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 №25

*Студент:*

Хоанг Ван Куан

Группа Р3266

*Преподаватель:*

Машина Екатерина Александровна

Санкт-Петербург, 2024

1. **Цель работы**

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

1. **Порядок выполнения работы**

* 1 часть: Вычислительная реализация задачи
  1. Сформировать таблицу табулирования заданной функции на указанном интервале
  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 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

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

1. **Рабочие формулы**

Аппроксимировать f(x) функцией

**

Коэффициент корреляции

Среднеквадратичное отклонение

Выбор аппроксимирующей функции

1. **Вычислительная часть** 
   * Сформировать таблицу табулирования функции

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.0 | 0.4 | 0.8 | 1.2 | 1.6 | 2.0 | 2.4 | 2.8 | 3.2 | 3.6 | 4.0 |
|  | 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

Вычисляем суммы:

Получим систему управнений для нахождения параметров и

Решая систему, получим значения коэффициентов:

Проверим правильность выбора линейной модели. Для этого вычислим значения аппроксимирующей финкции

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № п.п | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | 0 | 0.4 | 0.8 | 1.2 | 1.6 | 2 | 2.4 | 2.8 | 3.2 | 3.6 | 4.0 |
|  | 0 | 0.448 | 0.882 | 1.241 | 1.42 | 1.366 | 1.155 | 0.907 | 0.69 | 0.522 | 0.399 |
| = | 0.785 | 0.79212 | 0.79924 | 0.80636 | 0.81348 | 0.8206 | 0.82772 | 0.83484 | 0.84196 | 0.84908 | 0.8562 |
|  | -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 2.047

Среднеквадратичное отклонение

+ Квадратичное приближения:

Для определения вида зависимости. Выбираем многочлен второй степени и строим линейную модель

Сумма квадратов отклонений запишется следующим образом:

Вычислим:

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

Проверим правильность выбора линейной модели. Для этого вычислим значения аппроксимирующей финкции

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

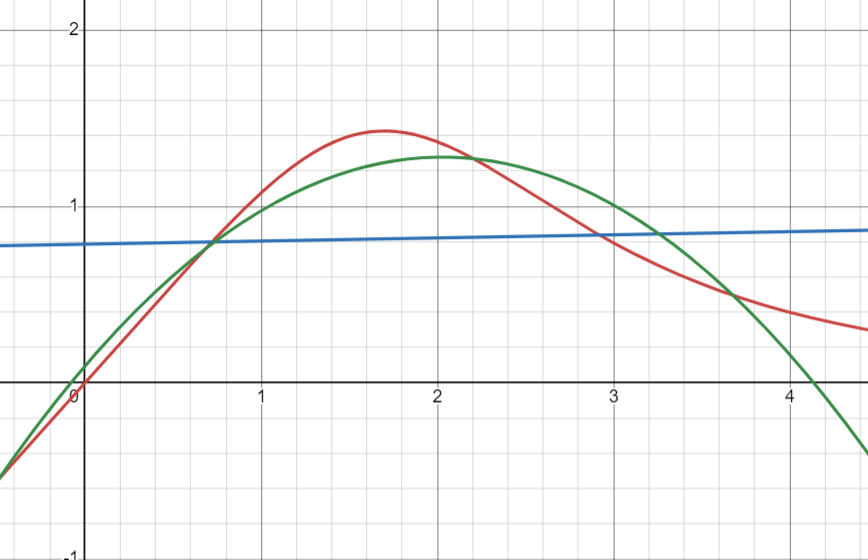
Определим меру отклонения 0.229

Среднеквадратичное отклонение

* + Выбрать наилучшее приближение

Наилучшее приближение: Квадратичное приближения

* + Построить графики заданной функции, а также полученные линейное и квадратичное приближения



y =

y=0.0178x+0.785

1. **Листинг программы**

from dataclasses import dataclass

import math

import numpy as np

@dataclass

class Result:

    coefficients: iter

    apply: callable

    function: str

    S: float

    deviation: float

    confidence: float

    r: float | None = None

    def \_\_lt\_\_(self, other):

        return self.deviation < other.deviation

    def \_\_le\_\_(self, other):

        return self.deviation == other.deviation

def getData(x, y):

    x = np.array(x)

    y = np.array(y)

    return x, y

# среднеквадратичное отклонение

def msr(phi, y):

    return (np.sum((phi - y) \*\* 2) / len(y)) \*\* 0.5

#  достоверность аппроксимации

def confidence(phi, y):

    return 1 - np.sum((y - phi) \*\* 2) / (np.sum(phi \*\* 2) - np.sum(phi) \*\* 2 / len(y))

# Коэффициент корреляции

def r(x, y):

    x0, y0 = np.mean(x), np.mean(y)

    return np.sum((x - x0) \* (y - y0)) / (np.sum((x - x0) \*\* 2) \* np.sum((y - y0) \*\* 2)) \*\* 0.5

# линейная функция - done

def Approximation\_linear(x, y):

    x, y = getData(x, y)

    SX, SXX, SY, SXY = np.sum(x), np.sum(x \*\* 2), np.sum(y), np.sum(x \* y)

    b, a = np.linalg.solve(

        np.array([[len(x), SX], [SX, SXX]]),

        np.array([SY, SXY])

    )

    # Сумма квадратов отклонений

    phi = a \* x + b

    S = np.sum((phi - y) \*\* 2)

    return Result(

        coefficients = (a, b),

        apply = lambda x: a \* x + b,

        function = f'{a:.4f}x + {b:.4f}',

        S = S,

        r = r(x, y),

        confidence = confidence(phi, y),

        deviation = msr(phi, y)

    )

# полиномиальная функция 2-й степени - done

def Approximation\_degree2(x, y):

    x, y = getData(x, y)

    SX, SXX, SXXX, SXXXX, SY, SXY, SXXY = np.sum(x), np.sum(x\*\*2), np.sum(x\*\*3), np.sum(x\*\*4), np.sum(y), np.sum(x\*y), np.sum(x\*x\*y)

    a0, a1, a2 = np.linalg.solve(

        np.array([[len(x), SX, SXX], [SX, SXX, SXXX], [SXX, SXXX, SXXXX]]),

        np.array([SY, SXY, SXXY])

    )

     # Сумма квадратов отклонений

    phi = a2\*x\*\*2 + a1\*x + a0

    S = np.sum((phi - y) \*\* 2)

    return Result(

        coefficients = (a0, a1, a2),

        apply = lambda x: a2\*x\*\*2 + a1\*x + a0,

        function = f'{a2:.4f}x² + {a1:.4f}x + {a0:.4f}',

        S = S,

        deviation = msr(phi, y),

        confidence = confidence(phi, y)

    )

# полиномиальная функция 3-й степени - done

def Approximation\_degree3(x, y):

    x, y = getData(x, y)

    SX,SX2, SX3, SX4, SX5, SX6, SY,SXY, SX2Y, SX3Y = np.sum(x), np.sum(x\*\*2), np.sum(x\*\*3), np.sum(x\*\*4), np.sum(x\*\*5), np.sum(x\*\*6) , np.sum(y), np.sum(x\*y), np.sum(x\*x\*y), np.sum(x\*x\*x\*y)

    a0, a1, a2, a3 = np.linalg.solve(

        np.array([[len(x), SX, SX2, SX3], [SX, SX2, SX3, SX4], [SX2, SX3, SX4, SX5], [SX3, SX4, SX5, SX6]]),

        np.array([SY, SXY, SX2Y, SX3Y])

    )

     # Сумма квадратов отклонений

    phi = a3\*x\*\*3 + a2\*x\*\*2 + a1\*x + a0

    S = np.sum((phi - y) \*\* 2)

    return Result(

        coefficients = (a0, a1, a2),

        apply = lambda x: a3\*x\*\*3 + a2\*x\*\*2 + a1\*x + a0,

        function = f'{a3:.4f}x³ + {a2:.4f}x² + {a1:.4f}x + {a0:.4f}',

        S = S,

        deviation = msr(phi, y),

        confidence = confidence(phi, y)

    )

class LnException(Exception):

    pass

# логарифмическая функция - done

def Approximation\_logarith(x, y):

    x, y = getData(x, y)

    if x[x < 0]:

        raise LnException('x должен быть больше чем 0')

    X = np.log(x)

    a, b = Approximation\_linear(X, y).coefficients

    phi = a\*np.log(x) + b

    S = np.sum((phi - y)\*\* 2)

    return Result(

        coefficients = (a, b),

        apply = lambda x: a\*math.log(x) + b,

        function = f'{a:.4f}ln(x) + {b:.4f}',

        S = S,

        deviation = msr(phi, y),

        confidence = confidence(phi, y)

    )

#  степенная функция - done

def Approximation\_power(x, y):

    x, y = getData(x,y)

    X, Y = np.log(x), np.log(y)

    B, A = Approximation\_linear(X, Y).coefficients

    a, b = math.exp(A), B

    phi = a\*x\*\*b

    S = np.sum((phi - y)\*\* 2)

    return Result(

        coefficients = (a, b),

        apply = lambda x: a\*math.pow(x, b),

        function = f'{a:.4f}x^{b:.4f}',

        S = S,

        deviation = msr(phi, y),

        confidence = confidence(phi, y)

    )

#  экспоненциальная функция - done

def Approximation\_exp(x, y):

    x, y = getData(x, y)

    Y = np.log(y)

    B, A = Approximation\_linear(x, Y).coefficients

    a, b = math.exp(A), B

    phi = a\*np.exp(b\*x)

    S = np.sum((phi - y)\*\* 2)

    return Result(

        coefficients = (a, b),

        apply = lambda x: a\*math.exp(b\*x),

        function = f'{a:.4f}e^{b:.4f}x',

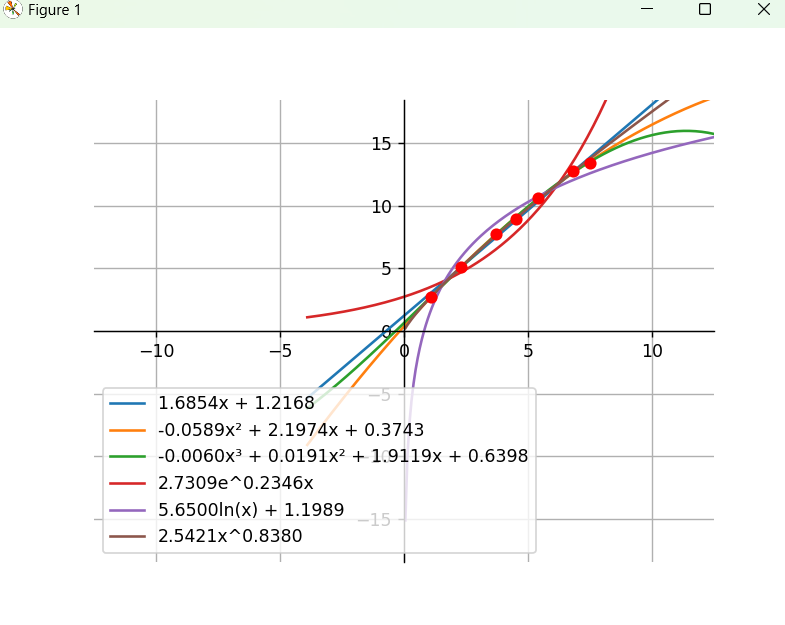
        S = S,

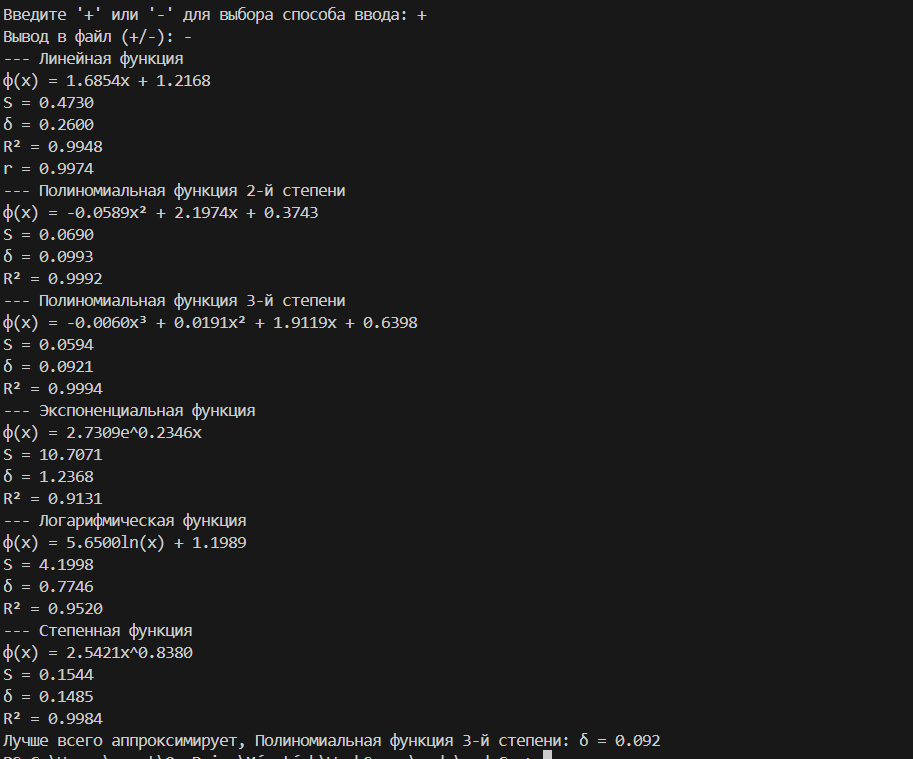
        deviation = msr(phi, y),

        confidence = confidence(phi, y)

    )

1. **Результаты выполнения программы**

****

****

1. **Выводы**

В результате выполнения данной лабораторной работой я познакомился с аппроксимация функции методом наименьших квадратов и реализовал 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.