**Министерство науки и высшего образования Российской Федерации**

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

Факультет программной инженерии и компьютерной техники

**Дисциплина:** Вычислительная математика

**Лабораторная работа №4**

**«Аппроксимация функции методом наименьших квадрадов»**

**Вариант №10**

Выполнил:

Мокров Семён Андреевич

Группа:

P3215

Проверила:

Малышева Татьяна Алексеевна

Санкт-Петербург

2022

# Цель работы

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

# Задание лабораторной работы

**Методика проведения исследования:**

* Вычислить меру отклонения: для всех исследуемых функций.
* Уточнить значения коэффициентов эмпирических функций, минимизируя функцию S.
* Сформировать массивы предполагаемых эмпирических зависимостей (.
* Определить среднеквадратичное отклонение для каждой аппроксимирующей функции. Выбрать наименьшее значение и, следовательно, наилучшее приближение.
* Построить графики полученных эмпирических функций.

**Вычислительная реализация задачи:**

* Для заданной функции (см. таблицу 1) построить наилучшие линейное и квадратичное приближения по 11 точкам указанного интервала.
* Найти среднеквадратические отклонения. Ответы дать с тремя знаками после запятой.
* Построить графики линейного и квадратичного приближений и заданной функции.
* ***Привести в отчете подробные вычисления***.

**Программная реализация задачи:**

* Предусмотреть ввод исходных данных из файла/консоли (таблица *y=f(x)* должна содержать 10 - 12 точек).
* Реализовать метод наименьших квадратов, исследуя все функции п.1.
* Предусмотреть вывод результатов в файл/консоль.
* Для линейной зависимости вычислить коэффициент корреляции Пирсона.
* Программа должна отображать наилучшую аппроксимирующую функцию.
* Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом).

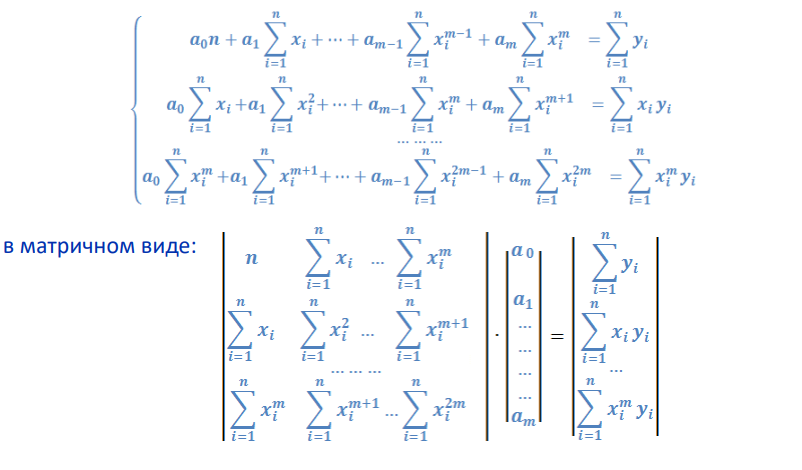
# Рабочие формулы

Параметры a0, a1, a2, ... , am эмпирической формулы находятся из условия минимума функции S = S(a0, a1, a2, ... , am). Так как здесь параметры выступают в роли независимых переменных функции S, то ее минимум найдем, приравнивая к нулю частные производные по этим переменным.

Изображение выглядит как текст

Автоматически созданное описание

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



В случае, когда мы проводим линейную (или сводим к линейной) аппроксимацию, то можно использовать формулы:

Изображение выглядит как текст

Автоматически созданное описание

# Вычислительная реализация задачи

**Исходная функция и ее значения в промежутке с шагом h = 0,2:**

Функция:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 0.2 | 0.4 | 0.6 | 0.8 | 1 | 1.2 | 1.4 | 1.6 | 1.8 | 2 |
|  | 0 | 0.999 | 0.200 | 0.294 | 0.374 | 0.429 | 0.446 | 0.427 | 0.382 | 0.327 | 0.273 |

**Вычисление коэффициентов:**

**Линейная аппроксимация:**

# Листинг программы

Весь исходный код программы располагается на GitHub (адрес: https://github.com/semwett0301/lab4\_math)

import math  
import numpy as np  
from math\_part import matrix\_interaction as mi  
  
  
def lin\_app(points):  
 s\_x = 0  
 s\_x\_x = 0  
 s\_y = 0  
 s\_x\_y = 0  
 for i in range(len(points)):  
 s\_x += points[i][0]  
 s\_x\_x += points[i][0] \*\* 2  
 s\_y += points[i][1]  
 s\_x\_y += points[i][0] \* points[i][1]  
  
 m\_x = s\_x / len(points)  
 m\_y = s\_y / len(points)  
  
 s\_m\_x\_y = 0  
 s\_m\_x = 0  
 s\_m\_y = 0  
 for i in range(len(points)):  
 s\_m\_x\_y += (points[i][0] - m\_x) \* (points[i][1] - m\_y)  
 s\_m\_x += (points[i][0] - m\_x) \*\* 2  
 s\_m\_y += (points[i][1] - m\_y) \*\* 2  
  
 # Коэффициент Пирсона  
 try:  
 r = round(s\_m\_x\_y / math.sqrt(s\_m\_x \* s\_m\_y), 3)  
 except Exception:  
 r = None  
  
 n = len(points)  
 delta = s\_x\_x \* n - s\_x \*\* 2  
 delta\_1 = s\_x\_y \* n - s\_x \* s\_y  
 delta\_2 = s\_x\_x \* s\_y - s\_x \* s\_x\_y  
  
 try:  
 a = round(delta\_1 / delta, 3)  
 b = round(delta\_2 / delta, 3)  
 result = lambda x: a \* x + b  
 result\_func = str(a) + "x + " + str(b)  
  
 s = round(sum([(points[i][1] - result(points[i][0])) \*\* 2 for i in range(n)]), 3)  
 sigma = round(math.sqrt(s / n), 3)  
 except Exception:  
 a = b = result\_func = s = result = sigma = None  
  
 return {  
 "name": "Линейная",  
 "function": result\_func,  
 "lambda": result,  
 "s": s,  
 "r": r,  
 "sigma": sigma  
 }  
  
  
def squad\_app(points):  
 n = len(points)  
 s\_x = s\_x\_x = s\_x\_x\_x = s\_x\_x\_x\_x = s\_y = s\_x\_y = s\_x\_x\_y = 0  
 for i in range(len(points)):  
 s\_x += points[i][0]  
 s\_x\_x += points[i][0] \*\* 2  
 s\_x\_x\_x += points[i][0] \*\* 3  
 s\_x\_x\_x\_x += points[i][0] \*\* 4  
 s\_y += points[i][1]  
 s\_x\_y += points[i][0] \* points[i][1]  
 s\_x\_x\_y += points[i][0] \* points[i][0] \* points[i][1]  
  
 matrix = [  
 [n, s\_x, s\_x\_x, s\_y],  
 [s\_x, s\_x\_x, s\_x\_x\_x, s\_x\_y],  
 [s\_x\_x, s\_x\_x\_x, s\_x\_x\_x\_x, s\_x\_x\_y]  
 ]  
  
 a\_0, a\_1, a\_2 = mi.calculating\_unknowns(matrix)  
  
 if a\_0 == a\_1 == a\_2 == 0:  
 function = s = sigma = result = None  
 else:  
 function = f"{round(a\_2, 3)}x^2 + {round(a\_1, 3)}x + {round(a\_0, 3)}"  
 result = lambda x: a\_2 \* x \*\* 2 + a\_1 \* x + a\_0  
 s = round(sum([(points[i][1] - result(points[i][0])) \*\* 2 for i in range(n)]), 3)  
 sigma = round(math.sqrt(s / n), 3)  
  
 return {  
 "name": "Квадратичная",  
 "function": function,  
 "lambda": result,  
 "s": s,  
 "sigma": sigma  
 }  
  
  
def qub\_appr(points):  
 n = len(points)  
 s\_x = s\_x\_2 = s\_x\_3 = s\_x\_4 = s\_x\_5 = s\_x\_6 = s\_y = s\_x\_y = s\_x\_2\_y = s\_x\_3\_y = 0  
  
 for i in range(len(points)):  
 s\_x += points[i][0]  
 s\_x\_2 += points[i][0] \*\* 2  
 s\_x\_3 += points[i][0] \*\* 3  
 s\_x\_4 += points[i][0] \*\* 4  
 s\_x\_5 += points[i][0] \*\* 5  
 s\_x\_6 += points[i][0] \*\* 6  
  
 s\_y += points[i][1]  
 s\_x\_y += points[i][0] \* points[i][1]  
 s\_x\_2\_y += points[i][0] \* points[i][0] \* points[i][1]  
 s\_x\_3\_y += points[i][0] \* points[i][0] \* points[i][0] \* points[i][1]  
  
 matrix = [  
 [n, s\_x, s\_x\_2, s\_x\_3, s\_y],  
 [s\_x, s\_x\_2, s\_x\_3, s\_x\_4, s\_x\_y],  
 [s\_x\_2, s\_x\_3, s\_x\_4, s\_x\_5, s\_x\_2\_y],  
 [s\_x\_3, s\_x\_4, s\_x\_5, s\_x\_6, s\_x\_3\_y]  
 ]  
  
 a\_0, a\_1, a\_2, a\_3 = mi.calculating\_unknowns(matrix)  
  
 if a\_0 == a\_1 == a\_2 == a\_3 == 0:  
 function = s = sigma = result = None  
 else:  
 function = f"{round(a\_3, 3)}x^3 + {round(a\_2, 3)}x^2 + {round(a\_1, 3)}x + {round(a\_0, 3)}"  
 result = lambda x: a\_3 \* x \*\* 3 + a\_2 \* x \*\* 2 + a\_1 \* x + a\_0  
 s = round(sum([(points[i][1] - result(points[i][0])) \*\* 2 for i in range(n)]), 3)  
 sigma = round(math.sqrt(s / n), 3)  
  
 return {  
 "name": "Кубическая",  
 "function": function,  
 "lambda": result,  
 "s": s,  
 "sigma": sigma  
 }  
  
  
def exp\_appr(points):  
 n = len(points)  
  
 for i in points:  
 if i[1] <= 0:  
 return {  
 "name": "Экспоненциальная",  
 "function": None,  
 "lambda": None,  
 "s": None,  
 "sigma": None  
 }  
  
 s\_x = s\_x\_2 = s\_y = s\_x\_y = 0  
 for i in range(n):  
 s\_x += points[i][0]  
 s\_x\_2 += points[i][0] \*\* 2  
 s\_y += math.log(points[i][1])  
 s\_x\_y += math.log(points[i][1]) \* points[i][0]  
  
 delta = s\_x\_2 \* n - s\_x \*\* 2  
 delta\_1 = s\_x\_y \* n - s\_x \* s\_y  
 delta\_2 = s\_x\_2 \* s\_y - s\_x \* s\_x\_y  
  
 try:  
 a = round(delta\_1 / delta, 3)  
 b = round(delta\_2 / delta, 3)  
 result = lambda x: np.exp(b) \* np.exp(a \* x)  
 function = f"{round(np.exp(b), 3)}e^({round(a, 3)}\*x)"  
  
 s = round(sum([(points[i][1] - result(points[i][0])) \*\* 2 for i in range(n)]), 3)  
 sigma = round(math.sqrt(s / n), 3)  
 except Exception:  
 a = b = function = s = sigma = result = None  
  
 return {  
 "name": "Экспоненциальная",  
 "function": function,  
 "lambda": result,  
 "s": s,  
 "sigma": sigma  
 }  
  
  
def log\_app(points):  
 n = len(points)  
  
 for i in points:  
 if i[1] <= 0:  
 return {  
 "name": "Логарифмическая",  
 "function": None,  
 "lambda": None,  
 "s": None,  
 "sigma": None  
 }  
 s\_x = s\_x\_2 = s\_y = s\_x\_y = 0  
 for i in range(n):  
 s\_x += math.log(points[i][0])  
 s\_x\_2 += math.log(points[i][0]) \*\* 2  
 s\_y += points[i][1]  
 s\_x\_y += math.log(points[i][0]) \* points[i][1]  
  
 delta = s\_x\_2 \* n - s\_x \*\* 2  
 delta\_1 = s\_x\_y \* n - s\_x \* s\_y  
 delta\_2 = s\_x\_2 \* s\_y - s\_x \* s\_x\_y  
  
 try:  
 a = round(delta\_1 / delta, 3)  
 b = round(delta\_2 / delta, 3)  
 result = lambda x: a \* np.log(x) + b  
  
 function = f"{round(a, 3)} \* ln(x) + {round(b, 3)}"  
  
 s = round(sum([(points[i][1] - result(points[i][0])) \*\* 2 for i in range(n)]), 3)  
 sigma = round(math.sqrt(s / n), 3)  
 except Exception:  
 a = b = result = function = s = sigma = None  
  
 return {  
 "name": "Логарифмическая",  
 "function": function,  
 "lambda": result,  
 "s": s,  
 "sigma": sigma  
 }  
  
  
def degree\_app(points):  
 n = len(points)  
  
 for i in points:  
 if i[1] <= 0 or i[0] <= 0:  
 return {  
 "name": "Степенная",  
 "function": None,  
 "lambda": None,  
 "s": None,  
 "sigma": None  
 }  
  
 s\_x = s\_x\_2 = s\_y = s\_x\_y = 0  
 for i in range(n):  
 s\_x += math.log(points[i][0])  
 s\_x\_2 += math.log(points[i][0]) \*\* 2  
 s\_y += math.log(points[i][1])  
 s\_x\_y += math.log(points[i][0]) \* math.log(points[i][1])  
  
 delta = s\_x\_2 \* n - s\_x \*\* 2  
 delta\_1 = s\_x\_y \* n - s\_x \* s\_y  
 delta\_2 = s\_x\_2 \* s\_y - s\_x \* s\_x\_y  
  
 try:  
 a = round(delta\_1 / delta, 3)  
 b = round(delta\_2 / delta, 3)  
 result = lambda x: math.exp(b) \* (x \*\* a)  
  
 function = f"{round(np.exp(b), 3)} \* x^{round(a, 3)}"  
  
 s = round(sum([(points[i][1] - result(points[i][0])) \*\* 2 for i in range(n)]), 3)  
 sigma = round(math.sqrt(s / n), 3)  
 except Exception:  
 a = b = result = function = s = sigma = None  
  
 return {  
 "name": "Степенная",  
 "function": function,  
 "lambda": result,  
 "s": s,  
 "sigma": sigma  
 }

# Примеры и результаты работы программы

**При выводе в файл:**

**Изображение выглядит как текст, монитор, снимок экрана, серебряный

Автоматически созданное описание**

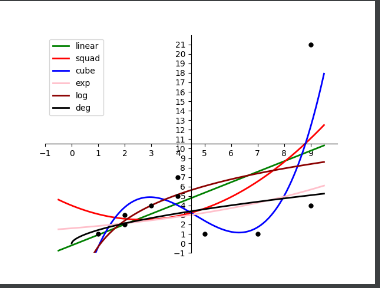
**При выводе на экран:**

Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание

**Графики функций:**

****

# Выводы

В данной лабораторной работе я:

* Аппроксимациями функциями методом наименьших квадратов;
* Ознакомился с принципами аппроксимации и оценкой точности аппроксимации исходных данных методом наименьших квадратов и наличия линейной зависимости между переменными (среднеквадратичное отклонение и коэффициент Пирсона);
* Реализовал методы аппроксимации методом наименьших квадратов на языке Python.