

ФАКУЛЬТЕТ: Информатика и системы управления

КАФЕДРА: Программное обеспечение ЭВМ и информационные технологии

**Лабораторная работа №4**

**Тема** Построение и программная реализация алгоритма наилучшего среднеквадратичного приближения.

**Студент** Зайцева А. А.

**Группа** ИУ7 – 42Б

**Оценка (баллы) \_\_\_\_\_\_\_**

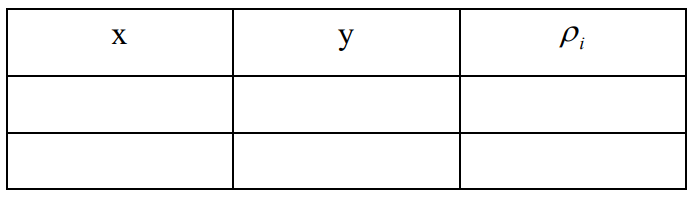
**Преподаватель** Градов В. М.

Москва.

2021 г

**Цель работы.** Получение навыков построения алгоритма метода наименьших квадратов с использованием полинома заданной степени при аппроксимации табличных функций с весами.

1. Исходные данные
   1. Таблица функции с весами ρi с количеством узлов N. Сформировать таблицу самостоятельно со случайным разбросом точек.



Предусмотреть в интерфейсе удобную возможность изменения пользователем весов в таблице.

1. Степень аппроксимирующего полинома n.
2. Код программы

Main.py – реализация алгоритма метода наименьших квадратов с использованием полинома заданной степени при аппроксимации табличных функций с весами.

|  |
| --- |
| import numpy as np import pandas as pd  # для работы с таблицами import my\_gaus  # реализация решения системы методом Гаусса import matplotlib.pyplot as plt  # для графиков   # таблица по умолчанию: x принимает значения [1, N] с шагом 1 # y - случайные значения на отрезке [1, N \* 2] # p - веса - все равны 1 def make\_table(N):    np.random.seed(1)    x = np.linspace(1, N, N, dtype=int)    y = np.random.randint(1, N \* 2 + 1, N)    p = np.array([1] \* N)    return x, y, p   # формирование левой части СЛАУ def find\_left\_part(n, N, x, p):    matrix = []    for k in range(n + 1):        row = []        for m in range(n + 1):            coeff = 0            for i in range(N):                coeff += p[i] \* (x[i] \*\* (k + m))            row.append(coeff)        matrix.append(row)     return matrix   # формирование правой части СЛАУ def find\_right\_part(n, N, x, y, p):    coeffs = []    for k in range(n + 1):        coeff = 0        for i in range(N):            coeff += p[i] \* y[i] \* (x[i] \*\* k)        coeffs.append(coeff)    return coeffs   # вычисление y с помошью коэффициентов полинома def count\_polynom(coeffs, x):    y = []    for x\_cur in x:        y\_cur = 0        for i in range(len(coeffs)):            y\_cur += coeffs[i] \* (x\_cur \*\* i)        y.append(y\_cur)    return y   # Вывод таблицы def output\_table(x, y, p):    print('Таблица:')    df = pd.DataFrame(data=zip(x, y, p),                      columns=['x', 'y', 'p'])    print(df)  # Отрисовка графиков def plot\_polynoms1(x, y, polynoms):    plt.figure(figsize=(7, 7))    plt.rcParams['font.size'] = '12'    plt.title('Аппроксимация функции методом наименьших квадратов')    plt.xlabel('x')    plt.ylabel('y')    plt.grid()     plt.scatter(x, y, label='dots')    for i in range(len(polynoms)):        plt.plot(x, count\_polynom(polynoms[i], x), label='n=%d'%(i+1))     plt.legend(loc='best')    plt.show()   def plot\_polynoms2(x, y, polynoms):    plt.figure(figsize=(7, 7))    plt.rcParams['font.size'] = '12'    plt.title('Аппроксимация функции методом наименьших квадратов')    plt.xlabel('x')    plt.ylabel('y')    plt.grid()     plt.scatter(x, y, label='dots')    plt.plot(x, count\_polynom(polynoms[0], x), label='pi=1')    plt.plot(x, count\_polynom(polynoms[1], x), label='pi изменены')     plt.legend(loc='best')    plt.show()   # первое задание def task\_1():    N = 5  # количество узлов    # создаем таблицу, где все веса равны 1    x, y, p = make\_table(N)    n\_min = 1    n\_max = 4  # максимальная степень полинома    polynoms = []  # массив для хранения полиномов    for n in range(n\_min, n\_max + 1):        # формируем левую и правую части СЛАУ        A = find\_left\_part(n, N, x, p)        B = find\_right\_part(n, N, x, y, p)        # находим решение методом Гаусса        coeffs = my\_gaus.solve\_gaus(A, B)        polynoms.append(coeffs)    print('Все веса равны 1')    output\_table(x, y, p)    plot\_polynoms1(x, y, polynoms)  # второе задание def task\_2():    N = 5  # количество узлов    # создаем таблицу, где все веса равны 1    x, y, p1 = make\_table(N)    p2 = [10, 1, 10, 1, 1]    n = 1  # степень полинома    polynoms = []  # массив для хранения полиномов    for p in [p1, p2]:        # формируем левую и правую части СЛАУ        A = find\_left\_part(n, N, x, p)        B = find\_right\_part(n, N, x, y, p)        # находим решение методом Гаусса        coeffs = my\_gaus.solve\_gaus(A, B)        polynoms.append(coeffs)    print('Все веса равны 1')    output\_table(x, y, p1)     print('Измененные веса')    output\_table(x, y, p2)    plot\_polynoms2(x, y, polynoms)   def main():    task\_1()    task\_2()   if \_\_name\_\_ == '\_\_main\_\_':    main() |

My\_gaus.py – решение СЛАУ методом Гаусса

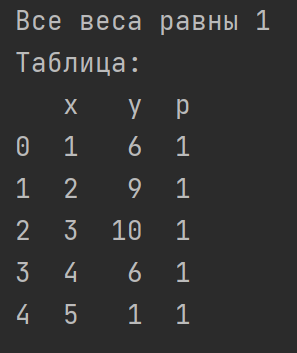
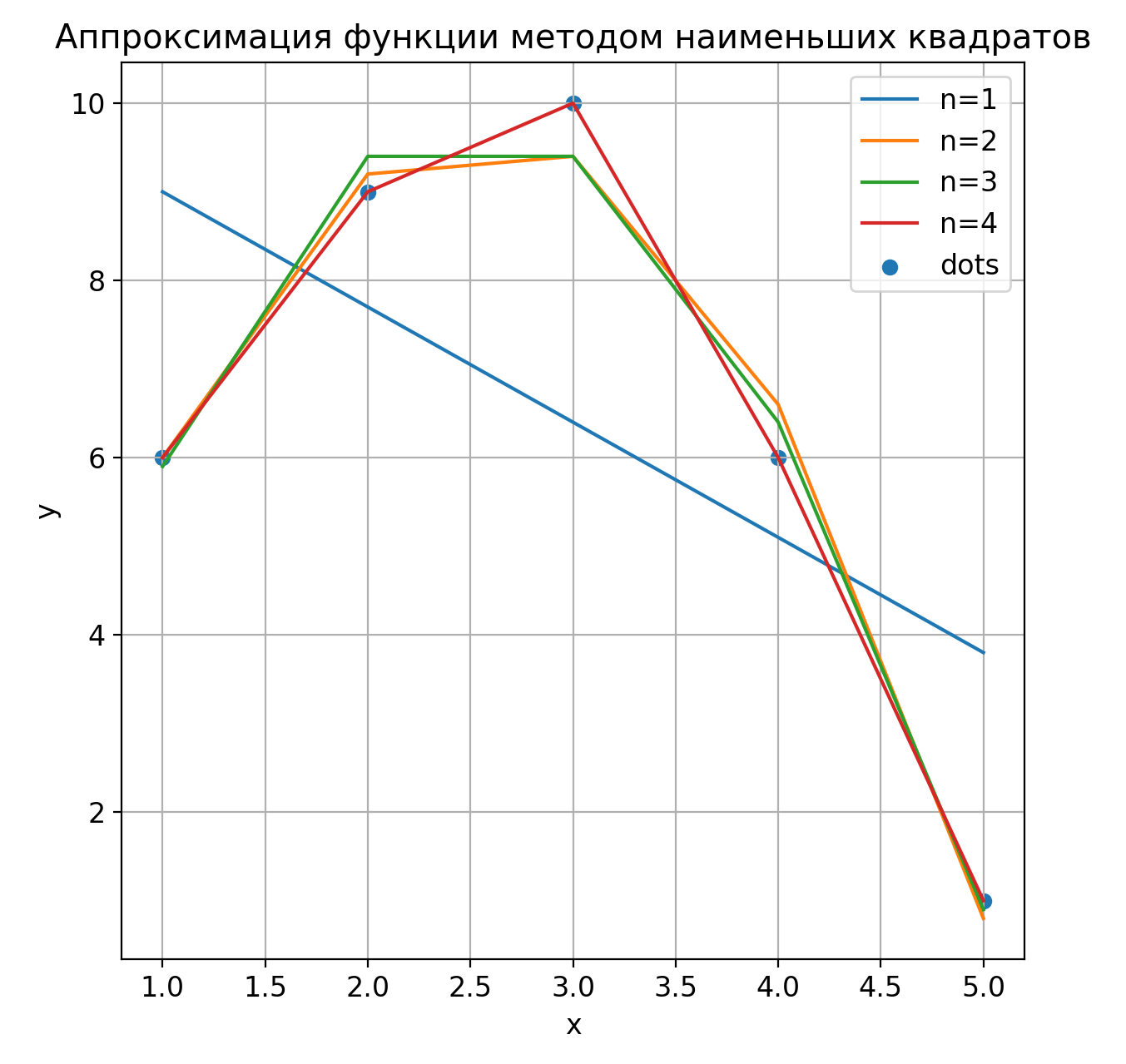
|  |
| --- |
| # поменять строки местами def rows\_swap(A, B, row1, row2):    A[row1], A[row2] = A[row2], A[row1]    B[row1], B[row2] = B[row2], B[row1]  # поделить строку на число def row\_div(A, B, row, divider):    A[row] = [a / divider for a in A[row]]    B[row] /= divider  # сложить сстроку с другой, умноженной на число def rows\_sum(A, B, row, source\_row, weight):    A[row] = [(a + k \* weight) for a, k in zip(A[row], A[source\_row])]    B[row] += B[source\_row] \* weight  # решение системы методом Гаусса def solve\_gaus(A, B):    column = 0    while column < len(B):        # Ищем максимальный по модулю элемент в column столбце        current\_row = None        for r in range(column, len(A)):            if current\_row is None or abs(A[r][column]) > abs(A[current\_row][column]):                 current\_row = r        if current\_row is None:            print("Ошибка: решений нет")            return None         if current\_row != column:            # Переставляем строку с найденным элементом наверх            rows\_swap(A, B, current\_row, column)        # Нормализуем строку с найденным элементом        row\_div(A, B, column, A[column][column])        # Обрабатываем нижележащие строки        for r in range(column + 1, len(A)):            rows\_sum(A, B, r, column, -A[r][column])        column += 1    # Матрица приведена к треугольному виду, считаем решение    X = [0] \* len(B)    for i in range(len(B) - 1, -1, -1):        X[i] = B[i] - sum(x \* a for x, a in zip(X[(i + 1):], A[i][(i + 1):]))    return X |

1. Результаты работы

Графики, построенные по аналогии с рис.1 в тексте Лекции №4: точки - заданная табличная функция, кривые - найденные полиномы. Обязательно приводить таблицы, по которым работала программа. При каких исходных условиях надо представить результаты в отчете?

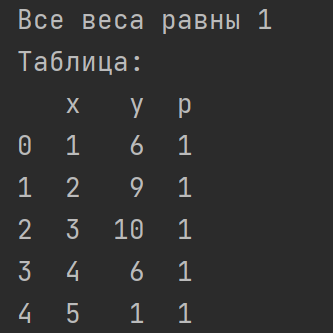
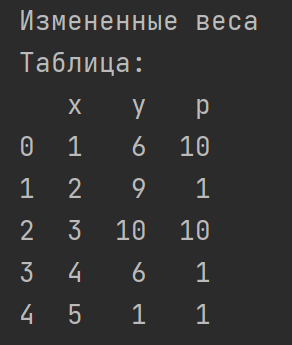
* 1. Веса всех точек одинаковы и равны, например, единице. Обязательно построить полиномы степеней n=1 и 2. Можно привести результаты и при других степенях полинома, однако, не загромождая сильно при этом рисунок.

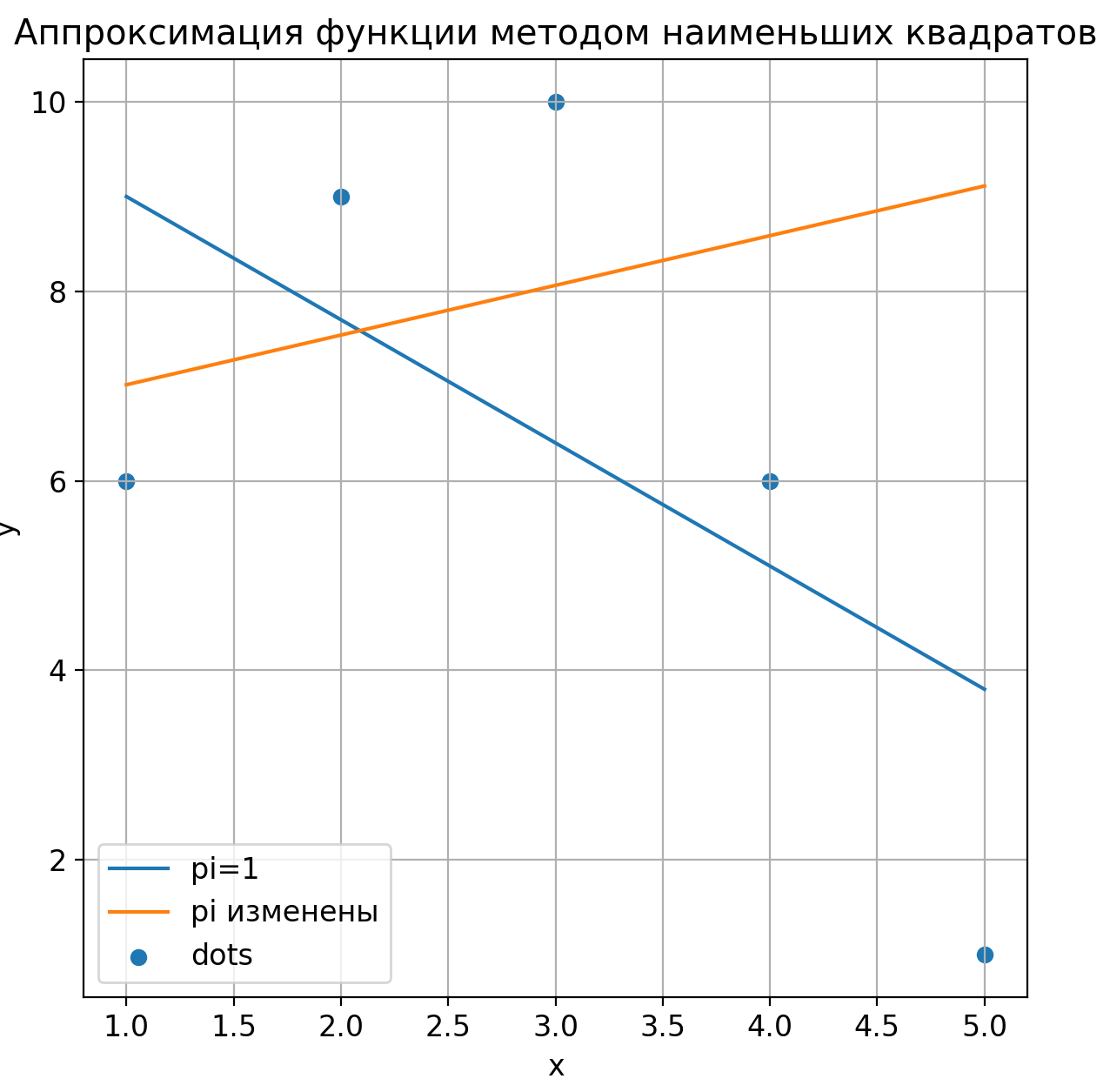
Количество точек N=5



С увеличением степени полинома до N-1 улучшается и аппроксимация.

* 1. Веса точек разные. Продемонстрировать, как за счет назначения весов точкам можно изменить положение на плоскости прямой линии (полином первой степени), аппроксимирующей один и тот же набор точек (одну таблицу y(x)). Например, назначая веса узлам в таблице изменить знак углового коэффициента прямой. На графике в итоге должны быть представлены точки исходной функции и две аппроксимирующие их прямые линии. Одна отвечает значениям ρ i =1 для всех узлов, а другая- назначенным разным весам точек. Информацию о том, какие именно веса были использованы в расчете обязательно указать, чтобы можно было проконтролировать работу программы (лучше это сделать в виде таблицы).

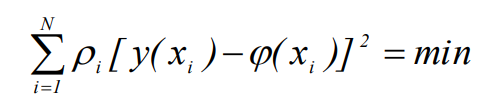
 



Увеличив веса у точек с индексами 0 и 2 в 10 раз, мы «вынудили» прямую изменить знак углового коэффициента прямой, чтобы приблизиться к этим точкам и тем самым минимизировать квадратичную ошибку с учетом весов.

1. Вопросы при защите лабораторной работы
   1. Что произойдет при задании степени полинома n=N-1 (числу узлов таблицы минус 1)?

Количество степеней свободы (коэффициентов в полиноме) будет равно n + 1= N количеству точек, и полином пройдет через каждую заданную точку (причем вне зависимости от весов, так как значения внутри скобок выражения

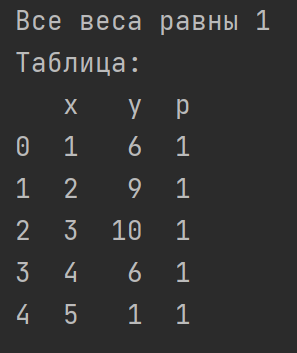


будут равны 0). Так происходит на первом приведенном графике: 5 точек и полином 4 степени.

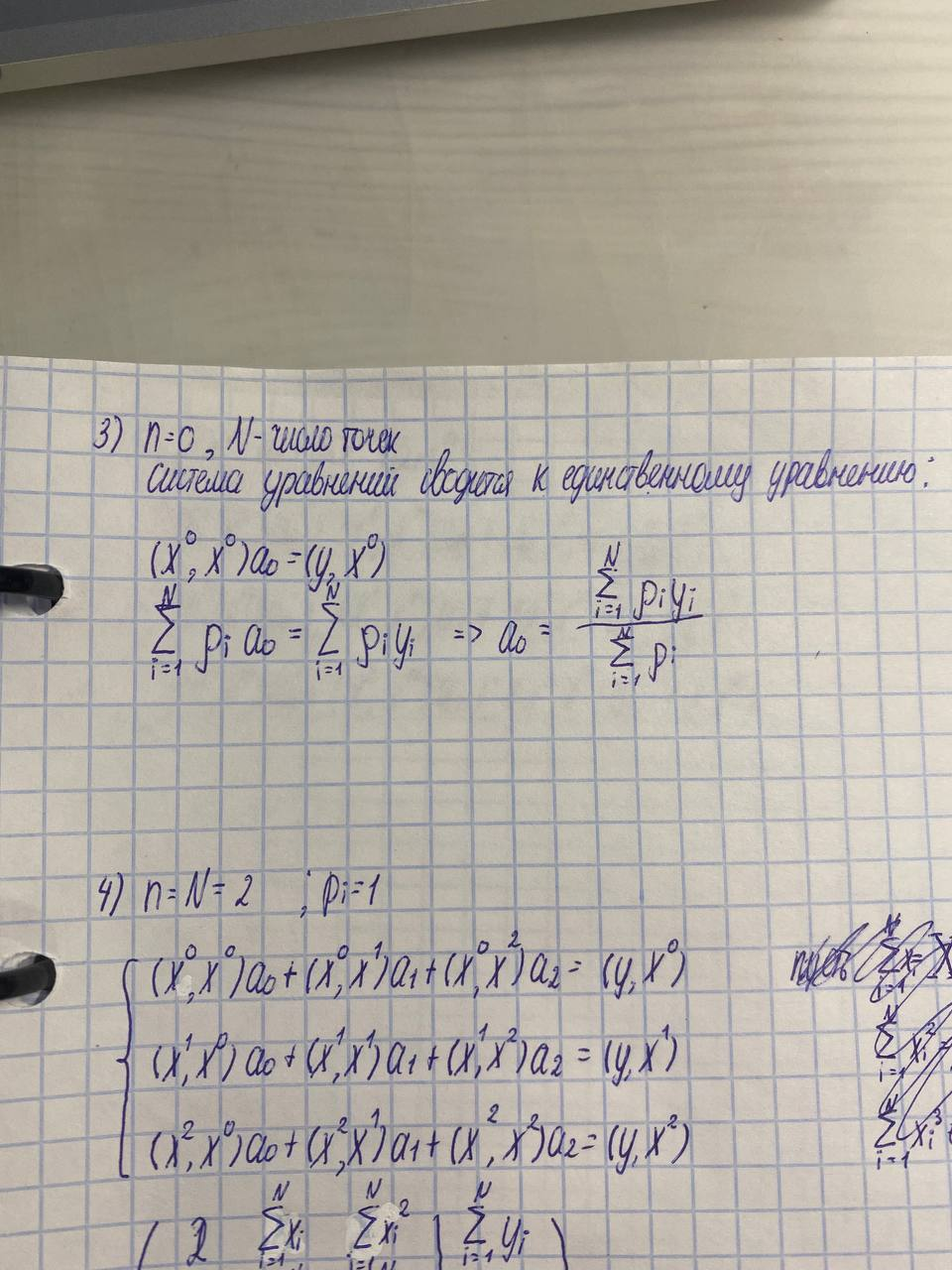
* 1. Будет ли работать Ваша программа при n >= N ? Что именно в алгоритме требует отдельного анализа данного случая и может привести к аварийной остановке?

При n >= N в составляемой СЛАУ появляются линейно-зависимые уравнения и ее определитель становится равен 0 (см. пункт 4). Тогда при приведении диагональной матрицы к единичной может произойти деление на ноль и программа аварийно завершится.

При моих исходных данных программа будет работать, но на заданном участке полиномы степеней n = N, N+1, N+2… совпадут с полиномом степени N-1

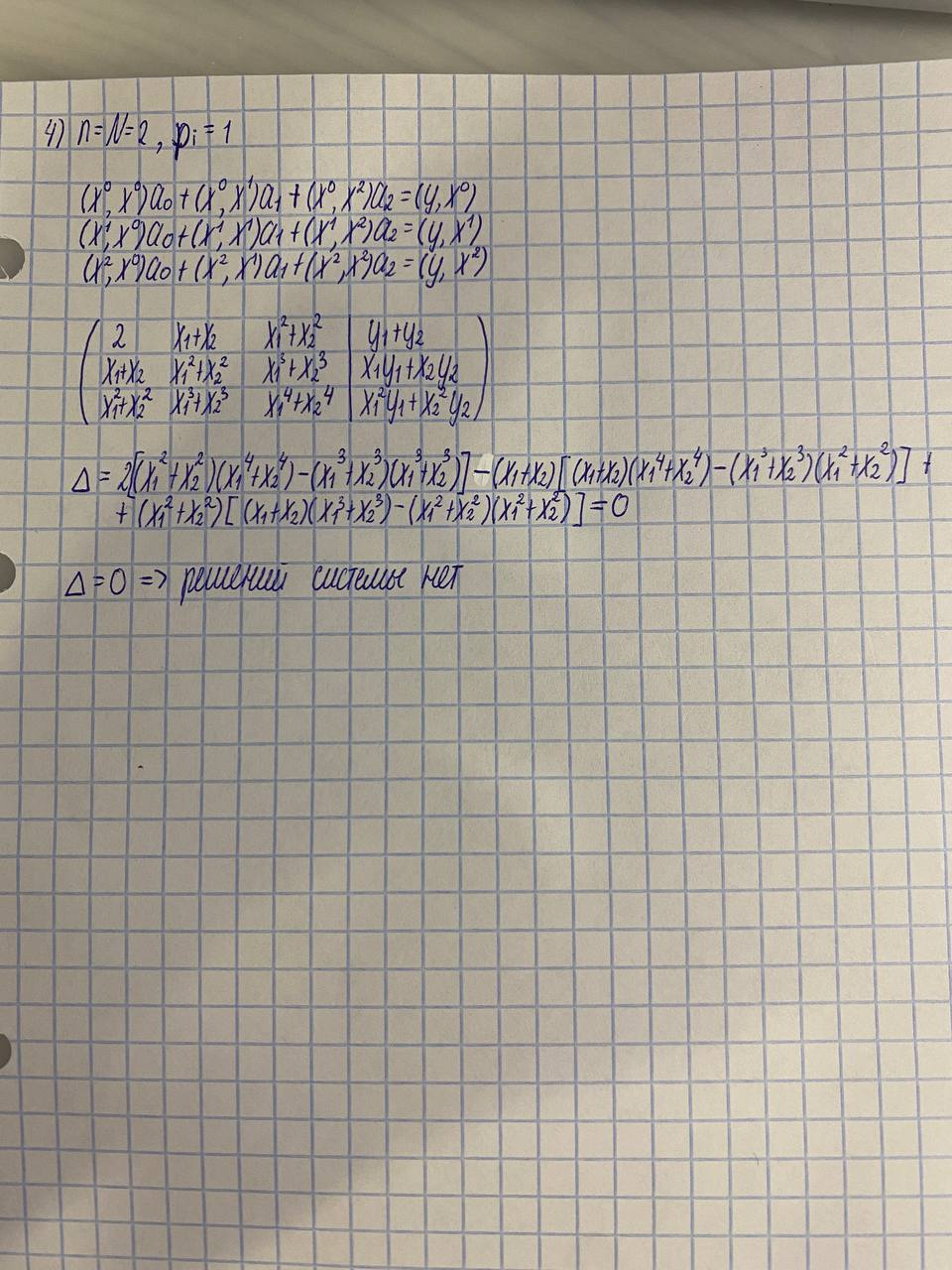


* 1. Получить формулу для коэффициента полинома a0 при степени полинома n=0. Какой смысл имеет величина, которую представляет данный коэффициент?

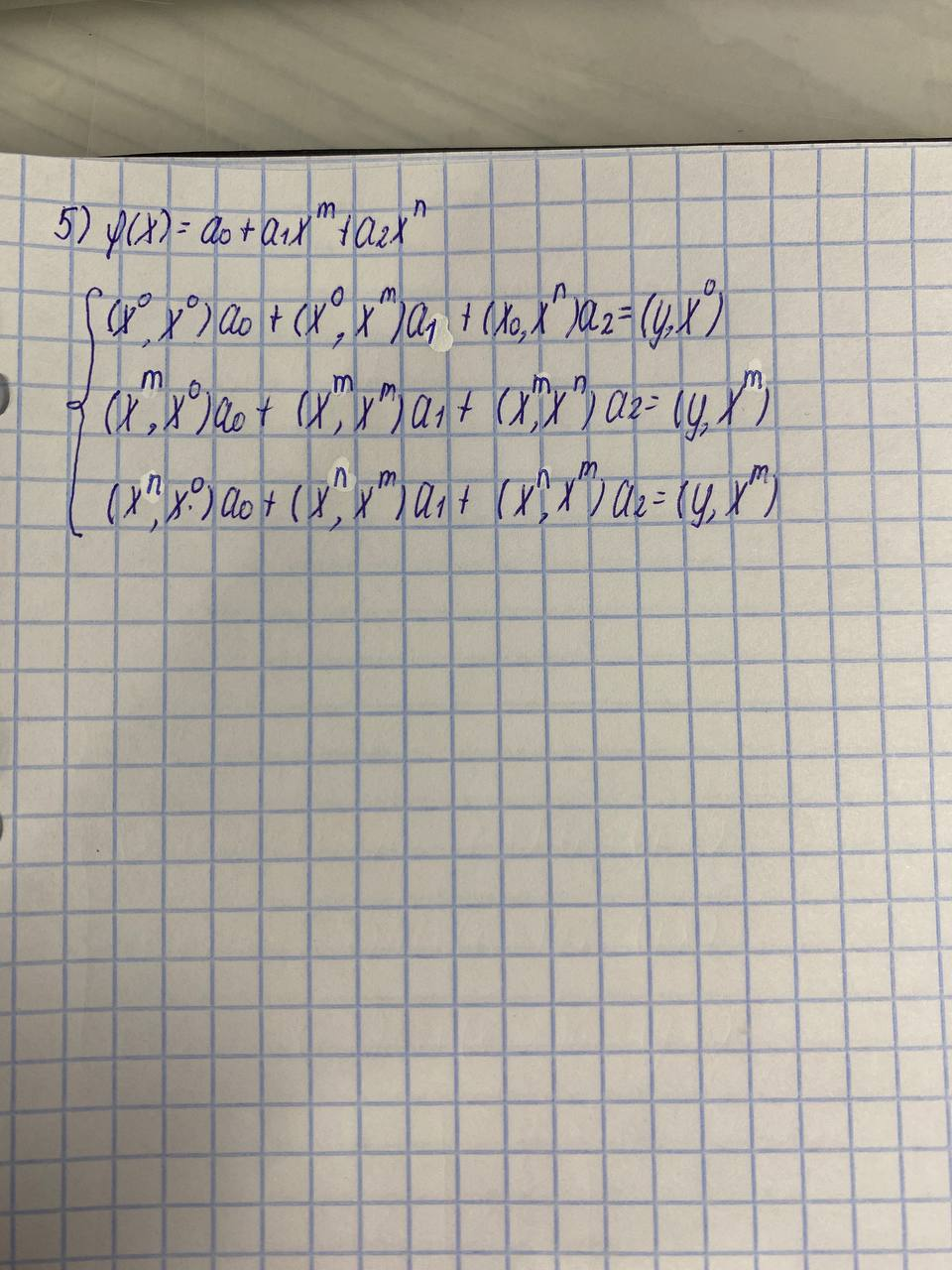


Смысл: это некоторый «центр тяжести» ординат точек (а если все веса точек равны 1, то это среднее значение всех y), и полином будет принимать это «центральное» значение при любых x.

* 1. Записать и вычислить определитель матрицы СЛАУ для нахождения коэффициентов полинома для случая, когда n=N=2. Принять все ρi =1.



* 1. Построить СЛАУ при выборочном задании степеней аргумента полинома , причем степени n и m в этой формуле известны.



* 1. Предложить схему алгоритма решения задачи из вопроса 5, если степени n и m подлежат определению наравне с коэффициентами ak , т.е. количество неизвестных равно 5.

Если у нас есть N различных точек, то можно составить СЛАУ из N-1 независимых переменных и построить полином n=N-1 степени. И тогда индексы ненулевых коэффициентов и есть искомые степени. Например, если N = 5, а в результате решения СЛАУ получены коэффициенты a0=5, a1=15, a2=a3=0, a4=25, то m=1, n=4