**Министерство науки и высшего образования Российской**

**Федерации**

**Федеральное государственное бюджетное образовательное**

**Учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

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** Пронин А.С.

**Группа** ИУ7-42Б

**Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Преподаватель** Градов В.М.

Москва.

2021 г

**Цель работы:** Получение навыков владения методами интерполяции таблично заданных функций с помощью кубических сплайнов.

# ****1. Исходные данные****

1. Таблица функции с весами с количеством узлов N. Сформировать таблицу

самостоятельно со случайным разбросом точек.

2. Степень аппроксимирующего полинома - n. (Вводятся пользователем)

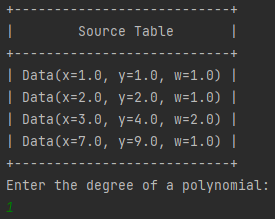
# 2. Код программы

from math import sin, pi, factorial, cos, exp, log  
import numpy as np  
import matplotlib.pyplot as plt  
from collections import namedtuple  
from prettytable import PrettyTable  
  
def phi(first\_point, second\_point):  
 return first\_point \*\* second\_point  
  
# Table generation:  
def creating\_table(file\_name):  
  
 input\_file = open(file\_name, 'r')  
  
 table\_data = PrettyTable()  
 data = []  
  
 for line in input\_file:  
 if line:  
 x, y, w = map(float, line.split())  
 data.append(Table(x, y, w))  
  
 table\_data.add\_column('Source Table', data)  
  
 print(table\_data)  
  
 input\_file.close()  
  
 return data  
  
  
# Finding the coefficient of the approximating function:  
def finding\_coefficient(table, degree):  
 SLAU, column\_of\_free = creating\_SLAU(table, degree)  
 inverse\_SLAU = inverse\_matrix(SLAU)  
 multiplication\_result = multiplication(column\_of\_free, inverse\_SLAU)  
 return multiplication\_result  
  
  
def creating\_SLAU(table, polynomial\_degree):  
 table\_size = len(table)  
 SLAU = [[0 for i in range(0, polynomial\_degree + 1)] for j in range(0, polynomial\_degree + 1)]  
 column\_of\_free = [0 for i in range(0, polynomial\_degree + 1)]  
 for m in range(0, polynomial\_degree + 1):  
 for i in range(0, table\_size):  
 buffer = table[i].w \* phi(table[i].x, m)  
  
 for k in range(polynomial\_degree + 1):  
 SLAU[m][k] += buffer \* phi(table[i].x, k)  
 column\_of\_free[m] += buffer \* table[i].y  
 return SLAU, column\_of\_free  
# To obtain the inverse matrix:  
def inverse\_matrix(matrix):  
 matrix\_size = len(matrix)  
 result = [[0 for i in range(0, matrix\_size)] for j in range(0, matrix\_size)]  
 for i in range(0, matrix\_size):  
 column = for\_column(matrix, i)  
 for j in range(0, matrix\_size):  
 result[j][i] = column[j]  
  
 return result  
  
  
# To convert a matrix into an inverse matrix:  
def for\_column(our\_matrix, column):  
 matrix\_size = len(our\_matrix)  
 mega\_matrix = [[our\_matrix[i][j] for j in range(matrix\_size)] for i in range(matrix\_size)]  
 new\_column = [0 for i in range(matrix\_size)]  
 for i in range(matrix\_size):  
 mega\_matrix[i].append(float(i == column))  
 for i in range(0, matrix\_size):  
 if mega\_matrix[i][i] == 0:  
 for j in range(i + 1, matrix\_size):  
 if mega\_matrix[j][j] != 0:  
 mega\_matrix[i], mega\_matrix[j] = mega\_matrix[j], mega\_matrix[i]  
 for j in range(i + 1, matrix\_size):  
 d = - mega\_matrix[j][i] / mega\_matrix[i][i]  
 for k in range(0, matrix\_size + 1):  
 mega\_matrix[j][k] += d \* mega\_matrix[i][k]  
 for i in range(matrix\_size - 1, -1, -1):  
 for\_result = 0  
 for j in range(matrix\_size):  
 for\_result += mega\_matrix[i][j] \* new\_column[j]  
 new\_column[i] = (mega\_matrix[i][matrix\_size] - for\_result) / mega\_matrix[i][i]  
 return new\_column  
  
  
# Multiplication of SLAU to column:  
def multiplication(column, SLAU):  
 column\_size = len(column)  
 result = [0 for j in range(column\_size)]  
 for j in range(column\_size):  
 for k in range(column\_size):  
 result[j] += column[k] \* SLAU[j][k]  
 return result

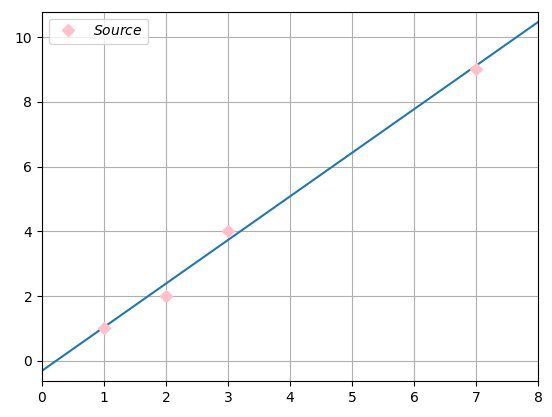
def print\_result(table, result, polynomial\_degree):  
  
 dx = 10  
  
 if len(table) > 1:  
 dx = (table[1].x - table[0].x)  
  
 # For approximating function:  
 x = np.linspace(table[0].x - dx, table[-1].x + dx, 100)  
 y = []  
 for i in x:  
 buffer = 0  
 for j in range(polynomial\_degree + 1):  
 buffer += phi(i, j) \* result[j]  
 y.append(buffer)  
  
 plt.plot(x, y)  
  
 x\_initial = [a.x for a in table]  
 y\_initial = [a.y for a in table]  
  
  
 plt.plot(x\_initial, y\_initial, 'kD', color = 'pink', label = '$Source$')  
  
 plt.grid(True)  
 plt.legend(loc = 'best')  
  
 min\_y = min(min(y), min(y\_initial))  
 max\_y = max(max(y), max(y\_initial))  
 dy = (max\_y - min\_y) \* 0.03  
  
 plt.axis([table[0].x - dx, table[-1].x + dx, min\_y - dy, max\_y + dy])  
  
 plt.show()  
 return  
  
Table = namedtuple('Data', ['x','y', 'w'])  
  
our\_table = creating\_table('data7.txt')  
  
polynomial\_degree = int(input('Enter the degree of a polynomial: \n'))  
  
result = finding\_coefficient(our\_table, polynomial\_degree)  
print\_result(our\_table, result, polynomial\_degree)

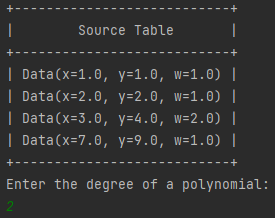
# 3. Результаты работы

1. Веса всех точек одинаковы и равны, например, единице. Обязательно построить полиномы степеней n = 1 и 2.

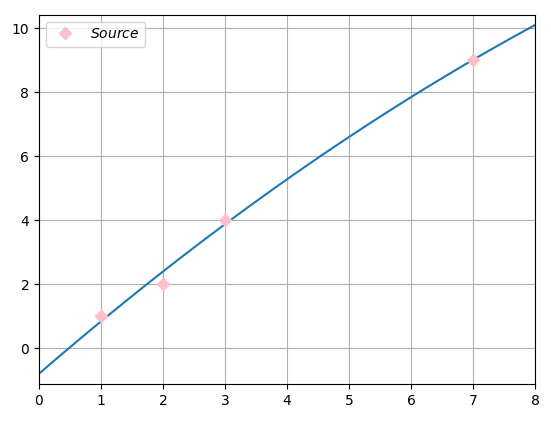


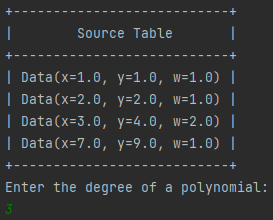
Степень полинома = 1



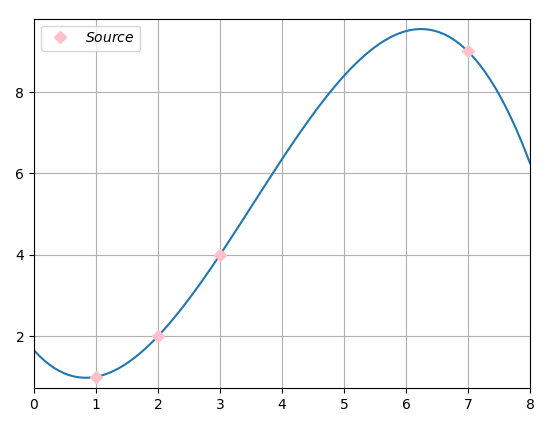


Степень полинома = 2

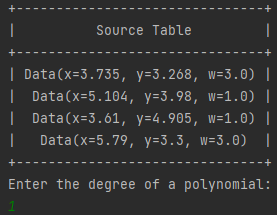


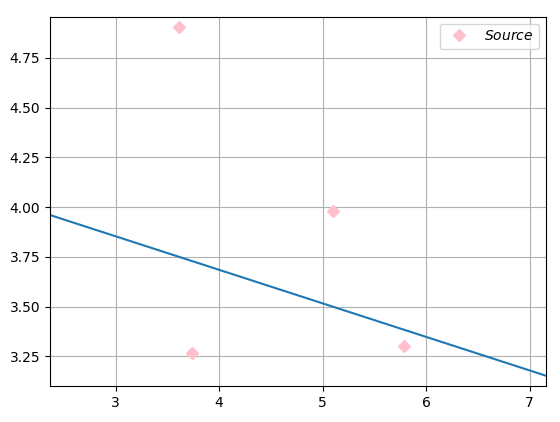


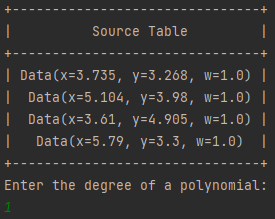
Степень полинома = 3



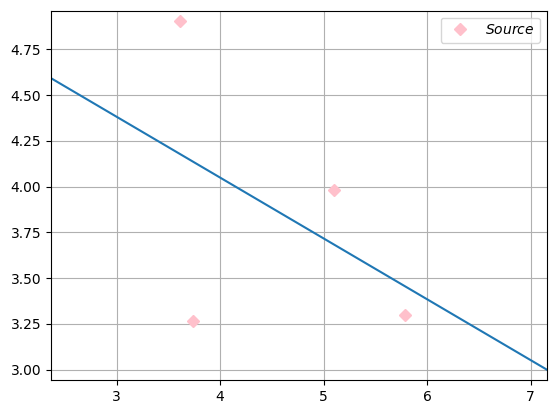
2. Веса точек разные.







Теже точки, но одинаковый вес



# 4. Вопросы при защите лабораторной работы

***1. Что произойдет при задании степени полинома n=N-1 (числу узлов таблицы минус 1)?***

Полученный полином пройдёт через все точки таблицы.

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

В поставленной ситуации СЛАУ не будет иметь решений, так как определитель матрицы будет равен 0. Это логично, так как нельзя построить кривую n-ой степени по n точкам. Программа может отработать на таких входных данных из-за погрешностей, что допускает случай, когда она работает, не так как ожидает пользователь. Разумно добавить дополнительную проверку данной ситуации.

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

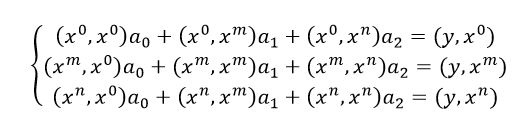
Математическое ожидание, где – вероятность встретить

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

Определитель будет равен 0 (см п.2).

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

Обнуляя остальные коэффициенты, получаем систему

******

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