МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №4

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №22

Студент:
Самарина Арина Анатольевна, Суржицкий Арсений Арсентьевич
Группа P3266

Преподаватель:
Машина Екатерина Александровна

Санкт-Петербург, 2024

1. Цель работы

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

2. Порядок выполнения работы

- 1 часть: Вычислительная реализация задачи
    1. Сформировать таблицу табулирования заданной функции на указанном интервале
       $y = \frac{28x}{x^4 + 25}, \quad x \in [0,4], \quad h = 0,4$
    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

- 2 часть: Программная реализация задачи
    1. Предусмотреть ввод исходных данных из файла/консоли (таблица $y = f(x)$должна содержать от 8 до 12 точек)
    2. Реализовать метод наименьших квадратов, исследуя все указанные функции
    3. Предусмотреть вывод результатов в файл/консоль: коэффициенты аппроксимирующих функций, среднеквадратичное отклонение, массивы значений $x_i, y_i, \varphi(x_i), \varepsilon_i$
    4. Для линейной зависимости вычислить коэффициент корреляции Пирсона
    5. Программа должна отображать наилучшую аппроксимирующую функцию
    6. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом)
    7. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных

3. Рабочие формулы

Аппроксимировать f(x) функцией $\varphi(x)$

$$\varphi(x) = a_0 + a_1 x + \cdots + a_n x^n$$

МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

Преобразуем полученную линейную систему уравнений: раскроем скобки и перенесем свободные слагаемые в правую часть выражения.

$$\begin{cases} a_0 n + a_1 \sum_{i=1}^{n} x_i + \dots + a_{m-1} \sum_{i=1}^{n} x_i^{m-1} + a_m \sum_{i=1}^{n} x_i^m = \sum_{i=1}^{n} y_i \\ a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + \dots + a_{m-1} \sum_{i=1}^{n} x_i^m + a_m \sum_{i=1}^{n} x_i^{m+1} = \sum_{i=1}^{n} x_i y_i \\ \dots\dots\dots \\ a_0 \sum_{i=1}^{n} x_i^m + a_1 \sum_{i=1}^{n} x_i^{m+1} + \dots + a_{m-1} \sum_{i=1}^{n} x_i^{2m-1} + a_m \sum_{i=1}^{n} x_i^{2m} = \sum_{i=1}^{n} x_i^m y_i \end{cases}$$

в матричном виде:

$$\begin{vmatrix} n & \sum_{i=1}^{n} x_i & \dots & \sum_{i=1}^{n} x_i^m \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \dots & \sum_{i=1}^{n} x_i^{m+1} \\ & \dots\dots\dots & & \\ \sum_{i=1}^{n} x_i^m & \sum_{i=1}^{n} x_i^{m+1} & \dots & \sum_{i=1}^{n} x_i^{2m} \end{vmatrix} \cdot \begin{vmatrix} a_0 \\ a_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ a_m \end{vmatrix} = \begin{vmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \dots \\ \sum_{i=1}^{n} x_i^m y_i \end{vmatrix}$$

Коэффициент корреляции

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Среднеквадратичное отклонение

$$\delta = \sqrt{\frac{\sum_{i=1}^{n}(\varphi(x_i) - y_i)^2}{n}}$$

Выбор аппроксимирующей функции

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \varphi_i)^2}{\sum_{i=1}^{n} \varphi_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} \varphi_i\right)^2}$$

4. Вычислительная часть

- Сформировать таблицу табулирования функции
  $y = \frac{28x}{x^4 + 25}, \quad x \in [0,4], \quad h = 0,4$

x	0.0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
y	0	0.448	0.882	1.241	1.42	1.366	1.155	0.907	0.69	0.522	0.399

- Построить линейное и квадратичное приближения по 11 точкам заданного интервала

+ Линейное приближения:

Для определения вида зависимости. Выбираем многочлен первой степени и строим линейную модель $\mathrm{P}_1(x) = ax + b$

Вычисляем суммы:

$$SX = \sum_{i=1}^{n} x_i = 22$$

$$SXX = \sum_{i=1}^{n} {x_i}^2 = 61{,}6$$

$$SY = \sum_{i=1}^{n} y_i = 9{,}03$$

$$SXY = \sum_{i=1}^{n} x_i y_i = 18{,}3728$$

Получим систему управнений для нахождения параметров a и b

$$\begin{cases} aSXX + bSX = SXY \\ aSX + bn = SY \end{cases} \rightarrow \begin{cases} 61{,}6a + 22b = 18{,}3728 \\ 22a + 11b = 9{,}03 \end{cases}$$

Решая систему, получим значения коэффициентов:

$a \approx 0{,}0178$

$b \approx 0{,}785$

Проверим правильность выбора линейной модели. Для этого вычислим значения аппроксимирующей финкции $P_1(x) = 0.0178x + 0.785$

№ п.п	1	2	3	4	5	6	7	8	9	10	11
x	0	0.4	0.8	1.2	1.6	2	2.4	2.8	3.2	3.6	4.0
y	0	0.448	0.882	1.241	1.42	1.366	1.155	0.907	0.69	0.522	0.399
$P_1(x)= ax + b$	0.785	0.79212	0.79924	0.80636	0.81348	0.8206	0.82772	0.83484	0.84196	0.84908	0.8562
ε_i	-0.785	-0.34412	0.08276	0.43464	0.60652	0.5454	0.32728	0.07216	-0.15196	-0.32708	-0.4572

Определим меру отклонения $S = \sum_{i=1}^{n} {\varepsilon_i}^2 = 2.047$

Среднеквадратичное отклонение $\delta = \sqrt{\frac{\sum_{i=1}^{n}(\varphi(x_i) - y_i)^2}{n}} = 0.4314$

+ Квадратичное приближения:

Для определения вида зависимости. Выбираем многочлен второй степени и строим линейную модель $\mathrm{P}_2(x) = a_0 + a_1x + a_2x^2$

Сумма квадратов отклонений запишется следующим образом:

$$S = \sum_{i=1}^{n} (a_0 + a_1x + a_2x^2 - y_i)^2 \rightarrow min$$

Вычислим:

$\sum_{i=1}^{n} x_i = 22$ $\quad$ $\sum_{i=1}^{n} {x_i}^2 = 61{,}6$

$\sum_{i=1}^{n} {x_i}^3 = 193{,}6$ $\quad$ $\sum_{i=1}^{n} {x_i}^4 = 648{,}5248$

$\sum_{i=1}^{n} y_i = 9{,}03$ $\quad$ $\sum_{i=1}^{n} x_i y_i = 18{,}3728$

$\sum_{i=1}^{n} {x_i}^2 y_i = 45{,}5008$

Получим систему линейных уравнений, решив которую, определим значения коэффициентов эмпирической формулы:

$$\begin{cases} 11a_0 + 22a_1 + 61{,}6a_2 = 9{,}03 \\ 22a_0 + 61{,}6a_1 + 193{,}6a_2 = 18{,}3728 \\ 61{,}6a_0 + 193{,}6a_1 + 648{,}5248a_2 = 45{,}5008 \end{cases} \rightarrow \begin{cases} a_0 = 0.095 \\ a_1 = 1.168 \\ a_2 = -0.288 \end{cases}$$

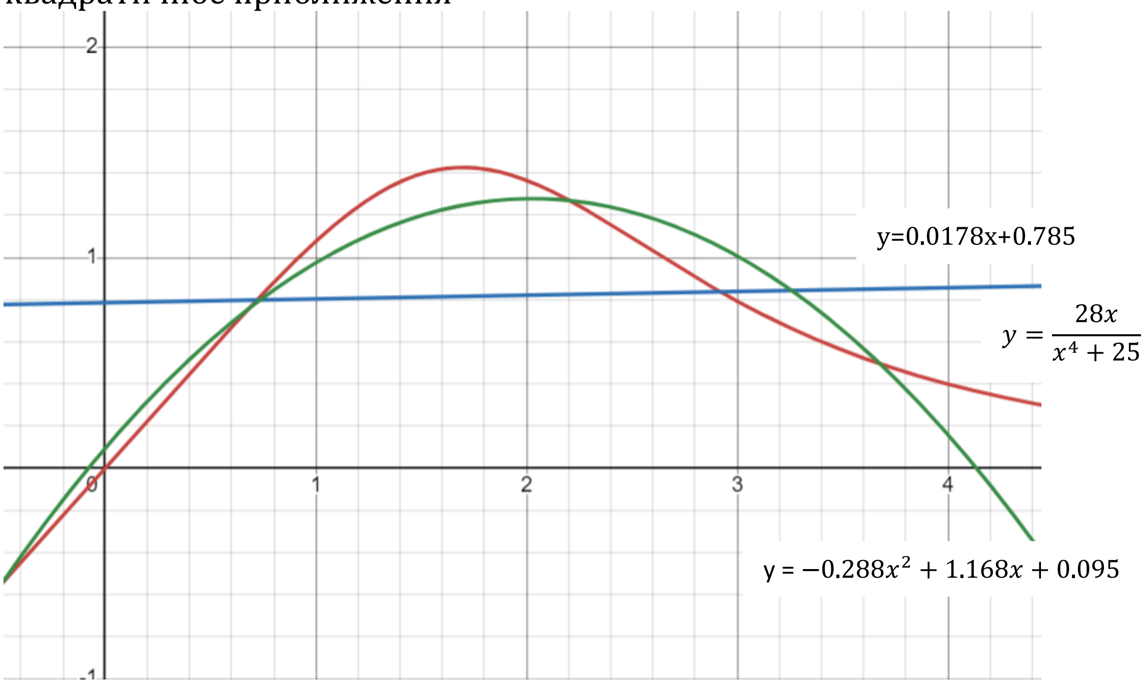
Проверим правильность выбора линейной модели. Для этого вычислим значения аппроксимирующей финкции $P_2(x) = -0.288x^2 + 1.168x + 0.095$

№ п.п	1	2	3	4	5	6	7	8	9	10	11
x	0	0.4	0.8	1.2	1.6	2	2.4	2.8	3.2	3.6	4.0
y	0	0.448	0.882	1.241	1.42	1.366	1.155	0.907	0.69	0.522	0.399
$P_2(x) = a_0 + a_1x + a_2x^2$	0.095	0.51612	0.84508	1.08188	1.22652	1.279	1.23932	1.10748	0.88348	0.56732	0.159
ε_i	-0.095	-0.06812	0.03692	0.15912	0.19348	0.087	-0.08432	-0.20048	-0.19348	-0.04532	0.24

Определим меру отклонения $S = \sum_{i=1}^{n} {\varepsilon_i}^2 = 0.229$

Среднеквадратичное отклонение $\delta = \sqrt{\frac{\sum_{i=1}^{n}(\varphi(x_i) - y_i)^2}{n}} = 0.1445$

- Выбрать наилучшее приближение
  Наилучшее приближение: Квадратичное приближения
- Построить графики заданной функции, а также полученные линейное и квадратичное приближения



5. Листинг программы

Main.py

```python
from approximation import *


def main():
    approximation()


if __name__ == "__main__":
    main()
```

approximation.py

```python
from valida_input import inpit_pairs_selection,
try_to_convert_to_int
from output import output_data
from sympy import symbols, Eq, solve
import math


def approximation():

    pairs = inpit_pairs_selection()
    linear_answer = linear_approximation(pairs)
    sqr_answer = sqr_approximation(pairs)
    cubic_answer = cubic_approximation(pairs)
    log_answer = log_approximation(pairs)
    exp_answer = exp_approximation(pairs)
    gradual_answer = gradual_approximation(pairs)

    answers = []
    try:
        answers.append(linear_answer)
    except TypeError:
        pass
    try:
        answers.append(sqr_answer)
    except TypeError:
        pass
    try:
        answers.append(cubic_answer)
    except TypeError:
        pass
    try:
        answers.append(log_answer)
    except TypeError:
        pass
    try:
        answers.append(exp_answer)
    except TypeError:
        pass
    try:
        answers.append(gradual_answer)
    except TypeError:
```

```
        pass
    output_data(
        pairs,
        answers,
        linear_answer,
        sqr_answer,
        cubic_answer,
        log_answer,
        exp_answer,
        gradual_answer,
    )


def linear_approximation(pairs):
    try:
        s_x = 0
        s_y = 0
        s_xx = 0
        s_xy = 0
        for pair in pairs:
            x = pair[0]
            y = pair[1]
            s_x += x
            s_y += y
            s_xx += x * x
            s_xy += x * y
        x_svg = s_x / len(pairs)
        y_svg = s_y / len(pairs)
        b = (s_y - s_x * s_xy / s_xx) / (len(pairs) - s_x * s_x
/ s_xx)
        a = (s_xy - s_x * b) / s_xx

        compare = []
        fi = []
        pi = []
        compare_sqr = 0
        for pair in pairs:
            temp = a * pair[0] + b
            fi.append(temp)
            e = temp - pair[1]
            temp2 = e / pair[1]
            pi.append(temp2)
            compare.append(e)
            compare_sqr += e * e
        tmp_top = 0
        tmp_bottom_left = 0
        tmp_bottom_right = 0
        for pair in pairs:
            tmp_top += (pair[0] - x_svg) * (pair[1] - y_svg)
            tmp_bottom_left += (pair[0] - x_svg) ** 2
            tmp_bottom_right += (pair[1] - y_svg) ** 2
        pirson = tmp_top / (tmp_bottom_left * tmp_bottom_right)
** 0.5
```

```
        S2 = (compare_sqr / len(pairs)) ** 0.5
        # print(f"P1(x) = {a}x + {b}\nS = {compare_sqr} Pirson =
{pirson}")
        return [S2, compare_sqr, f"P1(x) = {a}x + {b}", fi,
compare, pirson, pi, a, b]
    except ZeroDivisionError:
        print("There are nums < 0, cannot solve  linear
approximation")


def sqr_approximation(pairs):
    try:
        s_x = 0
        s_y = 0
        s_xx = 0
        s_xy = 0
        s_xxx = 0
        s_xxxx = 0
        s_xxy = 0
        for pair in pairs:
            x = pair[0]
            y = pair[1]
            s_x += x
            s_y += y
            s_xx += x * x
            s_xy += x * y
            s_xxx += x * x * x
            s_xxxx += x * x * x * x
            s_xxy += x * x * y

        x, y, z = symbols("x y z")
        eq1 = Eq(len(pairs) * x + s_x * y + s_xx * z, s_y)
        eq2 = Eq(s_x * x + s_xx * y + s_xxx * z, s_xy)
        eq3 = Eq(s_xx * x + s_xxx * y + s_xxxx * z, s_xxy)
        solution = solve((eq1, eq2, eq3), (x, y, z))

        a2 = try_to_convert_to_int(solution[x])
        a1 = try_to_convert_to_int(solution[y])
        a0 = try_to_convert_to_int(solution[z])
        fi = []
        pi = []
        compare = []
        compare_sqr = 0
        for pair in pairs:
            temp = a2 + a1 * pair[0] + a0 * pair[0] * pair[0]
            fi.append(temp)
            e = temp - pair[1]
            temp2 = e / pair[1]
            pi.append(temp2)
            compare.append(e)
            compare_sqr += e * e
        S2 = (compare_sqr / len(pairs)) ** 0.5
        return [
```

```
            S2,
            compare_sqr,
            f"P1(x) = {a0}x^2 + {a1}x + {a2}",
            fi,
            compare,
            pi,
            a0,
            a1,
            a2,
        ]
    except ZeroDivisionError:
        print("There are nums < 0, cannot solve  linear
approximation")


def cubic_approximation(pairs):
    s_x = 0
    s_y = 0
    s_xx = 0
    s_yy = 0
    s_xy = 0
    s_xxx = 0
    s_xxy = 0
    s_xyy = 0
    s_xxxx = 0
    s_xxyy = 0

    for pair in pairs:
        x = pair[0]
        y = pair[1]
        s_x += x
        s_y += y
        s_xx += x * x
        s_yy += y * y
        s_xy += x * y
        s_xxx += x * x * x
        s_xxy += x * x * y
        s_xyy += x * y * y
        s_xxxx += x * x * x * x
        s_xxyy += x * x * y * y

    x, y, z, w = symbols("x y z w")
    eq1 = Eq(len(pairs) * w + s_x * x + s_xx * y + s_xxx * z,
s_y)
    eq2 = Eq(s_x * w + s_xx * x + s_xxx * y + s_xxxx * z, s_xy)
    eq3 = Eq(s_xx * w + s_xxx * x + s_xxxx * y + s_xxyy * z,
s_xxy)
    eq4 = Eq(s_xxx * w + s_xxxx * x + s_xxyy * y + s_xyy * z,
s_xxx)

    solution = solve((eq1, eq2, eq3, eq4), (w, x, y, z))

    # Extract coefficients
```

```
    a3 = try_to_convert_to_int(solution[w])
    a2 = try_to_convert_to_int(solution[x])
    a1 = try_to_convert_to_int(solution[y])
    a0 = try_to_convert_to_int(solution[z])

    fi = []
    compare = []
    compare_sqr = 0
    for pair in pairs:
        temp = a3 + a2 * pair[0] + a1 * pair[0] ** 2 + a0 *
pair[0] ** 3
        fi.append(temp)
        e = temp - pair[1]
        compare.append(e)
        compare_sqr += e * e

    S2 = (compare_sqr / len(pairs)) ** 0.5

    return [
        S2,
        compare_sqr,
        f"P1(x) = {a0}x^3 + {a1}x^2 + {a2}x + {a3}",
        fi,
        compare,
        a0,
        a1,
        a2,
        a3,
    ]


def log_approximation(pairs):
    try:
        s_x = 0
        s_y = 0
        s_xx = 0
        s_xy = 0
        for pair in pairs:
            x = math.log(pair[0])
            y = pair[1]
            s_x += x
            s_y += y
            s_xx += x * x
            s_xy += x * y
        x_svg = s_x / len(pairs)
        y_svg = s_y / len(pairs)
        b = (s_y - s_x * s_xy / s_xx) / (len(pairs) - s_x * s_x
/ s_xx)
        a = (s_xy - s_x * b) / s_xx

        compare = []
        fi = []
        pi = []
```

```
        compare_sqr = 0
        for pair in pairs:
            temp = a * math.log(pair[0]) + b
            fi.append(temp)
            e = temp - pair[1]
            temp2 = e / pair[1]
            pi.append(temp2)
            compare.append(e)
            compare_sqr += e * e
        tmp_top = 0
        tmp_bottom_left = 0
        tmp_bottom_right = 0
        for pair in pairs:
            tmp_top += (math.log(pair[0]) - x_svg) * (pair[1] -
y_svg)
            tmp_bottom_left += (math.log(pair[0]) - x_svg) ** 2
            tmp_bottom_right += (pair[1] - y_svg) ** 2
        pirson = tmp_top / (tmp_bottom_left * tmp_bottom_right)
** 0.5
        S2 = (compare_sqr / len(pairs)) ** 0.5

        return [
            S2,
            compare_sqr,
            f"P1(x) = {a}ln(x) + {b}",
            fi,
            compare,
            pirson,
            pi,
            a,
            b,
        ]
    except ValueError:
        print("There are nums < 0, cannot solve logarithmic
approximation")


def exp_approximation(pairs):
    try:
        s_x = 0
        s_y = 0
        s_xx = 0
        s_xy = 0
        for pair in pairs:
            x = pair[0]
            y = math.log(pair[1])
            s_x += x
            s_y += y
            s_xx += x * x
            s_xy += x * y
        x_svg = s_x / len(pairs)
        y_svg = s_y / len(pairs)
        b = (s_y - s_x * s_xy / s_xx) / (len(pairs) - s_x * s_x
```

```
/ s_xx)
        a = (s_xy - s_x * b) / s_xx

        compare = []
        fi = []
        pi = []
        compare_sqr = 0
        for pair in pairs:
            temp = a * pair[0] + b
            fi.append(temp)
            e = temp - math.log(pair[1])
            temp2 = e / math.log(pair[1])
            pi.append(temp2)
            compare.append(e)
            compare_sqr += e * e
        tmp_top = 0
        tmp_bottom_left = 0
        tmp_bottom_right = 0
        for pair in pairs:
            tmp_top += (pair[0] - x_svg) * (math.log(pair[1]) -
y_svg)
            tmp_bottom_left += (pair[0] - x_svg) ** 2
            tmp_bottom_right += (math.log(pair[1]) - y_svg) ** 2
        pirson = tmp_top / (tmp_bottom_left * tmp_bottom_right)
** 0.5
        S2 = (compare_sqr / len(pairs)) ** 0.5
        a = round(a, 5)
        return [
            S2,
            compare_sqr,
            f"P1(x) = {a}x + ln({b})",
            fi,
            compare,
            pirson,
            pi,
            a,
            b,
        ]
    except ValueError:
        print("There are nums < 0, cannot solve logarithmic
approximation")
    except ZeroDivisionError:
        print("There are 0 in solution, cannot solve exp
approximation")


def gradual_approximation(pairs):
    try:
        s_x = 0
        s_y = 0
        s_xx = 0
        s_xy = 0
        for pair in pairs:
```

```python
            x = math.log(pair[0])
            y = math.log(pair[1])
            s_x += x
            s_y += y
            s_xx += x * x
            s_xy += x * y
        x_svg = s_x / len(pairs)
        y_svg = s_y / len(pairs)
        b = (s_y - s_x * s_xy / s_xx) / (len(pairs) - s_x * s_x
/ s_xx)
        a = (s_xy - s_x * b) / s_xx

        compare = []
        fi = []
        pi = []
        compare_sqr = 0
        for pair in pairs:
            temp = a * math.log(pair[0]) + b
            fi.append(temp)
            e = temp - math.log(pair[1])
            temp2 = e / math.log(pair[1])
            pi.append(temp2)
            compare.append(e)
            compare_sqr += e * e
        tmp_top = 0
        tmp_bottom_left = 0
        tmp_bottom_right = 0
        for pair in pairs:
            tmp_top += (math.log(pair[0]) - x_svg) *
(math.log(pair[1]) - y_svg)
            tmp_bottom_left += (math.log(pair[0]) - x_svg) ** 2
            tmp_bottom_right += (math.log(pair[1]) - y_svg) ** 2
        pirson = tmp_top / (tmp_bottom_left * tmp_bottom_right)
** 0.5
        S2 = (compare_sqr / len(pairs)) ** 0.5
        a = round(a, 5)
        return [
            S2,
            compare_sqr,
            f"P1(x) = {a}ln(x) + {b}",
            fi,
            compare,
            pirson,
            pi,
            a,
            b,
        ]
    except ValueError:
        print("There are nums < 0, cannot solve logarithmic
approximation")
    except ZeroDivisionError:
        print("There are 0 in solution, cannot solve gradual
approximation")
```

output.py

```
import os
import matplotlib.pyplot as plt
import numpy as np


def output_data(
    pairs,
    answers,
    linear_answer,
    sqr_answer,
    cubic_answer,
    log_answer,
    exp_answer,
    gradual_answer,
):
    input_variant = choose_output_format()
    switch_command = {
        1: output_in_console,
        2: output_in_file,
    }
    switch_command.get(input_variant, exit)(
        answers,
        linear_answer,
        sqr_answer,
        cubic_answer,
        log_answer,
        exp_answer,
        gradual_answer,
    )
    draw_grapth(
        pairs,
        answers,
        linear_answer,
        sqr_answer,
        cubic_answer,
        log_answer,
        exp_answer,
        gradual_answer,
    )


def choose_output_format():
    while True:
        try:
            print("Choose the output format:")
            input_variant = int(input("1) In console\n2) In 
file\n"))
            if input_variant in range(1, 3):
                break
            print("Invalid command")
```

```
            continue
        except ValueError:
            print("Incorrect value entered")
            continue
    return input_variant


def output_in_console(
    answers,
    linear_answer,
    sqr_answer,
    cubic_answer,
    log_answer,
    exp_answer,
    gradual_answer,
):
    print(find_best_method(answers))

    try:
        print(print_linear(linear_answer))
    except TypeError:
        print()
    try:
        print(print_sqr(sqr_answer))
    except TypeError:
        print()
    try:
        print(print_qubic(cubic_answer))
    except TypeError:
        print()
    try:
        print(print_log(log_answer))
    except TypeError:
        print()
    try:
        print(print_exp(exp_answer))
    except TypeError:
        print()
    try:
        print(print_gradual(gradual_answer))
    except TypeError:
        print()


def output_in_file(
    answers,
    linear_answer,
    sqr_answer,
    cubic_answer,
    log_answer,
    exp_answer,
    gradual_answer,
):
```

```
    try:
        filename = input("Enter the file name to save the data: ")

        file_path = os.path.join("./lb4/solutions", filename)

        if not os.path.exists("./lb4/solutions"):
            os.makedirs("./lb4/solutions")
        otput_best = find_best_method(answers)
        with open(file_path, "w") as file:
            file.write(otput_best)

            try:
                file.write(print_linear(linear_answer))
            except TypeError:
                file.write("\n")
            try:
                file.write(print_sqr(sqr_answer))
            except TypeError:
                file.write("\n")
            try:
                file.write(print_qubic(cubic_answer))
            except TypeError:
                file.write("\n")
            try:
                file.write(print_log(log_answer))
            except TypeError:
                file.write("\n")
            try:
                file.write(print_exp(exp_answer))
            except TypeError:
                file.write("\n")
            try:
                file.write(print_gradual(gradual_answer))
            except TypeError:
                file.write("\n")

        print(
            f"The values of variables have been successfully written to the file: {file_path}"
        )
    except FileNotFoundError:
        print("The specified path does not exist.")
    except PermissionError:
        print("You do not have permission to create a file in this directory.")


def print_linear(answers):
    fi = "fi ="
    ei = "ei ="
    for num in answers[3]:
        fi += f" {num},"
```

```
    for num in answers[4]:
        ei += f" {num},"
    pi = "Pi ="
    for num in answers[6]:
        pi += f" {num},"
    pirson = answers[5]
    return f"Linear\n{answers[2]}\n{fi}\nPirson = {pirson}\nS = 
{answers[1]}\nS2 = {answers[0]}\n{ei}\n{pi}\n\n"


def print_sqr(answers):
    fi = "fi ="
    ei = "ei ="
    for num in answers[3]:
        fi += f" {num},"
    for num in answers[4]:
        ei += f" {num},"
    pi = "Pi ="
    for num in answers[5]:
        pi += f" {num},"
    return f"Quadratic\n{answers[2]}\n{fi}\nS = {answers[1]}\nS2 
= {answers[0]}\n{ei}\n{pi}\n\n"


def print_qubic(answers):
    fi = "fi ="
    ei = "ei ="
    for num in answers[3]:
        fi += f" {num},"
    for num in answers[4]:
        ei += f" {num},"
    return f"Qubic\n{answers[2]}\n{fi}\nS = {answers[1]}\nS2 = 
{answers[0]}\n{ei}\n\n"


def print_log(answers):
    fi = "fi ="
    ei = "ei ="
    for num in answers[3]:
        fi += f" {num},"
    for num in answers[4]:
        ei += f" {num},"
    pi = "Pi ="
    for num in answers[6]:
        pi += f" {num},"
    pirson = answers[5]
    return f"Logarithmic\n{answers[2]}\n{fi}\nPirson = 
{pirson}\nS = {answers[1]}\nS2 = {answers[0]}\n{ei}\n{pi}\n\n"


def print_exp(answers):
    fi = "fi ="
    ei = "ei ="
```

```
    for num in answers[3]:
        fi += f" {num},"
    for num in answers[4]:
        ei += f" {num},"
    pi = "Pi ="
    for num in answers[6]:
        pi += f" {num},"
    pirson = answers[5]
    return f"Exp\n{answers[2]}\n{fi}\nPirson = {pirson}\nS = 
{answers[1]}\nS2 = {answers[0]}\n{ei}\n{pi}\n\n"


def print_gradual(answers):
    fi = "fi ="
    ei = "ei ="
    for num in answers[3]:
        fi += f" {num},"
    for num in answers[4]:
        ei += f" {num},"
    pi = "Pi ="
    for num in answers[6]:
        pi += f" {num},"
    pirson = answers[5]
    return f"Gradual\n{answers[2]}\n{fi}\nPirson = {pirson}\nS = 
{answers[1]}\nS2 = {answers[0]}\n{ei}\n{pi}\n\n"


def find_best_method(answers):
    min = 99999

    for answer in answers:
        try:
            if answer[0] < min:
                min = answer[0]
            # print(answer[2], end="zz")
        except TypeError:
            continue
    for answer in answers:
        try:
            if answer[0] == min:
                return f"Best quation is {answer[2]} with S2 = 
{answer[0]}\n"
        except TypeError:
            continue


def draw_grapth(
    pairs,
    answers,
    linear_answer,
    sqr_answer,
    cubic_answer,
    log_answer,
```

```
    exp_answer,
    gradual_answer,
):
    min = 99999
    max = -99999
    x_values_dots = [pair[0] for pair in pairs]
    y_values_dots = [pair[1] for pair in pairs]
    for pair in pairs:
        if pair[0] <= min:
            min = pair[0]
        if pair[0] >= max:
            max = pair[0]
    x = np.linspace(min - min * 0.3, max + max * 0.3, 400)
    try:
        y_values_linear = linear_answer[7] * x +
linear_answer[8]
        plt.plot(x, y_values_linear, label=linear_answer[2],
color="#461d9f")
    except TypeError:
        pass

    try:
        y_values_sqr = sqr_answer[6] * x**2 + sqr_answer[7] * x
+ sqr_answer[8]
        plt.plot(x, y_values_sqr, label=sqr_answer[2],
color="#FF5733")
    except TypeError:
        pass

    try:
        y_values_cubic = (
            cubic_answer[5] * x**3
            + cubic_answer[6] * x**2
            + cubic_answer[7] * x
            + cubic_answer[8]
        )
        plt.plot(x, y_values_cubic, label=cubic_answer[2],
color="#6C5CE7")
    except TypeError:
        pass

    try:
        y_values_log = log_answer[7] * np.log(x) + log_answer[8]
        plt.plot(x, y_values_log, label=log_answer[2],
color="#F08A5B")
    except TypeError:
        pass

    try:
        y_values_exp = exp_answer[7] * x + np.log(exp_answer[8])
        plt.plot(x, y_values_exp, label=exp_answer[2],
color="#5F27CD")
    except TypeError:
```

```
        pass

    try:
        y_values_grad = gradual_answer[7] * np.log(x) +
np.log(gradual_answer[8])
        plt.plot(x, y_values_grad, label=gradual_answer[2],
color="000")
    except TypeError:
        pass

    plt.scatter(x_values_dots, y_values_dots, label="Точки",
color="red", linewidths=2)
    # Добавим заголовок и метки осей
    plt.title("График функций")
    plt.xlabel("x")
    plt.ylabel("f(x)")

    # Добавим сетку
    plt.grid(True)

    # Отобразим линии вспомогательных осей
    plt.axhline(0, color="black", linewidth=0.5)
    plt.axvline(0, color="black", linewidth=0.5)

    # Добавим легенду
    plt.legend()

    # Отобразим график
    plt.show()
```

valida_input.py

```
import os


def inpit_pairs_selection():
    while True:
        try:
            print("Choose the quation:")
            input_selection = int(input("1) Hand input \n2) File
input\n"))
            if input_selection in range(1, 3):
                break
            print("Invalid command")
            continue
        except ValueError:
            print("Incorrect value entered")
            continue
    switch_command = {
        1: pairs_input_hand,
        2: pairs_input_file,
    }
    pairs = switch_command.get(input_selection, exit)()
```

```
    return pairs


def pairs_input_hand():
    pairs = []
    i = 1
    while i <= 12:
        try:
            temp = (
                input(f"Enter pair {i} x and y. For stop enter
end:\n")
                .strip()
                .replace(",", ".")
                .split(" ")
            )
            if temp == ["end"] and i >= 9:
                i += 1
                break
            elif temp == ["end"] and i < 9:
                print("Minimum 8 pairs, please add more")
                i += 1
                continue
            x, y = map(float, temp)
            i += 1

        except ValueError:
            print("Incorrect value entered")
            continue
        pair = [x, y]
        pairs.append(pair)
    print(f"Entered {i-1} pairs")

    return pairs


def pairs_input_file():
    current_working_directory = os.path.dirname(__file__)
    file_name = input("Enter the relative path to your file\n")
    print()
    file_path = os.path.join(current_working_directory,
file_name)
    with open(file_path, "r", encoding="utf-8") as file:
        # Читаем строки из файла
        lines = file.readlines()

    if len(lines) >= 8 and len(lines) <= 12:
        pairs = []
        for line in lines:
            try:
                var = line.replace(",", ".").split(" ")
                x, y = map(float, var)
                x = try_to_convert_to_int(x)
                y = try_to_convert_to_int(y)
```

```
                pair = [x, y]
                pairs.append(pair)
            except ValueError:
                print("Incorrect data in the specified file")
                exit()

            except UnboundLocalError:
                print("Incorrect data in the specified file")
                exit()
            continue

    else:
        print("Uncorrect count of lines in the specified file")
        exit()
    print(f"Readed {len(pairs)} pairs")
    for pair in pairs:
        print(f"pair[{pair[0]}, {pair[1]}]")
    print("\n\n")
    return pairs


def try_to_convert_to_int(number):
    try:
        number_float = float(number)
        if number_float.is_integer():
            return int(number_float)
        else:
            return number_float
    except ValueError:
        return float(number)
    )
```

6. Результаты выполнения программы

7. Выводы

В результате выполнения данной лабораторной работой мы познакомились с аппроксимацией функции методом наименьших квадратов и реализовали их на языке программирования Python, закрепив знания.

Аппроксимация может потребоваться, например, в случае, если из эксперимента известны лишь некоторые значения функции и требуется найти неизвестное. Или же, если изначальная функция слишком сложна для регулярного использования.

Можно выделить следующие достоинства метода: расчеты довольны просты необходимо лишь найти коэффициенты, полученная функция также проста, разнообразие возможных аппроксимирующих функций.

Основным недостатком МНК является чувствительность оценок к резким выбросам, которые встречаются в исходных данных.