МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3

по дисциплине «Системный анализ и обработка информации»

тема: «Аппроксимация функций по данным измерений методом наименьших квадратов с весовыми коэффициентами»

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Полунин А. И.

Белгород 2021 г.

**Цель работы:** изучить методы аппроксимации при наличии ошибок в измерениях функции.

**Вариант 3.**

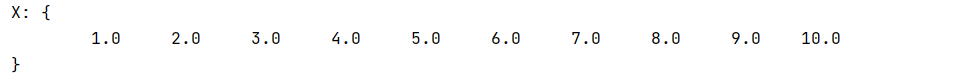
Функция:

Аппроксимирующий полином:

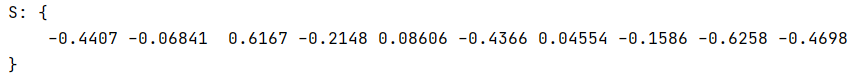
Число функций К: 4

σ измерения функции: 0.5

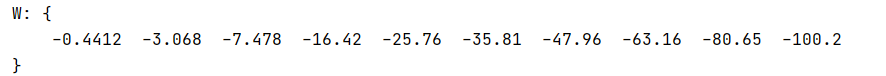
Вектор аргументов X:

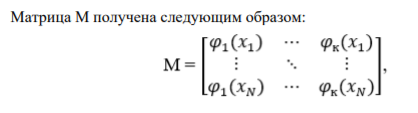


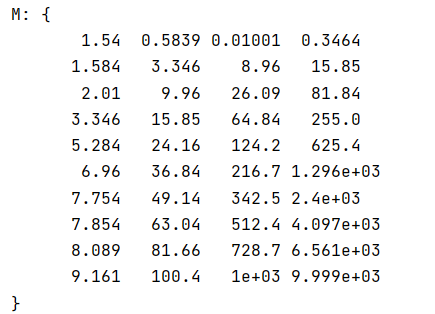
Вектор среднеквадратических погрешностей измерений S:

**

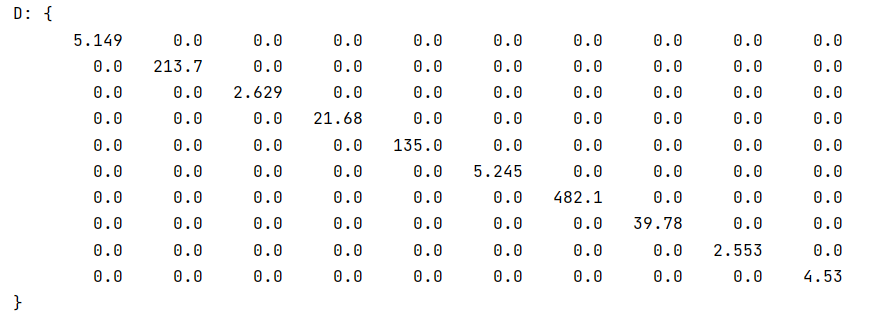
Вектор измерений W

**

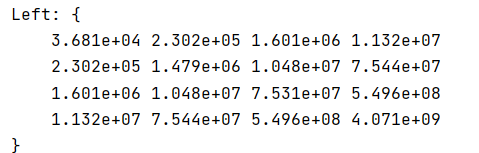
**

* # Импорт нужных библиотек.*

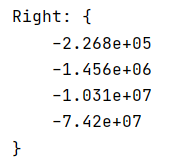
Диагональная матрица D, элементами которой являются величины 𝑆𝑖 −2 :



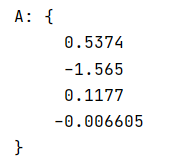
C:\Users\500a5\Desktop\07-04-2021 18-37-03.png

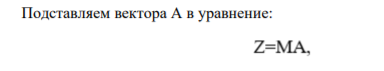


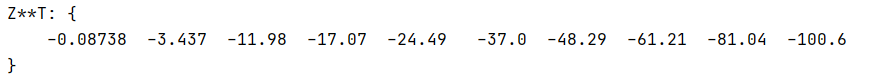
Правая часть:



Решаем систему методом Гаусса и получаем искомый вектор A:







**Текст программы**

from math import cos, log, e, floor,sin,pow  
from random import normalvariate  
from numpy.linalg import solve  
from matplotlib.pyplot import subplots, plot, scatter, legend, show  
  
  
*# Исходная функция.*def source\_func(x: float) -> float:  
 return (sin(cos(x)+x)-x\*x)  
  
*# Аппроксимирующий полином.*def approximating\_func(x: float, i: float) -> float:  
 return (cos(i\*x)+pow(x,i))  
  
  
*# Фунция вывода матрицы matrix с заголовком header.*def print\_matrix(header: str, matrix: list) -> None:  
 print(header, **'{'**, sep=**''**)  
 for row in matrix:  
 print(**'**\t**'**, sep=**''**, end=**''**)  
 for element in row:  
 print(**'{0:7.4}'**.format(element), end=**' '**)  
 print()  
 print(**'}'**)  
  
  
*# Возвращает транспонированную к матрице matrix.*def transpose\_matrix(matrix: list) -> list:  
 return [[matrix[i][j] for i in range(len(matrix))] for j in  
range(len(matrix[0]))]  
  
  
*# Возвращает диагольную матрицу, элементами которой являются array[i] ^power.*def diagonal\_powered\_matrix(array: list, power: float) -> list:  
 size = len(array)  
 matrix = [[0. for j in range(size)] for i in range(size)]  
 for i in range(size):  
 matrix[i][i] = array[i] \*\* power  
 return matrix  
  
  
*# Возвращает произведение матриц a и b.*def matrix\_product(a: list, b: list) -> list:  
 l\_size = len(a)  
 m\_size = len(b)  
 n\_size = len(b[0])  
  
 if m\_size != len(a[0]):  
 raise Exception(**'Sizes error.'**)  
  
 matrix = [[0. for j in range(n\_size)] for i in range(l\_size)]  
 for i in range(l\_size):  
 for j in range(n\_size):  
 for k in range(m\_size):  
 matrix[i][j] += a[i][k] \* b[k][j]  
 return matrix  
  
  
*# Возвращает значение в точке по вектору z с n точками в пределах [a, b].*def get\_approximate\_value(approximate\_matr: list, x: float, a: float, b: float, n: int) -> float:  
 dn = (b - a) / (n - 1)  
 index\_x = (x - a) / dn  
 floor\_x = int(floor(index\_x))  
 frac\_x = index\_x - floor\_x  
 if abs(frac\_x) < 0.0000000001:  
 return approximate\_matr[0][floor\_x]  
 else:  
 return frac\_x \* (approximate\_matr[0][floor\_x] + approximate\_matr[0][floor\_x + 1])  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 *# Диапозон значений.* a\_limit = 1.  
 b\_limit = 10.  
 *# Количество значений.* n = 10  
 *# Количество аппроксимирующих функций.* k = 4  
 *# Находим длину каждого отрезка.* dn = (b\_limit - a\_limit) / (n - 1)  
 *# Вектор точек (аргументов).* x = [[a\_limit + dn \* i for i in range(n)]]  
 *# Вектор значений функции.* y = [[source\_func(x) for x in x[0]]]  
 *# Математическое ожидание и среднеквадратическое отклонение.* mu = 0  
 sigma = 0.5  
 *# Вектор среднеквадратических погрешностей измерений.* s = [[normalvariate(mu, sigma) for i in range(n)]]  
 *# Вектор измерений.* w = [[y[0][i] + s[0][i] for i in range(n)]]  
 *# W^T.* wt = transpose\_matrix(w)  
 *# Матрица M.* m = [[approximating\_func(x[0][i], j + 1) for j in range(k)] for i in range(n)]  
 *# M^T.* mt = transpose\_matrix(m)  
 *# Диагональная матрцица D.* d = diagonal\_powered\_matrix(s[0], -2)  
 *# Произведение M^T \* D.* mt\_d = matrix\_product(mt, d)  
 *# Левая часть: M^T \* D \* M.* left = matrix\_product(mt\_d, m)  
 *# Правая часть: M^T \* D \* W^T.* right = matrix\_product(mt\_d, wt)  
 *# Вектора A получаем решением полученной системы.* a = solve(left, right)  
 *# Z = M \* A.* z = matrix\_product(m, a)  
 *# Z^T.* zt = transpose\_matrix(z)  
 if 1:  
 print\_matrix(**'X: '**, x)  
 print\_matrix(**'S: '**, s)  
 print\_matrix(**'W: '**, w)  
 print\_matrix(**'W\*\*T: '**, wt)  
 print\_matrix(**'M: '**, m)  
 print\_matrix(**'M\*\*T: '**, mt)  
 print\_matrix(**'D: '**, d)  
 print\_matrix(**'Left: '**, left)  
 print\_matrix(**'Right: '**, right)  
 print\_matrix(**'A: '**, a)  
 print\_matrix(**'Z: '**, z)  
 print\_matrix(**'X: '**, x)  
 print\_matrix(**'Y: '**, y)  
 print\_matrix(**'Z\*\*T: '**, zt)  
  
 print(**'Y(4.5)='**, source\_func(4.5))  
 print(**'Z(4.5)='**, get\_approximate\_value(zt, 4.5, a\_limit, b\_limit,n))  
 print(**'Y(9.5)='**, source\_func(9.5))  
 print(**'Z(9.5)='**, get\_approximate\_value(zt, 9.5, a\_limit, b\_limit,n))  
 fig, ax = subplots()  
 ax.scatter(x[0], w[0], label=**'W'**, color=**'gray'**, s=15)  
 ax.plot(x[0], zt[0], label=**'Z'**)  
 ax.legend()  
 show()

Результаты работы программы

При n=10,100,1000

