НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
Факультет “Компьютерных технологий в дизайне”

Лабораторная работа № 4
по дисциплине “Вычислительная математика”
вариант №21

Выполнили:
Савельева Елизавета Юрьевна
Фурзикова Александра Евгеньевна
Группа: P3266

Преподаватель:
Машина Екатерина Алексеевна

г. Санкт-Петербург
2025

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

Вычислительная часть:
Вариант 21: $f(x) = \frac{14x}{x^4+21}$; $x \in [-4,\ 0];\ h = 0{,}4$

Таблица табулирования заданной функции на указанном интервале
n =11- количество узлов
a = -4 – начало интервала
b = 0 – конец интервала
xi =a+(i−1)h ,i=1,…,n - узлы

Получаем последовательность: −4.0, −3.6, −3.2, −2.8, −2.4, −2.0, −1.6, −1.2, −0.8, −0.4, 0.0

Для каждого xi подставляем аргумент и находим yi = = f(xi) = $\frac{14x}{xi^4+21}$

i	xi	yi = f(xi)
1	-4	-0.202
2	-3.6	-0.267
3	-3.2	-0.356
4	-2.8	-0.475
5	-2.4	-0.62
6	-2	-0.757
7	-1.6	-0.813
8	-1.2	-0.728
9	-0.8	-0.523
10	-0.4	-0.266
11	0	0

Линейная аппроксимация:

Мы предполагаем, что значение функции в каждой точке можно приблизить выражением φ1(x)=ax+b, где два неизвестных параметра a и b нужно подобрать так, чтобы суммарное отклонение от табличных значений yi стало минимальным

Функция ошибки: S(a,b)=∑ (axi+b−yi)^2 ⟶ min

Минимум квадратичной формы достигается, если обнулить частные производные ∂S/∂a=0, ∂S/∂b=0

$$\begin{cases} \partial S/\partial a = 0 \\ \partial S/\partial b = 0 \end{cases} \Leftrightarrow \begin{cases} 2\sum_{i=1}^{n}(axi + b - yi)xi = 0 \\ 2\sum_{i=1}^{n}(axi + b - yi) = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} a\sum_{i=1}^{n}x_i^2 + b\sum_{i=1}^{n}xi = \sum_{i=1}^{n}xiyi \\ a\sum_{i=1}^{n}xi + bn = \sum_{i=1}^{n}yi \end{cases}$$

Подставим значения сумм в систему: ∑xi = -22, ∑xi^2 = 61,6, ∑yi = -5.007, ∑xi yi =9.9396, n=11

$$\begin{cases} 61{,}6a - 22b = 9.9396 \\ -22a + 11b = -5.007 \end{cases}$$

Для решения системы используем правило Крамера:

Определитель: Δ=61.6·11-22^2=193.6

Частные определители: (в частных определителях столбец коэффициентов при соответствующей неизвестной заменяется столбцом свободных членов системы)

Δa=5.4032·11-(-22)·(−5.007)=-0.818

Δb=61.6−5.007+22·9.9396=-89.76

Параметры:

a = Δa/Δ = -0.004

b = Δb/Δ = -0.464

Функция линейной аппроксимации: φ1(x)=ax+b = -0.004x -0.464

Построение приближённых значений – поставляем параметры для каждого узла y1i=axi+b

i	xi	yi = f(xi)	y1i = axi+b	εi = y1i-yi
1	-4	-0.202	-0.447	-0.245
2	-3.6	-0.267	-0.448	-0.181
3	-3.2	-0.356	-0.45	-0.094
4	-2.8	-0.475	-0.452	0.023

5	-2.4	-0.62	-0.453	0.166
6	-2	-0.757	-0.455	0.302
7	-1.6	-0.813	-0.457	0.356
8	-1.2	-0.728	-0.459	0.269
9	-0.8	-0.523	-0.46	0.063
10	-0.4	-0.266	-0.462	-0.196
11	0	0	-0.464	-0.464

Сумма квадратов: $S1=\sum \varepsilon i^2=0.678$

Среднеквадратическое отклонение: $\sigma 1=\sqrt{\frac{S1}{n}}=0.248$

Квадратичная аппроксимация:

Мы предполагаем, что значение функции в каждой точке можно приблизить выражением $\varphi 2(x)=ax^2+bx+c$, где три неизвестных параметра a, b и c нужно подобрать так, чтобы суммарное отклонение от табличных значений yi стало минимальным

Функция ошибки: $S(a,b,c)=\sum (axi^2 +bxi + c-yi)^2 \longrightarrow \min$

Минимум квадратичной формы достигается, если обнулить частные производные $\partial S/\partial a=0$, $\partial S/\partial b=0$, $\partial S/\partial c=0$

$$\begin{cases} \partial S/\partial a = 0 \\ \partial S/\partial b = 0 \\ \partial S/\partial c = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} 2\sum_{i=1}^{n}(axi\text{^}2 + bxi + c - yi)xi\text{^}2 = 0 \\ 2\sum_{i=1}^{n}(axi\text{^}2 + bxi + c - yi)xi = 0 \\ 2\sum_{i=1}^{n}(axi\text{^}2 + bxi + c - yi) = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} a\sum_{i=1}^{n}xi\text{^}4 + b\sum_{i=1}^{n}xi\text{^}3 + c\sum_{i=1}^{n}xi\text{^}2 = \sum_{i=1}^{n}xi\text{^}2 * yi \\ a\sum_{i=1}^{n}xi\text{^}3 + b\sum_{i=1}^{n}xi\text{^}2 + c\sum_{i=1}^{n}xi = \sum_{i=1}^{n}xi * yi \\ a\sum_{i=1}^{n}xi\text{^}2 + b\sum_{i=1}^{n}xi + nc = \sum_{i=1}^{n}yi \end{cases}$$

Подставим значения сумм в систему: ∑xi = -22, ∑xi^2 = 61,6, ∑xi^3 = -193.6, ∑xi^4 = 648.525 ∑yi = -5.007, ∑xi yi = 9.939, ∑xi^2 yi = -24.168, n=11

$$\begin{cases} 648.525a - 193.6b + 61{,}6c = -24.168 \\ -193.6a + 61{,}6b - 22c = 9.939 \\ 61{,}6a - 22b + 11c = -5.007 \end{cases}$$

Для решения системы используем правило Крамера:

a = Δa/Δ = -0.073

b = Δb/Δ = 0.647

c = Δc/Δ = 0.163

Функция квадратичной аппроксимации: φ2(x)= ax^2+bx+c= 0.163x^2+0.647x-0.073

Построение приближённых значений – поставляем параметры для каждого узла y2i=axi^2+bxi+c

i	xi	yi = f(xi)	y2i=axi^2+bxi+c	εi = y2i-yi
1	-4	-0.202	-0.056	0.146
2	-3.6	-0.267	-0.292	-0.025

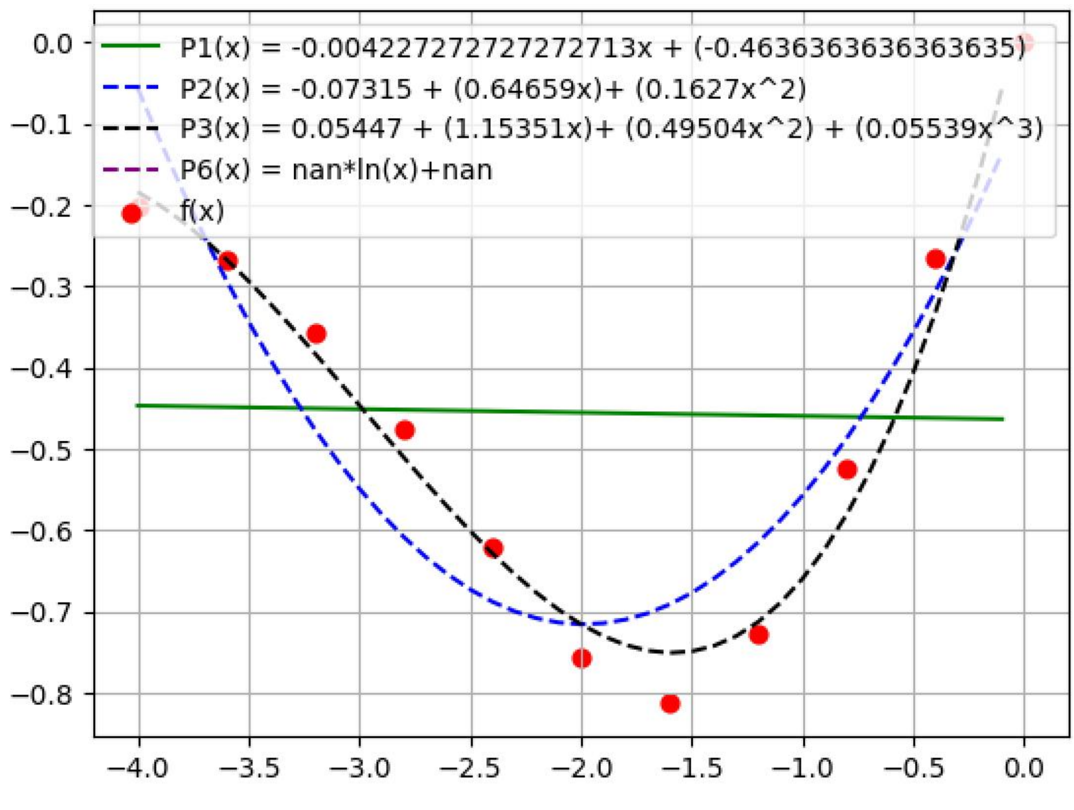
3	-3.2	-0.356	-0.476	-0.120
4	-2.8	-0.475	-0.608	-0.133
5	-2.4	-0.62	-0.688	-0.068
6	-2	-0.757	-0.715	0.041
7	-1.6	-0.813	-0.691	0.122
8	-1.2	-0.728	-0.615	0.113
9	-0.8	-0.523	-0.486	0.037
10	-0.4	-0.266	-0.306	-0.039
11	0	0	-0.073	-0.073

Сумма квадратов: S2=∑εi^2=0.096

Среднеквадратическое отклонение: $\sigma 2=\sqrt{\frac{S2}{n}}=0.093$

Выбор наилучшего приближения
Чем меньше среднеквадратичное отклонение, тем более точная модель построена.
σ2=0.093< σ1=0.248
Следовательно, наилучшим является квадратичное приблежение

Построение графика заданной функции, а также полученного линейного и квадратичного приближения



Листинг программы:

```
import warnings

from Terminal import Terminal

if __name__ == "__main__":
    warnings.filterwarnings('ignore')
    terminal = Terminal()
    while True:
        terminal.refresh()
        terminal.work()
        print("Хотите продолжить работу с программой? y/n")
        if not input().__eq__('y'):
            break
from numpy import exp, log

import numpy as np

from ApproximationFunctions import ApproximationFunctions
from Validation.Validator import Validator
from Exceptions.IncorrectValueException import IncorrectValueException
import matplotlib.pyplot as plt

INTERVAL_FOR_GRAPHIC_X = 1


class Terminal:
    def __init__(self):
        self.__isFromFile: bool = False

    def isFile(self, mode) -> bool:
        if mode == 0:
            print("Выберите формат ввода(с клавиатуры - k или из файла - f):")
        else:
            print("Хотите записать результаты в файл?(нет - k или  да - f):")
        is_file = input()
        if is_file.__eq__('k'):
            return False
        elif is_file.__eq__('f'):
            return True
        else:
            return self.isFile(mode)

    def refresh(self):
        self.__isFromFile = False

    def work(self):
        try:
```

```
            print('\t\t\tАппроксимация функции методом наименьших квадратов')
            self.__isFromFile = self.isFile(0)
            if self.__isFromFile:
                x_array, y_array = self.enterArraysFromFile()
            else:
                x_array, y_array = self.enterArraysFromKeyboard()
            min_x, max_x = min(x_array), max(x_array)
            x = np.arange(min_x, max_x, 0.1)
            self.__isFromFile = self.isFile(1)
            results = list()
            approximation = ApproximationFunctions(x_array, y_array)
            results.append(approximation.linearFunction())
            results.append(approximation.polynomialSecondFunction())
            results.append(approximation.polynomialThirdFunction())
            results.append(approximation.exponentialFunction())
            results.append(approximation.exponentFunction())
            results.append(approximation.logarithmFunction())
            self.printTableOfCompare(results)
            index = self.minSD(results)
            print(f'Наилучшее приближение :\n'
                  f'1.Аппроксимирующая функция: {results[index][0]}\n'
                  f'2. Коэффициенты:{results[index][1]}\n'
                  f'3. Среднеквадратичное отклонение: {results[index][3]}')

            plt.scatter(x_array, y_array, color='red', s=40, marker='o',
label='f(x)')
            plt.plot(x, [results[0][1]['a'] * x + results[0][1]['b'] for x in
x], color='green', label=results[0][0])
            plt.plot(x,
                     [results[1][1]['a0'] + x * results[1][1]['a1'] +
results[1][1]['a2'] * x ** 2 for x in x],
                     '--',
                     color='blue', label=results[1][0])
            plt.plot(x,
                     [results[2][1]['a0'] + x * results[2][1]['a1'] +
results[2][1]['a2'] * x ** 2 + results[2][1][
                         'a3'] * x ** 3 for x in x], '--',
                     color='black', label=results[2][0])
            plt.plot(x,
                     [results[5][1]['a'] * log(x) + results[5][1]['b'] for x
in x], '--',
                     color='purple', label=results[5][0])
            plt.legend()
            plt.grid()
            plt.figure(2)
            plt.plot(x, [results[3][1]['a'] * x ** results[3][1]['b'] for x
in x], '--',
                     color='pink', label=results[3][0])
            plt.scatter(x_array, y_array, color='red', s=40, marker='o',
label='f(x)')
            plt.legend()
            plt.grid()
            plt.figure(3)
            plt.plot(x,
                     [results[4][1]['a'] * exp(x * results[4][1]['b']) for x
in x], '--',
                     color='orange', label=results[4][0])
            plt.scatter(x_array, y_array, color='red', s=40, marker='o',
label='f(x)')
            plt.legend()
            plt.grid()
        except IncorrectValueException as e:
            print(e.message)
            return
```

```
        except ValueError as e:
            print(e.message)
            return
        except RuntimeWarning:
            pass
        finally:
            plt.show()

    def enterArray(self, variable) -> object:
        try:
            print(f'Введите значения {variable} (через ;):')
            arr = [Validator.validateNumber(x) for x in input().split(';')]
            Validator.validateArraySize(arr)
            return arr
        except IncorrectValueException as e:
            print(e.message)
            return self.enterArray(variable)

    def enterArraysFromKeyboard(self):
        x_array = self.enterArray('x')
        y_array = self.enterArray('y')
        Validator.validateEqualityLengths(x_array, y_array)
        return x_array, y_array

    def enterArraysFromFile(self):
        try:
            f = open(input('Введите путь к файлу:'), 'r')
            x_array = [Validator.validateNumber(x) for x in
f.readline().split(';')]
            Validator.validateArraySize(x_array)
            y_array = [Validator.validateNumber(x) for x in
f.readline().split(';')]
            Validator.validateArraySize(y_array)
            Validator.validateEqualityLengths(x_array, y_array)
            return x_array, y_array
        except IncorrectValueException as e:
            print(e.message)
            return self.enterArraysFromFile()
        except FileNotFoundError as e:
            print(e.strerror)
            return self.enterArraysFromFile()

    def printTableOfCompare(self, results):
        print('\t\t\tРезультаты ')
        print('| Вид функции         |      Коэффициенты         '
              '|   Мера отклонения     |  Среднеквадратичное отклонение    |')
        for i in range(len(results)):
            print(f'| {results[i][0]} | ', end='')
            for key, val in results[i][1].items():
                print(f'{key} = {val}|', end='')
            print(f"| {results[i][2]} | {results[i][3]} |")

    def minSD(self, results):
        min = results[0][3]
        index = 0
        for i in range(len(results)):
            if results[i][3] < min:
                index = i
        return index
import math

from numpy import exp, log

from Approximation import Approximation
```

```python
import numpy as np


class ApproximationFunctions(Approximation):

    def __calculateLinearFunctionCoefficient(self):
        SX = self.calculateSumX(1)
        SXX = self.calculateSumX(2)
        SY = self.calculateSumY()
        SXY = self.calculateSumXY(1)
        N = self.getN()
        delta = SXX * N - SX * SX
        delta1 = SXY * N - SX * SY
        delta2 = SXX * SY - SX * SXY
        a = delta1 / delta
        b = delta2 / delta
        self._setPX(self._calculateLinearValues(a, b))
        self._setEpsilon(self._calculateEpsilon())
        S = self._deviationMeasure()
        SD = self._standardDeviation()
        return {'SX': SX, 'SXX': SXX, 'SY': SY,
                'SXY': SXY, 'N': N, 'delta': delta,
                'delta1': delta1, 'delta2': delta2,
                'a': a, 'b': b, 'S': S, 'SD': SD}

    # Линейная функция
    def linearFunction(self):
        coefficients = self.__calculateLinearFunctionCoefficient()
        print('\t\t\tЛинейная аппроксимация')
        print(f"\tСистема уравнений:\n"
              f"\t\t{coefficients['SXX']}a + {coefficients['SX']}b = 
{coefficients['SXY']}\n"
              f"\t\t{coefficients['SX']}a + {coefficients['N']}b = 
{coefficients['SY']}")
        print(f"\tИз нее находим:\n"
              f"\t\tdelta = {coefficients['SXX']}*{coefficients['N']} - 
{coefficients['SX']}*{coefficients['SX']} = {coefficients['delta']}\n"
              f"\t\tdelta1 = {coefficients['SXY']}*{coefficients['N']} - 
{coefficients['SX']}*{coefficients['SY']} = {coefficients['delta1']}\n"
              f"\t\tdelta2 = {coefficients['SXX']}*{coefficients['SY']} - 
{coefficients['SX']}*{coefficients['SXY']} = {coefficients['delta2']}\n"
              f"\t\t\ta = {coefficients['delta1']}/{coefficients['delta']} = 
{coefficients['a']}\n"
              f"\t\t\tb = {coefficients['delta2']}/{coefficients['delta']} = 
{coefficients['b']}\n")
        function = f"P1(x) = {coefficients['a']}x + ({coefficients['b']})"
        self.__printCoefficients(coefficients, function)
        print(f"Коэффициент корреляции Пирсона: 
{self._coefficientCorrelation()}")
        return function, {'a': coefficients['a'], 'b': coefficients['b']}, 
coefficients['S'], coefficients['SD']

    def __calculatePolynomialSecondFunctionCoefficient(self):
        SX = self.calculateSumX(1)
        SXX = self.calculateSumX(2)
        SXXX = self.calculateSumX(3)
        SXXXX = self.calculateSumX(4)
        SY = self.calculateSumY()
        SXY = self.calculateSumXY(1)
        SXXY = self.calculateSumXY(2)
        N = self.getN()
        M1 = np.array([[N, SX, SXX], [SX, SXX, SXXX], [SXX, SXXX, SXXXX]])  # 
Матрица (левая часть системы)
        V1 = np.array([SY, SXY, SXXY])  # Вектор (правая часть системы)
```

```
        a0, a1, a2 = np.linalg.solve(M1, V1)
        self._setPX(self._calculatePolynomialSecondValues(a0, a1, a2))
        self._setEpsilon(self._calculateEpsilon())
        S = self._deviationMeasure()
        SD = self._standardDeviation()
        return {'SX': SX, 'SXX': SXX, 'SXXX': SXXX, 'SXXXX': SXXXX,
                'SY': SY, 'SXY': SXY, 'SXXY': SXXY, 'N': N,
                'a0': round(a0, 5), 'a1': round(a1, 5), 'a2': round(a2, 5),
'S': S, 'SD': SD}

    def polynomialSecondFunction(self):
        print('\t\t\tПолиномиальная функция 2-ой степени')
        coefficients = self.__calculatePolynomialSecondFunctionCoefficient()
        function = f"P2(x) = {coefficients['a0']} + ({coefficients['a1']}x)+
({coefficients['a2']}x^2)"
        print(f"\tПолучаем систему уравнений:\n"
              f"\t\t{coefficients['N']}a0 + {coefficients['SX']}a1 +
{coefficients['SXX']}a2 = {coefficients['SY']}\n"
              f"\t\t{coefficients['SX']}a0 + {coefficients['SXX']}a1 +
{coefficients['SXXX']}a2= {coefficients['SXY']}\n"
              f"\t\t{coefficients['SXX']}a0 + {coefficients['SXXX']}a1 +
{coefficients['SXXXX']}a2= {coefficients['SXXY']}\n")
        print(f"\tНайденные коэффициенты:\n"
              f"\t\ta0 = {coefficients['a0']}\n"
              f"\t\ta1 = {coefficients['a1']}\n"
              f"\t\ta2 = {coefficients['a2']}\n")
        self.__printCoefficients(coefficients, function)
        return function, {'a0': coefficients['a0'], 'a1': coefficients['a1'],
'a2': coefficients['a2']}, \
            coefficients['S'], coefficients['SD']

    def __calculatePolynomialThirdFunctionCoefficient(self):
        SX = self.calculateSumX(1)
        SXX = self.calculateSumX(2)
        SXXX = self.calculateSumX(3)
        SXXXX = self.calculateSumX(4)
        SXXXXX = self.calculateSumX(5)
        SXXXXXX = self.calculateSumX(6)
        SY = self.calculateSumY()
        SXY = self.calculateSumXY(1)
        SXXY = self.calculateSumXY(2)
        SXXXY = self.calculateSumXY(3)
        N = self.getN()
        M1 = np.array([[N, SX, SXX, SXXX], [SX, SXX, SXXX, SXXXX], [SXX, SXXX,
SXXXX, SXXXXX],
                       [SXXX, SXXXX, SXXXXX, SXXXXXX]])  # Матрица (левая
часть системы)
        V1 = np.array([SY, SXY, SXXY, SXXXY])  # Вектор (правая часть системы)
        a0, a1, a2, a3 = np.linalg.solve(M1, V1)
        self._setPX(self._calculatePolynomialThirdValues(a0, a1, a2, a3))
        self._setEpsilon(self._calculateEpsilon())
        S = self._deviationMeasure()
        SD = self._standardDeviation()
        return {'SX': SX, 'SXX': SXX, 'SXXX': SXXX, 'SXXXX': SXXXX, 'SXXXXX':
SXXXXX, 'SXXXXXX': SXXXXXX,
                'SY': SY, 'SXY': SXY, 'SXXY': SXXY, 'SXXXY': SXXXY, 'N': N,
                'a0': round(a0, 5), 'a1': round(a1, 5), 'a2': round(a2, 5),
'a3': round(a3, 5), 'S': S, 'SD': SD}

    def polynomialThirdFunction(self):
        print('\t\t\tПолиномиальная функция 3-ей степени')
        coefficients = self.__calculatePolynomialThirdFunctionCoefficient()
        function = f"P3(x) = {coefficients['a0']} + ({coefficients['a1']}x)+
({coefficients['a2']}x^2) + ({coefficients['a3']}x^3)"
```

```
        print(f"\tПолучим систему уравнений:\n"
              f"\t\t{coefficients['N']}a0 + {coefficients['SX']}a1 + 
{coefficients['SXX']}a2 + {coefficients['SXXX']}a3 = {coefficients['SY']}\n"
              f"\t\t{coefficients['SX']}a0 + {coefficients['SXX']}a1 + 
{coefficients['SXXX']}a2 + {coefficients['SXXXX']}a3= 
{coefficients['SXY']}\n"
              f"\t\t{coefficients['SXX']}a0 + {coefficients['SXXX']}a1 + 
{coefficients['SXXXX']}a2 + {coefficients['SXXXXX']}a3= 
{coefficients['SXXY']}\n"
              f"\t\t{coefficients['SXXX']}a0 + {coefficients['SXXXX']}a1 + 
{coefficients['SXXXXX']}a2 + {coefficients['SXXXXXX']}a3= 
{coefficients['SXXXY']}\n")
        print(f"\tНайденные коэффициенты:\n"
              f"\t\ta0 = {coefficients['a0']}\n"
              f"\t\ta1 = {coefficients['a1']}\n"
              f"\t\ta2 = {coefficients['a2']}\n"
              f"\t\ta3 = {coefficients['a3']}\n")
        self.__printCoefficients(coefficients, function)
        return function, {'a0': coefficients['a0'], 'a1': coefficients['a1'], 
'a2': coefficients['a2'],
                          'a3': coefficients['a3']}, \
            coefficients['S'], coefficients['SD']

    # Экспоненциальная
    def __calculateExponentFunctionCoefficient(self):
        y_arr = self.getArrayY()
        self.setArrayY([log(y) for y in y_arr])
        A, B, a, b = self.calculateLinearForOtherFunctions()
        self.setArrayY(y_arr)
        self._setPX(self._calculateExponentValues(a, b))
        self._setEpsilon(self._calculateEpsilon())
        S = self._deviationMeasure()
        SD = self._standardDeviation()
        return {'A': A, 'B': B, 'a': a,
                'b': b, 'S': S, 'SD': SD}

    def exponentFunction(self):
        print('\t\t\tЭкспоненциальная функция')
        coefficients = self.__calculateExponentFunctionCoefficient()
        function = f"P4(x) = {coefficients['a']}*e({coefficients['b']}*x)"
        self.__printCoefficients(coefficients, function)
        return function, {'a': coefficients['a'], 'b': coefficients['b']}, \
            coefficients['S'], coefficients['SD']

    # Степенная
    def __calculateExponentialFunctionCoefficient(self):
        y_arr, x_arr = self.getArrayY(), self.getArrayX()
        self.setArrayY([log(y) for y in y_arr])
        self.setArrayX([log(x) for x in x_arr])
        A, B, a, b = self.calculateLinearForOtherFunctions()
        self.setArrayY(y_arr)
        self.setArrayX(x_arr)
        self._setPX(self._calculateExponentialValues(a, b))
        self._setEpsilon(self._calculateEpsilon())
        S = self._deviationMeasure()
        SD = self._standardDeviation()
        return {'A': A, 'B': B, 'a': a,
                'b': b, 'S': S, 'SD': SD}

    def exponentialFunction(self):
        print('\t\t\tСтепенная функция')
        coefficients = self.__calculateExponentialFunctionCoefficient()
        function = f"P5(x) = {coefficients['a']}*x^({coefficients['b']})"
        self.__printCoefficients(coefficients, function)
```

```
        return function, {'a': coefficients['a'], 'b': coefficients['b']}, \
            coefficients['S'], coefficients['SD']

    # Логарифмическая
    def __calculateLogarithmFunctionCoefficient(self):
        x_arr = self.getArrayX()
        self.setArrayX([log(x) for x in x_arr])
        coefficient = self.__calculateLinearFunctionCoefficient()
        A, B = coefficient['a'], coefficient['b']
        a, b = A, B
        self.setArrayX(x_arr)
        self._setPX(self._calculateLogarithmValues(a, b))
        self._setEpsilon(self._calculateEpsilon())
        S = self._deviationMeasure()
        SD = self._standardDeviation()
        return {'A': A, 'B': B, 'a': a,
                'b': b, 'S': S, 'SD': SD}

    def logarithmFunction(self):
        print('\t\t\tЛогарифмическая функция')
        coefficients = self.__calculateLogarithmFunctionCoefficient()
        function = f"P6(x) = {coefficients['a']}*ln(x)+{coefficients['b']}"
        self.__printCoefficients(coefficients, function)
        return function, {'a': coefficients['a'], 'b': coefficients['b']}, \
            coefficients['S'], coefficients['SD']

    def __printCoefficients(self, coefficients, function):
        for key, val in coefficients.items():
            print(f" {key} = {val}.")
        print(f"Аппроксимирующая функция имеет вид {function}")
        self._printTable()
        print(f"Мера отклонения S = {coefficients['S']}")
        print(f"Среднеквадратическое отклонение SD = {coefficients['SD']}")

    def calculateLinearForOtherFunctions(self):
        coefficient = self.__calculateLinearFunctionCoefficient()
        A, B = coefficient['b'], coefficient['a']
        a, b = math.exp(A), B
        return A, B, a, b

from numpy import sqrt, log, exp


class Approximation:

    def __init__(self, x_arr, y_arr):
        self.__x_array = x_arr
        self.__y_array = y_arr
        self.__n = len(self.__x_array)
        self.__p_x = []
        self.__epsilon = []

    def getArrayX(self):
        return self.__x_array

    def setArrayX(self, arr):
        self.__x_array = arr

    def getArrayY(self):
        return self.__y_array

    def setArrayY(self, arr):
        self.__y_array = arr
```

```python
    def getN(self):
        return self.__n

    def getEpsilon(self):
        return self.__epsilon

    def _setEpsilon(self, eps):
        self.__epsilon = eps

    def getPX(self):
        return self.__p_x

    def _setPX(self, p_x):
        self.__p_x = p_x

    def calculateSumX(self, degree):
        return sum([x ** degree for x in self.__x_array])

    def calculateSumY(self):
        return sum(self.__y_array)

    def calculateSumXY(self, degree):
        return sum([self.__x_array[i] ** degree * self.__y_array[i] for i in
range(self.__n)])

    def _calculateEpsilon(self):
        return [self.__p_x[i] - self.__y_array[i] for i in range(self.__n)]

    def _calculateExponentialValues(self, a, b):
        return [a * x ** b for x in self.__x_array]

    def _calculateLogarithmValues(self, a, b):
        return [a * log(x) + b for x in self.__x_array]

    def _calculateExponentValues(self, a, b):
        return [a * exp(b * x) for x in self.__x_array]

    def _calculateLinearValues(self, a, b):
        return [a * x + b for x in self.__x_array]

    def _calculatePolynomialSecondValues(self, a0, a1, a2):
        return [a0 + a1 * x + a2 * x ** 2 for x in self.__x_array]

    def _calculatePolynomialThirdValues(self, a0, a1, a2, a3):
        return [a0 + a1 * x + a2 * x ** 2 + a3 * x ** 3 for x in
self.__x_array]

    def _deviationMeasure(self):
        return sum([eps ** 2 for eps in self.__epsilon])

    def _standardDeviation(self):
        return sqrt(self._deviationMeasure() / self.__n)

    def _printTable(self):
        print('|   X   |   Y  | P(x) | epsilon |')
        for i in range(self.__n):
            print(f"| {self.__x_array[i]} | {self.__y_array[i]} |
{self.__p_x[i]} | {self.__epsilon[i]} |")

    def _coefficientCorrelation(self):
        x_mean = self._calculateMeanX()
        y_mean = self._calculateMeanY()
        return sum([(self.__x_array[i] - x_mean) * (self.__y_array[i] -
y_mean)
```

```
                    for i in range(self.__n)]) / sqrt(
            sum([(self.__x_array[i] - x_mean) ** 2 for i in range(self.__n)])
*
            sum([(self.__y_array[i] - y_mean) ** 2 for i in range(self.__n)]))

    def _calculateMeanX(self):
        return sum(self.__x_array) / self.__n

    def _calculateMeanY(self):
        return sum(self.__y_array) / self.__n
from Exceptions.IncorrectValueException import IncorrectValueException

AMOUNT_OF_INPUT_ARRAYS = 2
MIN_DOTS = 8
MAX_DOTS = 12


class Validator:

    @staticmethod
    def validateNumber(number: str):
        try:
            number = float(number)
            return number
        except ValueError:
            raise IncorrectValueException(f'Необходимо ввести число.')


    @staticmethod
    def validateArraySize(arr):
        length = len(arr)
        if MIN_DOTS > length or length > MAX_DOTS:
            raise IncorrectValueException('Размер массива должен находится в
пределах [8;12].')

    @staticmethod
    def validateEqualityLengths(arr_1, arr_2):
        if len(arr_1) != len(arr_2):
            raise IncorrectValueException('Длины массивов не равны.')
class IncorrectValueException(Exception):

    def __init__(self, message):
        self.message = message
```

Вывод

В ходе лабораторной работы была исследована задача аппроксимации табличной функции одиннадцатью точками методом наименьших квадратов. Сначала вручную были построены линейная и квадратичная модели и рассчитаны их среднеквадратические ошибки: квадратичное приближение показало более низкий RMS, чем линейное, что уже свидетельствовало об отсутствии линейного закона.
Все результаты — сводная таблица S, RMS и R^2 для каждого семейства, подробный отчёт с отдельными значениями $\varphi(x_i)$ и ε_i, а также графики — автоматически сохраняются в отдельной папке result_N при каждом запуске программы. Таким образом, поставленная цель — найти оптимальную

функцию-аппроксимацию методом наименьших квадратов — успешно достигнута.