислительная реализация задачи (вариант 10)**:
   1. Таблица табулирования заданной функции на указанном интервале.

0	0,4	0,8	1,2	1,6	2	2,4	2,8	3,2	3,6	4
0	0,72	1,38	1,79	1,74	1,38	1,00	0,71	0,50	0,36	0,27

   2. Вспомогательные обозначения

$$\boldsymbol{SX} = \sum_{i=1}^{11} x_i = 22,\ \boldsymbol{SXX} = \sum_{i=1}^{11} x_i^2 = 61{,}6,\ \boldsymbol{SY} = \sum_{i=1}^{11} y_i = 9{,}86,$$

$$\boldsymbol{SXY} = \sum_{i=1}^{111} x_i y_i = 17{,}468,\ \boldsymbol{SXXX} = \sum_{i=1}^{11} x_i^3 = 193{,}6,\ \boldsymbol{SXXY} = \sum_{i=1}^{11} x_i^2 y_i = 39{,}046,$$

$$\boldsymbol{SXXXX} = \sum_{i=1}^{11} x_i^4 = 648{,}525.$$

   3. Линейное приближение

Нахождение коэффициентов линейного уравнения сводится к решению системы:

$$\begin{cases} an + bSX = SY \\ aSX + bSXX = SXY \end{cases}$$

где a, b – коэффициенты из формулы $y = a + bx$

Из неё находим:

$$\begin{cases} 11a + 22b = 9{,}86 \\ 22a + 61{,}6\,b = 17{,}468 \end{cases}$$

$$\Delta = SXX \cdot n - SX^2 = 61{,}6 \cdot 11 - 22^2 = 193{,}6$$

$$\Delta_1 = SXY \cdot n - SX \cdot SY = 17{,}468 \cdot 11 - 22 \cdot 9{,}857 = -24{,}707$$

$$\Delta_2 = SXX \cdot SY - SX \cdot SXY = 61{,}6 \cdot 9{,}857 - 22 \cdot 17{,}468 = 222{,}897$$

$$a = \frac{\Delta_2}{\Delta} = 1{,}1513, \qquad b = \frac{\Delta_1}{\Delta} = -0{,}1276$$

Расчёт СКО:

x	0,000	0,400	0,800	1,200	1,600	2,000	2,400	2,800	3,200	3,600	4,000
y	0,000	0,718	1,383	1,789	1,740	1,385	1,001	0,705	0,501	0,364	0,271
$\varphi(x)$	1,151	1,100	1,049	0,998	0,947	0,896	0,845	0,794	0,743	0,692	0,641
ε	1,325	0,146	0,112	0,625	0,628	0,239	0,024	0,008	0,058	0,107	0,137

$$S = \sum_{i=1}^{11} \varepsilon^2 = \sum_{i=1}^{11} (\varphi(x_i) - y_i)^2 = \sum_{i=1}^{11} (a + bx_i - y_i)^2 = 3{,}41$$

$$\delta = \sqrt{\frac{S}{n}} = \sqrt{\frac{3{,}41}{11}} = 0{,}557 - \text{СКО}$$

4. Квадратичное приближение

Нахождение коэффициентов квадратичного уравнения сводится к решению системы:

$$\begin{cases} an + bSX + cSXX = SY \\ aSX + bSXX + cSXXX = SXY \\ aSXX + bSXXX + cSXXXX = SXXY \end{cases}$$

где a, b, c – коэффициенты из формулы $y = a + bx + cx^2$

Из неё находим:

$$\begin{cases} 11a + 22b + 61{,}6c = 9{,}86 \\ 22a + 61{,}6b + 193{,}6c = 17{,}468 \\ 61{,}6a + 193{,}6b + 648{,}525c = 39{,}046 \end{cases} =$$

$$\begin{cases} a = 0{,}368 \\ b = 1{,}178 \\ c = -0{,}326 \end{cases}$$

Расчёт СКО:

x	0,000	0,400	0,800	1,200	1,600	2,000	2,400	2,800	3,200	3,600	4,000
y	0,000	0,718	1,383	1,789	1,740	1,385	1,001	0,705	0,501	0,364	0,271
$\varphi(x)$	0,368	0,787	1,102	1,312	1,418	1,420	1,317	1,111	0,799	0,384	-0,136
ε	0,135	0,005	0,079	0,227	0,103	0,001	0,100	0,164	0,089	0,000	0,165

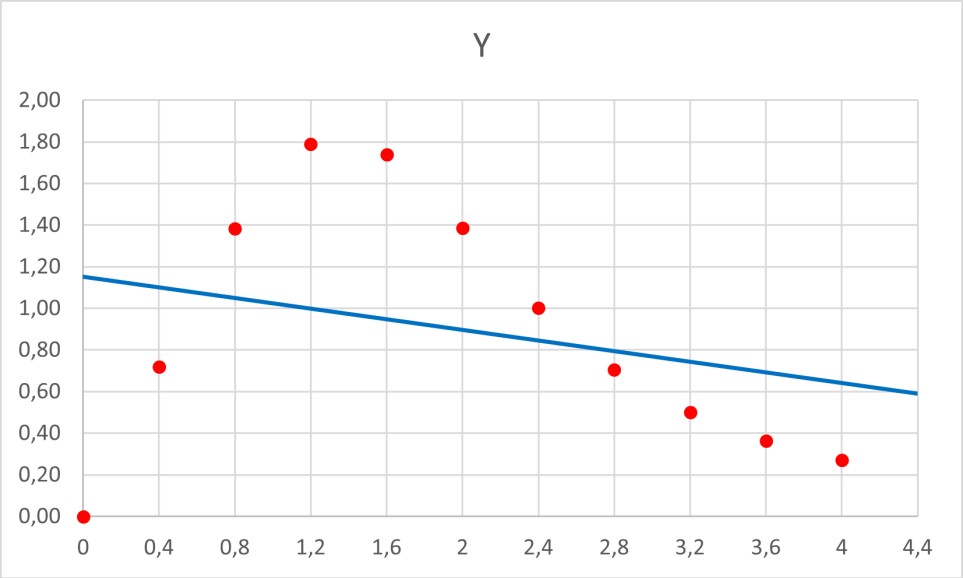
$$S = \sum_{i=1}^{11} \varepsilon^2 = \sum_{i=1}^{11} (\varphi(x_i) - y_i)^2 = \sum_{i=1}^{11} (a + bx_i + cx_i^2 - y_i)^2 = 1{,}071$$

$$\delta = \sqrt{\frac{S}{n}} = \sqrt{\frac{1{,}071}{11}} = 0{,}312 - \text{СКО}$$

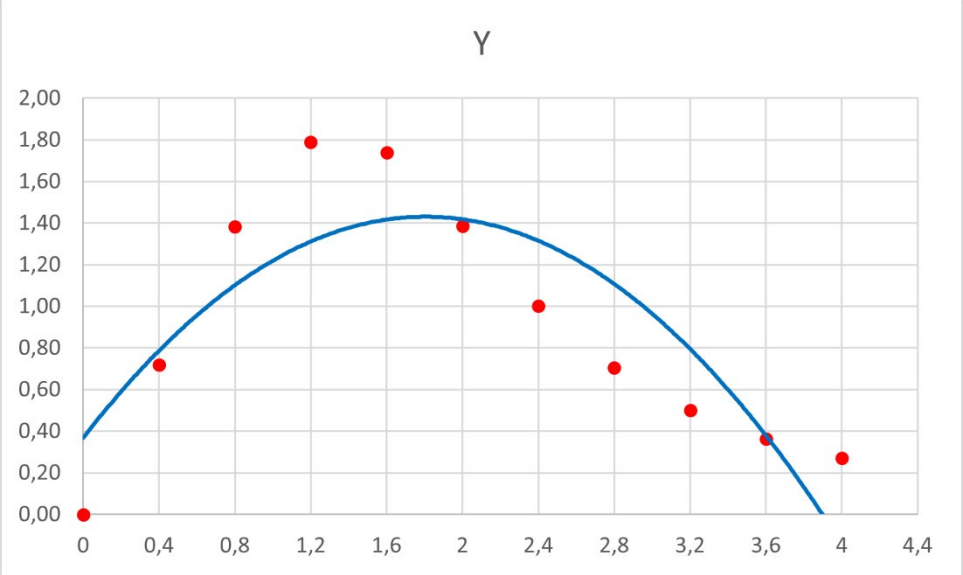
Наилучшее приближение является квадратичным с СКО равным $\delta = 0{,}312$.

5. Графики

1. Линейное приближение:



2. Квадратичное приближение:



5. Листинг программы

main.py:

```
from InputManager import InputManager
from OLS import *

output_result = ""


def my_print(*args):
    global output_result
```

```
    print(*args)
    for i in args:
        output_result += str(i) + " "
    output_result += '\n'


if __name__ == '__main__':

    seriesManager = SeriesManager()
    # ввод исходных данных
    seriesManager.enter_points()
    seriesManager.print_points()

    # выбор функции и решение

    variants = ["Линейная функция", "Полиномиальная функция 2-й степени",
"Полиномиальная 3-й степени",
                "Экспоненциальная фунция", "Логарифмическая функция", "Степенная
функция"]
    values = [LinearFunction, PolynomialFunction2ndDegree,
PolynomialFunction3rdDegree,
              ExponentialFunction, LogarithmicFunction, PowerFunction]
    results = []

    best_approximation = None
    for i in range(6):
        my_print(variants[i] + ":")
        function = values[i](seriesManager)
        if function.broken_flag:
            my_print("В аппроксимации учтены не все точки. Решение учитываться не
будет.")
            my_print()
        else:
            result = function.solve()
            if function.mse is not None:
                if best_approximation is None:
                    best_approximation = function
                elif best_approximation.mse > function.mse:
                    best_approximation = function
            my_print(result)
            if InputManager.yes_or_no_input("Нарисовать график функции?"):
                function.draw_graphic(seriesManager)
            print()

    if best_approximation is not None:
        my_print("Наилучшее приближение:", best_approximation.method_name)
        my_print("СКО:", round(best_approximation.mse, 4))
    else:
        my_print("Наилучшего приближения не существует.")

    if InputManager.yes_or_no_input("Сохранить результаты в файл?"):
        try:
            filename = InputManager.string_input("Введите имя файла: ")
            with open(filename, "w", encoding='utf-8') as f:
                f.write(output_result)
                f.flush()
        except Exception:
            print("Не удалось сохранить результаты в файл.")
```

InputManager.py:

```
class InputManager:
    @staticmethod
    def string_input(message=""):
        buf = ""
        while buf == "":
            buf = input(message).strip()
        return buf

    @staticmethod
```

```
    def _check_number(buf):
        try:
            float(buf.replace(',', '.'))
            return True
        except ValueError:
            return False

    @staticmethod
    def _convert_to_number(num):
        try:
            return float(num.replace(',', '.'))
        except ValueError:
            return None

    @staticmethod
    def float_input(message=""):
        number = None
        while number is None:
            number =
InputManager._convert_to_number(InputManager.string_input(message))
        return number

    @staticmethod
    def int_input(message=""):
        return int(InputManager.float_input(message))

    @staticmethod
    def yes_or_no_input(message=""):
        answer = "0"
        while answer[0].lower() not in ["y", "n", "д", "н"]:
            answer = InputManager.string_input(message + " [y/n]: ")
        return answer[0].lower() in ["y", "д"]

    @staticmethod
    def enum_input(variants_list, message=""):
        variants_list = [str(i) for i in variants_list]
        buf = ""
        while buf not in variants_list:
            buf = InputManager.string_input(message)
        return buf

    @staticmethod
    def multiple_choice_input(variant_list, values_list, message=""):
        n = len(variant_list)
        if n == 0 or len(variant_list) != len(values_list):
            raise ValueError

        if message != "":
            print(message)

        for i in range(n):
            s = f"{i + 1}."
            lines = variant_list[i].split('\n')
            print('\t' + s, lines[0])
            for line in lines[1:]:
                print("\t" + ' ' * len(s), line)

        i = int(InputManager.enum_input([*range(1, n + 1)], f"Введите число от 1 до
{n}: "))
        return values_list[i - 1]

    @staticmethod
    def epsilon_input(message=""):
        e = InputManager.float_input(message)
        while not (0 < e and e <= 1):
            print("Эпсилон должно быть в промежутке от 0 до 1!")
            e = InputManager.float_input(message)
        return e

    @staticmethod
```

```
    def int_input_with_borders(left, right, message=""):
        if left >= right:
            raise ValueError
        if message != "":
            print(message)
        i = int(InputManager.enum_input([*range(left, right + 1)], f"Введите число от
{left} до {right}: "))
        return i

    @staticmethod
    def point_input(message=""):
        x, y = None, None
        while x is None or y is None:
            line = InputManager.string_input(message).split()
            if len(line) != 2:
                continue
            x, y = InputManager._convert_to_number(line[0]),
InputManager._convert_to_number(line[1])
        return x, y
```

series.py:

```
from InputManager import InputManager


class SeriesManager():
    def __init__(self):
        self.n = 0
        self.X = []
        self.Y = []

    def drop_context(self):
        self.n = 0
        self.X = []
        self.Y = []

    def enter_points(self):
        self.drop_context()
        while True:
            if InputManager.yes_or_no_input("Хотите считать точки из файла?"):
                filename = InputManager.string_input("Введите имя файла: ")
                try:
                    self._enter_points_from_file(filename)
                    break
                except FileNotFoundError:
                    print("Файл не найден.")
                except Exception:
                    print("Произошла ошибка во время считывания. Проверьте содержание
файла.")
            else:
                self._enter_points_from_console()
                break

    def _enter_points_from_file(self, filename):
        file = open(filename, "r")
        self.n = int(file.readline())
        if self.n < 8 or self.n > 12:
            print("Необходимо ввести не менее 8 и не более 12 точек.")
            raise ValueError
        for i in range(self.n):
            x, y = map(float, file.readline().split())
            self.X.append(x)
            self.Y.append(y)
        file.close()

    def _enter_points_from_console(self):
        self.n = InputManager.int_input_with_borders(8, 12, "Введите число точек от 8
до 12: ")
        print("Теперь введите x y координаты точек через пробел.")
        for i in range(self.n):
            x, y = InputManager.point_input(f"Координаты точки {i + 1}: ")
```

```
            self.X.append(x)
            self.Y.append(y)

    def set_n(self, n):
        self.n = n

    def set_x(self, x):
        self.X = x
        self.n = min(len(x), self.n)

    def set_y(self, y):
        self.Y = y
        self.n = min(len(y), self.n)

    def set_xy(self, x, y):
        self.set_n(min(len(x), len(y)))
        self.set_x(x)
        self.set_y(y)

    def print_points(self):
        print("Исходные точки: ", end="")
        print(*[f"({self.X[i]}, {self.Y[i]})" for i in range(self.n)], sep=", ")

    def copy(self):
        sm = SeriesManager()
        sm.set_n(self.n)
        sm.set_x(self.X)
        sm.set_y(self.Y)
        return sm
```

OLS.py:

```
# OLS means Ordinary least squares

import matplotlib.pyplot as plt
import numpy as np

from series import SeriesManager

np.set_printoptions(formatter={'float_kind': "{:.2f}".format})


class Function:

    def __init__(self, ser):
        self.series = ser
        self.mse = None
        self.phi = lambda x: x * 0
        self.method_name = None

    def get_epsilon_list(self, phi_x):
        result = np.array(self.series.Y) - phi_x
        return result

    def draw_graphic(self, series=None):
        x = np.linspace(min(self.series.X), max(self.series.X), 100)
        ax = plt.subplot()
        if series is None:
            ax.scatter(self.series.X, self.series.Y, c="r")
        else:
            ax.scatter(series.X, series.Y, c="r")
        ax.plot(x, self.phi(x))
        ax.set_title(self.method_name)
        plt.show()

    def calculate(self, points):
        return self.phi(np.array(points))

    def solve(self):
        return None
```

```
    def check_series(self, series, condition=lambda x, y: x > 0 and y > 0):
        for i in range(series.n):
            if not condition(series.X[i], series.Y[i]):
                return False
        return True


class LinearFunction(Function):
    def __init__(self, series):
        self.broken_flag = False
        super().__init__(series.copy())
        self.method_name = "Линейная функция."

    def get_pearson_correlation_coefficient(self):
        X, Y, n = np.array(self.series.X), np.array(self.series.Y), self.series.n
        x_mean = sum(X) / n
        y_mean = sum(Y) / n
        result = sum([(X[i] - x_mean) * (Y[i] - y_mean) for i in range(n)])
        result /= np.sqrt(sum((X - x_mean) ** 2) * sum((Y - y_mean) ** 2))
        return result

    def define_linear_relationship(self, r):
        r = round(r, 3)
        if r == 0:
            return "отсутствует"
        elif r < 0.3:
            return "слабая"
        elif r < 0.5:
            return "умеренная"
        elif r < 0.7:
            return "заметная"
        elif r < 0.9:
            return "высокая"
        elif r < 0.99:
            return "весьма высокая"
        else:
            return "прямая"

    def get_a_b(self):
        sx = sum(self.series.X)
        sy = sum(self.series.Y)
        sxx = sum([x ** 2 for x in self.series.X])
        sxy = sum([self.series.X[i] * self.series.Y[i] for i in range(self.series.n)])
        delta = sxx * self.series.n - sx * sx
        delta1 = sxy * self.series.n - sx * sy
        delta2 = sxx * sy - sx * sxy
        a = delta1 / delta
        b = delta2 / delta
        return a, b

    def solve(self):
        a, b = self.get_a_b()
        self.phi = lambda x: a * x + b
        p = self.calculate(self.series.X)
        epsilon_list = self.get_epsilon_list(p)
        self.mse = sum([e ** 2 for e in epsilon_list]) / self.series.n
        r = self.get_pearson_correlation_coefficient()
        result = \
            f"""    Формула функции: phi(x) = ax + b
    Коэффициенты: a={round(a, 4)}, b={round(b, 4)}
    Значение функции в точках: {p.round(4)}
    Отклонения: {epsilon_list.round(4)}
    Сумма отклонений: {round(sum(epsilon_list), 4)}
    Коэффициенты: [{round(a, 4)} {round(b, 4)}]
    Среднеквадратичное отклонение: {np.sqrt(round(self.mse, 4))}
    Коэффициент корреляции: {round(r, 2)}
    Линейная зависимость: {self.define_linear_relationship(r)}\n"""
        return result
```

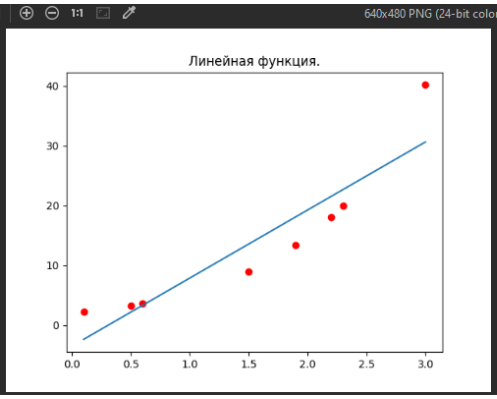
```
class PolynomialFunction2ndDegree(Function):

    def __init__(self, series):
        self.broken_flag = False
        super().__init__(series.copy())
        self.method_name = "Полиномиальная функция 2-й степени."

    def get_solution(self):
        x = np.array(self.series.X)
        y = np.array(self.series.Y)
        sx1, sx2, sx3, sx4 = sum(x), sum(x ** 2), sum(x ** 3), sum(x ** 4)
        matrix = np.array(
            [[self.series.n, sx1, sx2],
             [sx1, sx2, sx3],
             [sx2, sx3, sx4]]
        )
        b = np.array([sum(y), sum(x * y), sum(x ** 2 * y)])
        return np.linalg.solve(matrix, b)

    def solve(self):
        solution = self.get_solution()
        self.phi = lambda x: solution[0] + solution[1] * x + solution[2] * x ** 2
        p = self.calculate(self.series.X)
        epsilon_list = self.get_epsilon_list(p)
        self.mse = sum([e ** 2 for e in epsilon_list]) / self.series.n
        result = \
            f"""    Формула функции: phi(x) = a0 + a1*x + a2*x^2
    Коэффициенты: {solution.round(4)}
    Значение функции в точках: {p.round(4)}
    Отклонения: {epsilon_list.round(4)}
    Сумма отклонений: {round(sum(epsilon_list), 4)}
    Среднеквадратичное отклонение: {np.sqrt(round(self.mse, 4))}\n"""
        return result


class PolynomialFunction3rdDegree(Function):

    def __init__(self, series):
        self.broken_flag = False
        super().__init__(series.copy())
        self.method_name = "Полиномиальная функция 3-й степени."

    def get_solution(self):
        x = np.array(self.series.X)
        y = np.array(self.series.Y)
        sx1, sx2, sx3, sx4, sx5, sx6 = sum(x), sum(x ** 2), sum(x ** 3), sum(x ** 4),
sum(x ** 5), sum(x ** 6)
        matrix = np.array(
            [[self.series.n, sx1, sx2, sx3],
             [sx1, sx2, sx3, sx4],
             [sx2, sx3, sx4, sx5],
             [sx3, sx4, sx5, sx6]]
        )
        b = np.array([sum(y), sum(x * y), sum(x ** 2 * y), sum(x ** 3 * y)])
        return np.linalg.solve(matrix, b)

    def solve(self):
        solution = self.get_solution()
        self.phi = lambda x: solution[0] + solution[1] * x + solution[2] * x ** 2 +
solution[3] * x ** 3
        p = self.calculate(self.series.X)
        epsilon_list = self.get_epsilon_list(p)
        self.mse = sum([e ** 2 for e in epsilon_list]) / self.series.n
        result = \
            f"""    Формула функции: phi(x) = a0 + a1*x + a2*x^2 + a3*x^3
    Коэффициенты: {solution.round(4)}
    Значение функции в точках: {p.round(4)}
    Отклонения: {epsilon_list.round(4)}
    Сумма отклонений: {round(sum(epsilon_list), 4)}
```

```
    Среднеквадратичное отклонение: {round(np.sqrt(self.mse), 4)}\n"""
        return result


class ExponentialFunction(Function):

    def __init__(self, series):
        ser = series.copy()
        condition = lambda x, y: y > 0
        self.broken_flag = False
        if not self.check_series(series=ser, condition=condition):
            self.broken_flag = True
            x, y = [], []
            for i in range(series.n):
                if series.Y[i] > 0:
                    x.append(series.X[i])
                    y.append(series.Y[i])
            ser.set_xy(x, y)
        super().__init__(ser)
        self.method_name = "Экспоненциальная функция."

    def solve(self):
        x = np.array(self.series.X)
        y = np.log(np.array(self.series.Y))
        ser = self.series.copy()
        ser.set_xy(x, y)

        linf = LinearFunction(ser)
        b, a = linf.get_a_b()
        a = np.exp(a)

        self.phi = lambda x: a * np.exp(b * x)
        p = self.calculate(self.series.X)
        epsilon_list = self.get_epsilon_list(p)
        self.mse = sum([e ** 2 for e in epsilon_list]) / self.series.n
        result = \
            f"""    Формула функции: phi(x) = a * exp(bx)
    Коэффициенты: a={round(a, 4)}, b={round(b, 4)}
    Значение функции в точках: {p.round(4)}
    Отклонения: {epsilon_list.round(4)}
    Сумма отклонений: {round(sum(epsilon_list), 4)}
    Среднеквадратичное отклонение: {round(np.sqrt(self.mse), 4)}\n"""
        return result


class LogarithmicFunction(Function):
    def __init__(self, series):
        condition = lambda x, y: x > 0
        ser = series.copy()
        self.broken_flag = False
        if not self.check_series(series, condition):
            self.broken_flag = True
            x, y = [], []
            for i in range(series.n):
                if series.X[i] > 0:
                    x.append(series.X[i])
                    y.append(series.Y[i])
            ser = SeriesManager()
            ser.set_xy(x, y)
        super().__init__(ser)
        self.method_name = "Логарифмическая функция."

    def solve(self):
        x = np.log(np.array(self.series.X))
        y = np.array(self.series.Y)
        ser = self.series.copy()
        ser.set_xy(x, y)
        linf = LinearFunction(ser)
        a, b = linf.get_a_b()
        # a = np.exp(a)
```

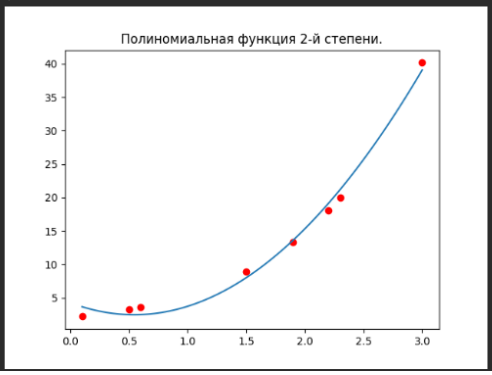
```
        self.phi = lambda x: a * np.log(x) + b
        p = self.calculate(self.series.X)
        epsilon_list = self.get_epsilon_list(p)
        self.mse = sum([e ** 2 for e in epsilon_list]) / self.series.n
        result = \
            f"""    Формула функции: phi(x) = a * ln(x) + b
    Значение функции в точках: {p.round(4)}
    Отклонения: {epsilon_list.round(4)}
    Сумма отклонений: {round(sum(epsilon_list), 4)}
    Среднеквадратичное отклонение: {round(np.sqrt(self.mse), 4)}\n"""
        return result


class PowerFunction(Function):

    def __init__(self, series):
        condition = lambda x, y: x > 0 and y > 0
        ser = series.copy()
        self.broken_flag = False
        if not self.check_series(series, condition):
            self.broken_flag = True
            x, y = [], []
            for i in range(series.n):
                if series.X[i] > 0 and series.Y[i] > 0:
                    x.append(series.X[i])
                    y.append(series.Y[i])
            ser = SeriesManager()
            ser.set_xy(x, y)
        super().__init__(ser)
        self.method_name = "Степенная функция."

    def solve(self):
        x = np.log(np.array(self.series.X))
        y = np.log(np.array(self.series.Y))
        ser = self.series.copy()
        ser.set_xy(x, y)
        linf = LinearFunction(ser)
        b, a = linf.get_a_b()
        a = np.exp(a)
        self.phi = lambda x: a * x ** b
        p = self.calculate(self.series.X)
        epsilon_list = self.get_epsilon_list(p)
        self.mse = sum([e ** 2 for e in epsilon_list]) / self.series.n
        result = \
            f"""    Формула функции: phi(x) = a * x^b
    Коэффициенты: a={round(a, 4)}, b={round(b, 4)}
    Значение функции в точках: {p.round(4)}
    Отклонения: {epsilon_list.round(4)}
    Сумма отклонений: {round(sum(epsilon_list), 4)}
    Среднеквадратичное отклонение: {round(np.sqrt(self.mse), 4)}\n"""
        return result
```

6. Пример работы программы.

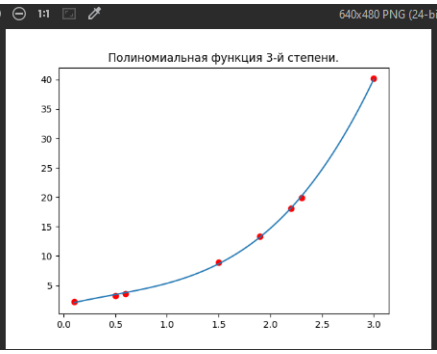
```
Хотите считать точки из файла? [y/n]: y
Введите имя файла: examples/exponential.txt
Исходные точки: (0.1, 2.21), (0.5, 3.29), (0.6, 3.64), (1.5, 8.96), (1.9,
 13.37), (2.2, 18.05), (2.3, 19.95), (3.0, 40.17)
Линейная функция:
    Формула функции: phi(x) = ax + b
    Коэффициенты: a=11.3792, b=-3.506
    Значение функции в точках: [-2.37 2.18 3.32 13.56 18.11 21.53 22.67 30.63]
    Отклонения: [4.58 1.11 0.32 -4.60 -4.74 -3.48 -2.72 9.54]
    Сумма отклонений: -0.0
    Коэффициенты: [11.3792 -3.506]
    Среднеквадратичное отклонение: 4.696232532573318
    Коэффициент корреляции: 0.92
    Линейная зависимость: весьма высокая

Нарисовать график функции? [y/n]: y
```



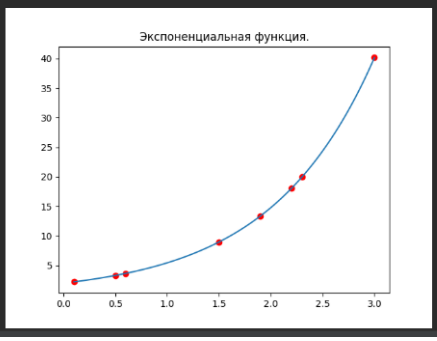

```
Полиномиальная функция 2-й степени:
    Формула функции: phi(x) = a0 + a1*x + a2*x^2
    Коэффициенты: [4.27 -6.61 6.07]
    Значение функции в точках: [3.67 2.48 2.49 8.01 13.63 19.11 21.18 39.08]
    Отклонения: [-1.46 0.81 1.15 0.95 -0.26 -1.06 -1.23 1.09]
    Сумма отклонений: 0.0
    Среднеквадратичное отклонение: 1.0548933595392476

Нарисовать график функции? [y/n]: y

Полиномиальная 3-й степени:
    Формула функции: phi(x) = a0 + a1*x + a2*x^2 + a3*x^3
    Коэффициенты: [1.66 4.31 -2.35 1.73]
    Значение функции в точках: [2.07 3.45 3.78 8.68 13.23 18.17 20.17 40.09]
    Отклонения: [0.14 -0.16 -0.14 0.28 0.14 -0.12 -0.22 0.08]
    Сумма отклонений: 0.0
```




```
Полиномиальная 3-й степени:
    Формула функции: phi(x) = a0 + a1*x + a2*x^2 + a3*x^3
    Коэффициенты: [1.66 4.31 -2.35 1.73]
    Значение функции в точках: [2.07 3.45 3.78 8.68 13.23 18.17 20.17 40.09]
    Отклонения: [0.14 -0.16 -0.14 0.28 0.14 -0.12 -0.22 0.08]
    Сумма отклонений: 0.0
    Среднеквадратичное отклонение: 0.1712

Нарисовать график функции? [y/n]: y

Экспоненциальная фунция:
    Формула функции: phi(x) = a * exp(bx)
    Коэффициенты: a=1.9975, b=1.0005
    Значение функции в точках: [2.21 3.29 3.64 8.96 13.37 18.05 19.95 40.18]
    Отклонения: [0.00 -0.00 -0.00 0.00 0.00 0.00 0.00 -0.01]
```

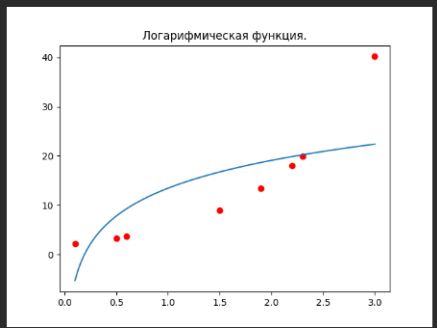



```
Экспоненциальная фунция:
    Формула функции: phi(x) = a * exp(bx)
    Коэффициенты: a=1.9975, b=1.0005
    Значение функции в точках: [2.21 3.29 3.64 8.96 13.37 18.05 19.95 40.18]
    Отклонения: [0.00 -0.00 -0.00 0.00 0.00 0.00 0.00 -0.01]
    Сумма отклонений: -0.0017
    Среднеквадратичное отклонение: 0.0044

Нарисовать график функции? [y/n]: y

Логарифмическая функция:
    Формула функции: phi(x) = a * ln(x) + b
    Значение функции в точках: [-5.31 7.79 9.28 16.74 18.67 19.86 20.22 22.39]
    Отклонения: [7.52 -4.50 -5.64 -7.78 -5.30 -1.81 -0.27 17.78]
```



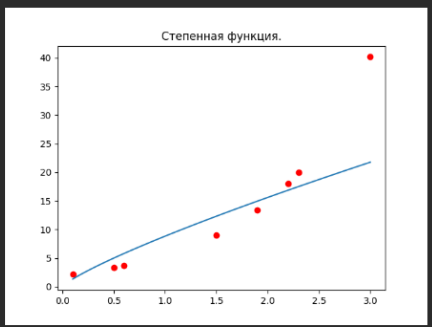
```
Логарифмическая функция:
    Формула функции: phi(x) = a * ln(x) + b
    Значение функции в точках: [-5.31 7.79 9.28 16.74 18.67 19.86 20.22 22.39]
    Отклонения: [7.52 -4.50 -5.64 -7.78 -5.30 -1.81 -0.27 17.78]
    Сумма отклонений: 0.0
    Среднеквадратичное отклонение: 8.0382

Нарисовать график функции? [y/n]: y

Степенная функция:
    Формула функции: phi(x) = a * x^b
    Коэффициенты: a=8.832, b=0.821
    Значение функции в точках: [1.33 5.00 5.81 12.32 14.96 16.87 17.50 21.77]
    Отклонения: [0.88 -1.71 -2.17 -3.36 -1.59 1.18 2.45 18.40]
    Сумма отклонений: 14.0815
```

640x480 PNG (24-bit color) 15,6 kB




```
Степенная функция:
    Формула функции: phi(x) = a * x^b
    Коэффициенты: a=8.832, b=0.821
    Значение функции в точках: [1.33 5.00 5.81 12.32 14.96 16.87 17.50 21.77]
    Отклонения: [0.88 -1.71 -2.17 -3.36 -1.59 1.18 2.45 18.40]
    Сумма отклонений: 14.0815
    Среднеквадратичное отклонение: 6.7851

Нарисовать график функции? [y/n]: y

Наилучшее приближение: Экспоненциальная функция.
СКО: 0.0
Сохранить результаты в файл? [y/n]: y
Введите имя файла: asd.txt
```

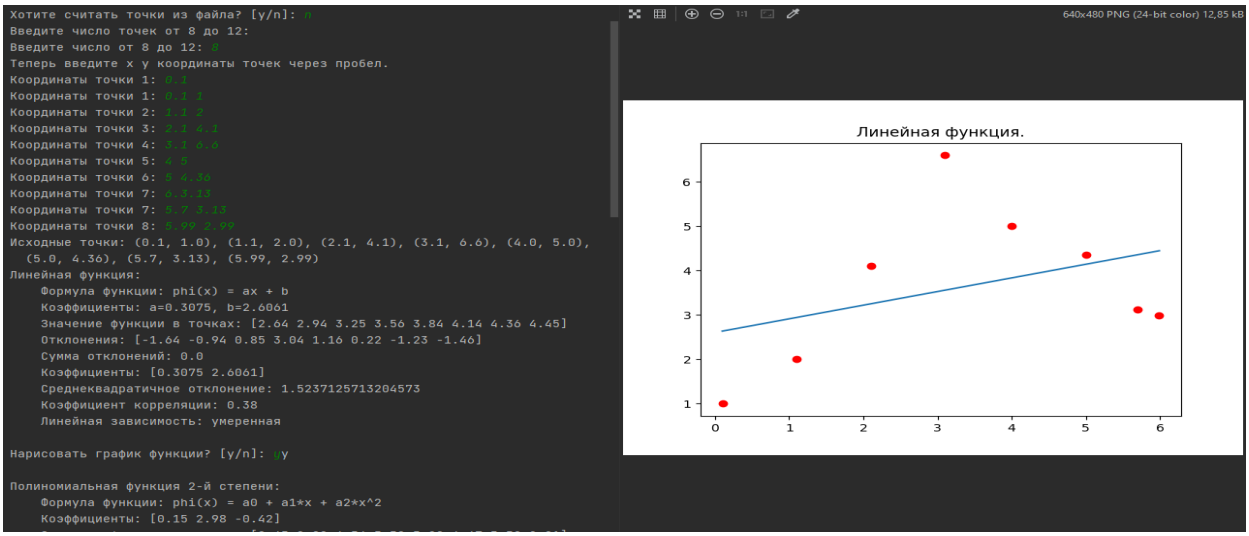
640x480 PNG (24-bit color) 15,09 kB

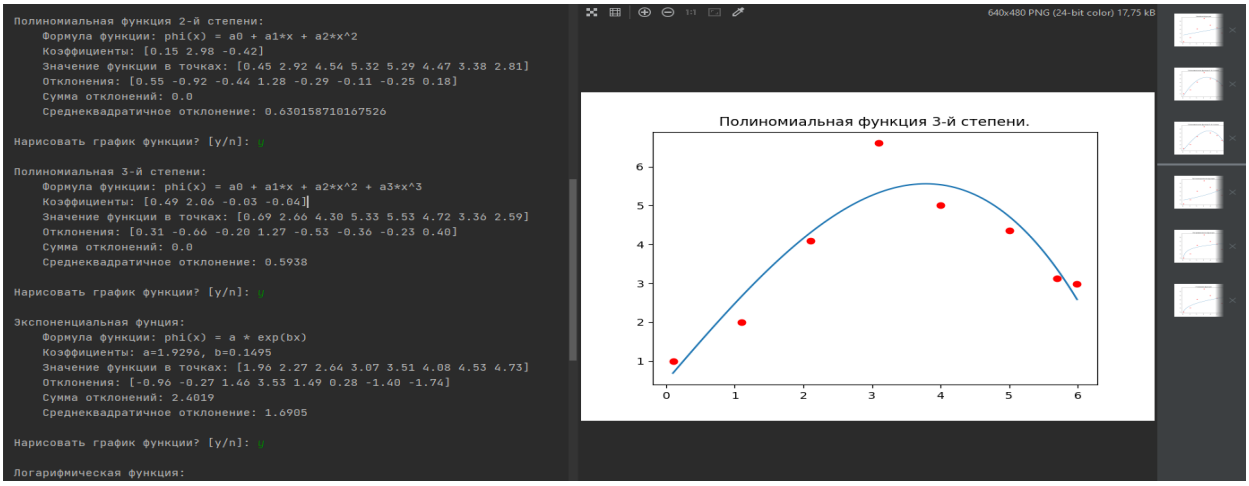


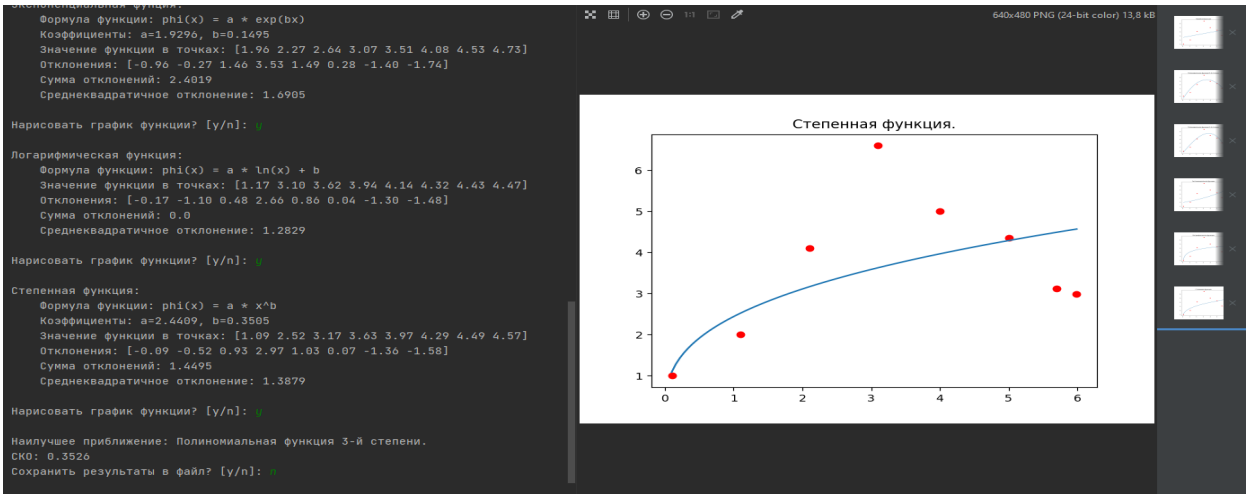
Результаты работы программы, записанные в файл:

```
Линейная функция:
    Формула функции: phi(x) = ax + b
    Коэффициенты: a=11.3792, b=-3.506
    Значение функции в точках: [-2.37 2.18 3.32 13.56 18.11 21.53 22.67 30.63]
    Отклонения: [4.58 1.11 0.32 -4.60 -4.74 -3.48 -2.72 9.54]
    Сумма отклонений: -0.0
    Коэффициенты: [11.3792 -3.506]
    Среднеквадратичное отклонение: 4.696232532573318
    Коэффициент корреляции: 0.92
    Линейная зависимость: весьма высокая

Полиномиальная функция 2-й степени:
    Формула функции: phi(x) = a0 + a1*x + a2*x^2
    Коэффициенты: [4.27 -6.61 6.07]
    Значение функции в точках: [3.67 2.48 2.49 8.01 13.63 19.11 21.18 39.08]
    Отклонения: [-1.46 0.81 1.15 0.95 -0.26 -1.06 -1.23 1.09]
    Сумма отклонений: 0.0
    Среднеквадратичное отклонение: 1.0548933595392476

Полиномиальная 3-й степени:
    Формула функции: phi(x) = a0 + a1*x + a2*x^2 + a3*x^3
    Коэффициенты: [1.66 4.31 -2.35 1.73]
    Значение функции в точках: [2.07 3.45 3.78 8.68 13.23 18.17 20.17 40.09]
    Отклонения: [0.14 -0.16 -0.14 0.28 0.14 -0.12 -0.22 0.08]
    Сумма отклонений: 0.0
    Среднеквадратичное отклонение: 0.1712

Экспоненциальная функция:
    Формула функции: phi(x) = a * exp(bx)
    Коэффициенты: a=1.9975, b=1.0005
    Значение функции в точках: [2.21 3.29 3.64 8.96 13.37 18.05 19.95 40.18]
    Отклонения: [0.00 -0.00 -0.00 0.00 0.00 0.00 0.00 -0.01]
    Сумма отклонений: -0.0017
    Среднеквадратичное отклонение: 0.0044

Логарифмическая функция:
    Формула функции: phi(x) = a * ln(x) + b
    Значение функции в точках: [-5.31 7.79 9.28 16.74 18.67 19.86 20.22 22.39]
    Отклонения: [7.52 -4.50 -5.64 -7.78 -5.30 -1.81 -0.27 17.78]
    Сумма отклонений: 0.0
    Среднеквадратичное отклонение: 8.0382

Степенная функция:
    Формула функции: phi(x) = a * x^b
    Коэффициенты: a=8.832, b=0.821
    Значение функции в точках: [1.33 5.00 5.81 12.32 14.96 16.87 17.50 21.77]
    Отклонения: [0.88 -1.71 -2.17 -3.36 -1.59 1.18 2.45 18.40]
    Сумма отклонений: 14.0815
    Среднеквадратичное отклонение: 6.7851

Наилучшее приближение: Экспоненциальная функция.
СКО: 0.0

```

7. Ещё пример работы программы:







8. Выводы

В ходе выполнения лабораторной работы я изучил и реализовал несколько методов аппроксимации функции по табличным данным: линейную, квадратичную, кубическую, логарифмическую, экспоненциальную и степенную.