Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04

Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №4

Вариант 11

Выполнил: Савельева Диана Александровна

P32082

Преподаватель: Машина Екатерина Алексеевна

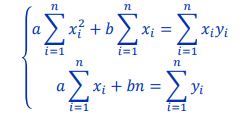
Санкт-Петербург, 2023 г.

***1. Цель:*** найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

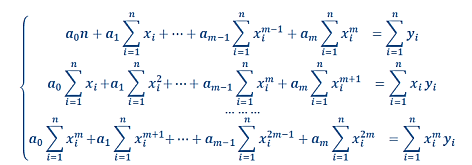
***2. Рабочие формулы метода:***

* *Линейная аппроксимация:*

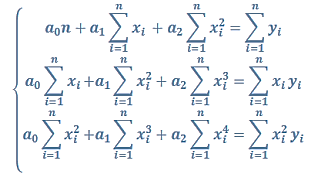
**

**

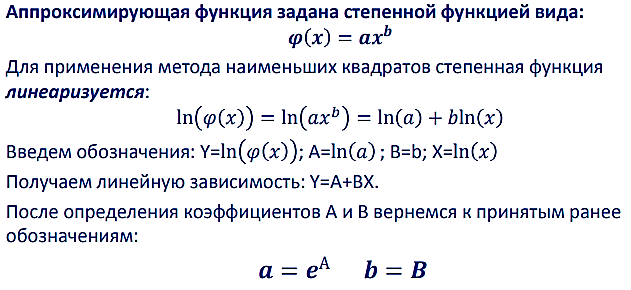
* *Полиномиальная аппроксимация:*

**

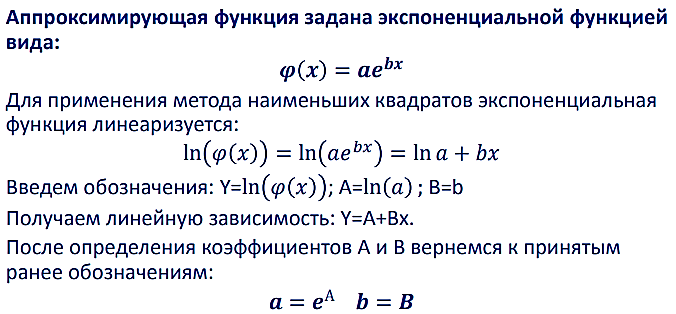
Квадратичная:

**

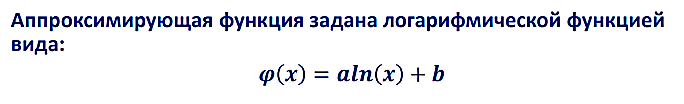
* *Степенная аппроксимация:*

**

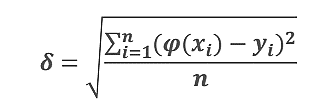
* *Экспоненциальная аппроксимация:*

**

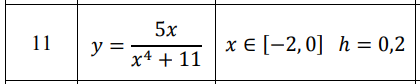
* *Логарифмическая аппроксимация:*

**

* *Среднеквадратическое отклонение:*

**

***3. Вычислительная часть:***

**

*Задание*

*1. Сформировать таблицу табулирования заданной функции на указанном интервале.*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *X* | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| *Y* | -0.3707 | -0,41865 | -0,45575 | -0,47165 | -0,45894 | -0,41667 | -0,35058 | -0,26955 | -0,1814 | -0,0909 | *0* |

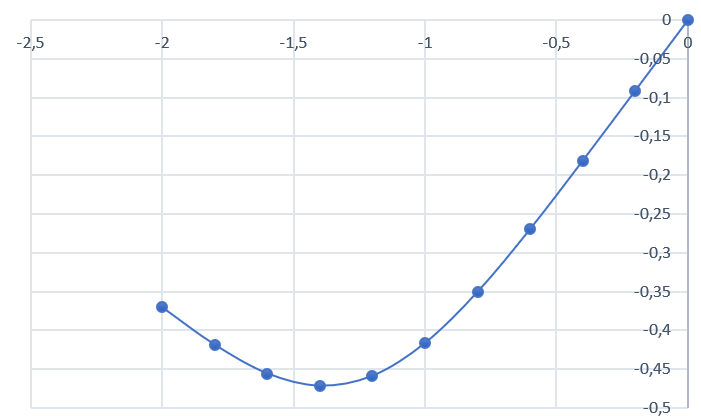
**

Рисунок 1 - Табулирование

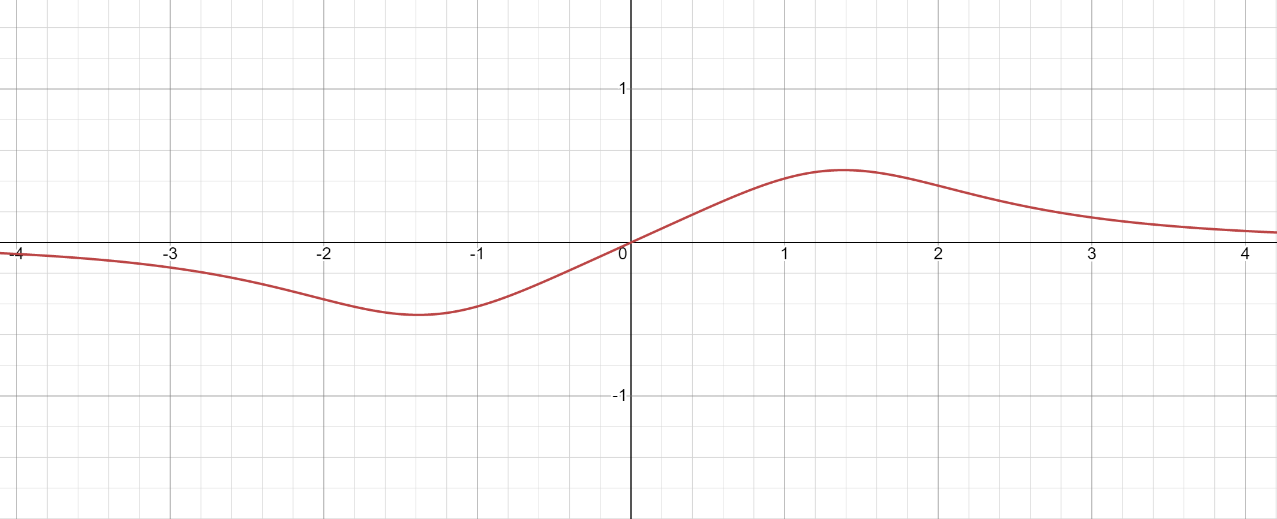
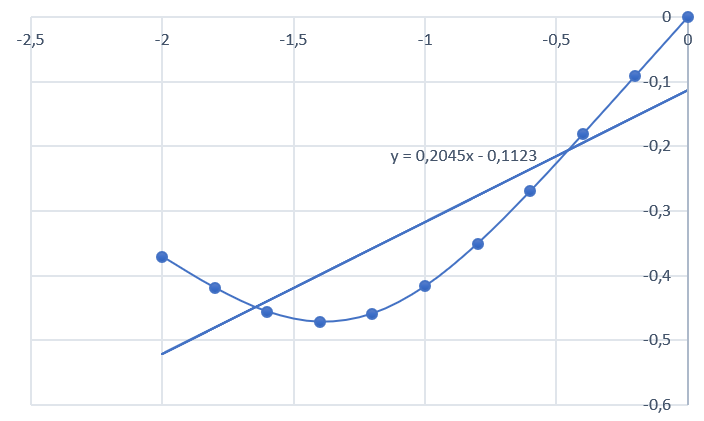
**

Рисунок 2 – График заданной функции

*2. Построить линейное и квадратичное приближения по 11 точкам заданного интервала.*

**Линейная апроксимация.**

Для определения вида зависимости нанесем экспериментальные точки на график: в качестве аппроксимирующей функции выбираем многочлен первой степени и строим линейную модель 𝑃1(𝑥) = 𝑎𝑥 + 𝑏.

**

Вычисляем суммы

* SX= -11.
* SXX = 15,4
* SY= -3,48445
* SXY=4,384143

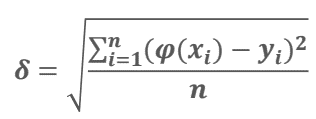
Получаем систему уравнений для нахождения параметров a и b:

Тогда коэффициенты a = 0,2045 и b = -0,1123.

Проверим правильность выбора линейной модели. Для этого вычислим значения аппроксимирующей функции P1(x) = 0,2045x-0,1123

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *X* | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| *Y* | -0.3707 | -0,41865 | -0,45575 | -0,47165 | -0,45894 | -0,41667 | -0,35058 | -0,26955 | -0,1814 | -0,0909 | 0 |
| *P1(x)* | -0,5213 | -0,4804 | -  -0,4395 | -0,3986 | -0,3577 | -0,3168 | -0,2759 | -0,5213 | -  -0,235 | -0,1532 | -0,1123 |
| *eps* | 0,15093 | 0,061749 | 0,016247 | 0,073047 | 0,10124 | 0,099867 | 0,074682 | 0,034551 | 0,012704 | 0,062304 | 0,1123 |

Мера отклонения: S = 0,076.

Среднеквадратичное отклонение =  = 0,083.

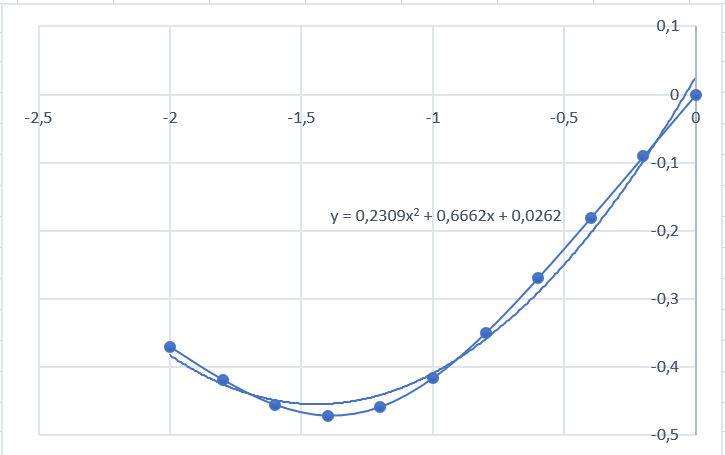
Наилучшее приближение: P1(x) = 0,2045x-0,1123.

**Вывод:** исследуемая функциональная зависимость может быть приближенно описана линейной моделью 𝑷𝟏(𝒙) = 0,2045x-0,1123, т.к.𝑷𝟏(𝒙𝒊 ) ≈ 𝒀𝒊 , 𝜺𝒊→min.

**Квадратичная аппроксимация.**

Для определения вида зависимости нанесем экспериментальные точки на график: в качестве аппроксимирующей функции выбираем многочлен второй степени и строим полиномиальную модель 𝑃2(𝑥) = a0 +a1x+a2x2.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *X* | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| *Y* | -0.3707 | -0,41865 | -0,45575 | -0,47165 | -0,45894 | -0,41667 | -0,35058 | -0,26955 | -0,1814 | -0,0909 | *0* |

****

Вычисляем суммы

* Сумма xi2= 15,4
* Сумма xi= -11
* Сумма yi= -3,484
* Сумма xi3= -24,2
* Сумма xi4= 40,533
* Сумма xiyi = 4,384
* Сумма x2iyi = -6,361

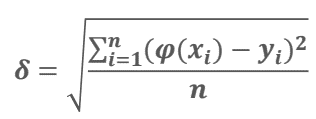
Получаем систему уравнений для нахождения параметров a0, a1 и a2:

Тогда коэффициенты a0 =0,0262, a1 = 0,6662, a2 = 0,2309.

Проверим правильность выбора линейной модели. Для этого вычислим значения аппроксимирующей функции P2(x) = 0,262+0,6662x+0,2309x2.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *X* | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| *Y* | -0.3707 | -0,41865 | -0,45575 | -0,47165 | -0,45894 | -0,41667 | -0,35058 | -0,26955 | -0,1814 | -0,0909 | 0 |
| *P1(x)* | -0,3826 | -0,4248 | -0,4486 | -0,4539 | -0,4407 | -0,4091 | -0,359 | -0,2904 | -0,2033 | -0,0978 | 0,0262 |
| *eps* | 0,01223 | 0,00619 | 0,00713 | 0,01773 | 0,0182 | 0,00757 | 0,0084 | 0,02084 | 0,02194 | 0,00691 | 0,0262 |

Мера отклонения: S = 0, 153.

Среднеквадратичное отклонение =  = 0,016

Наилучшее приближение: P2(x) = 0,262+0,6662x+0,2309x2.

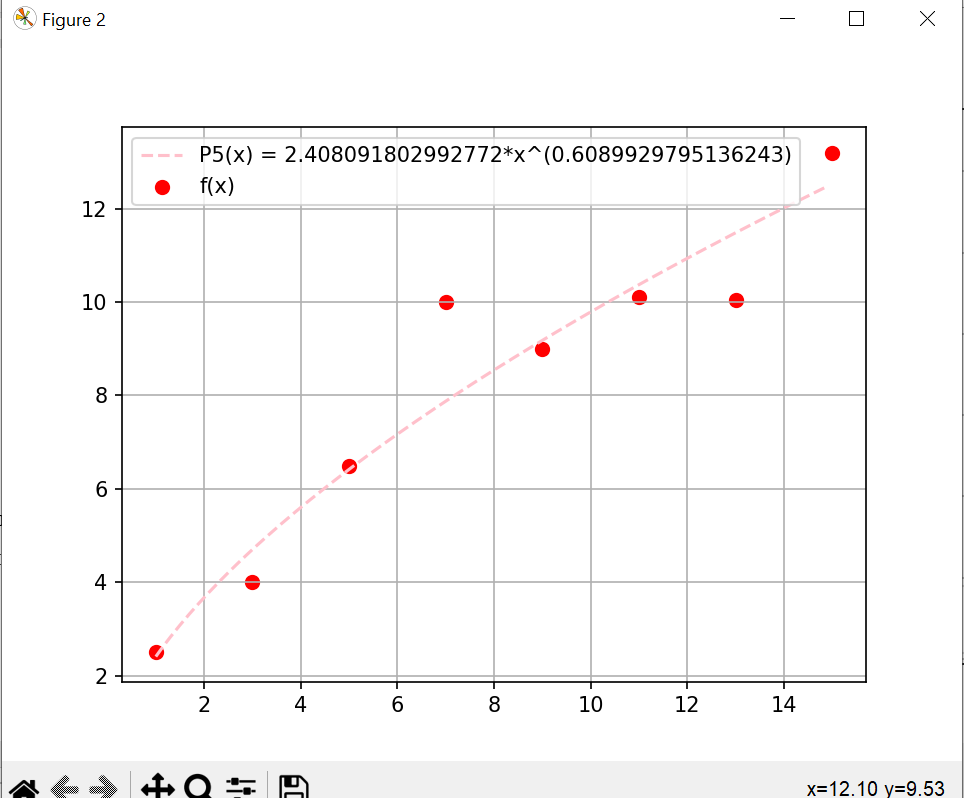
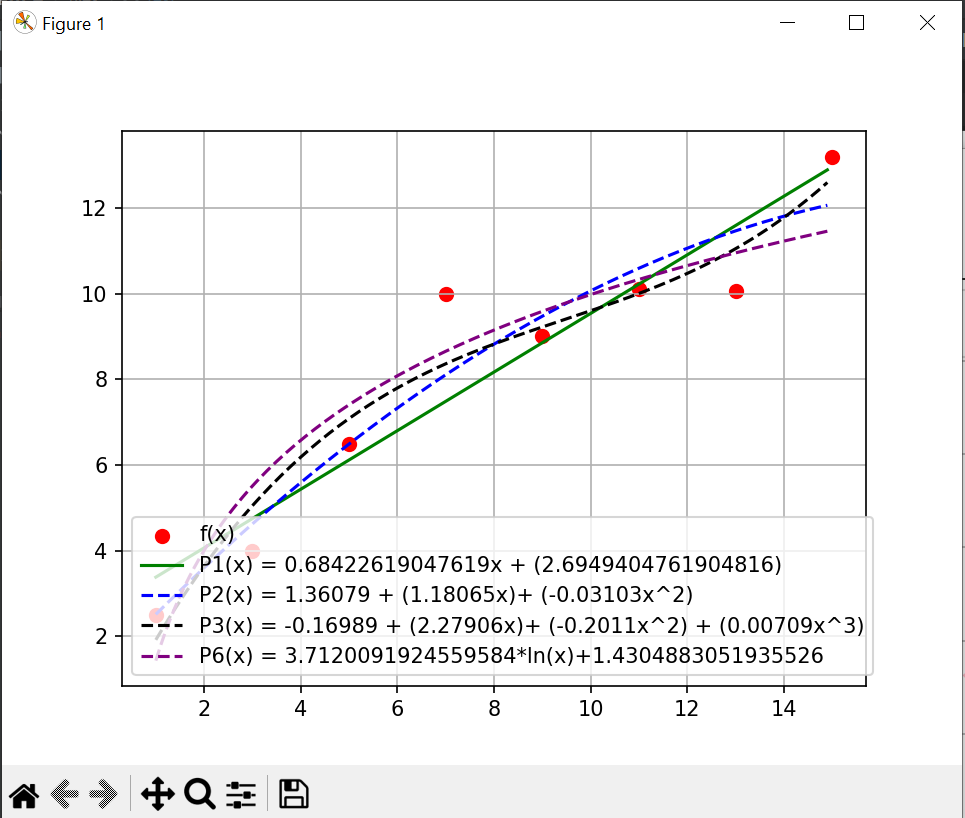
**Вывод:** исследуемая функциональная зависимость может быть приближенно описана линейной моделью 𝑷2(𝒙) = 0,262+0,6662x+0,2309x2, т.к.𝑷2(𝒙𝒊 ) ≈ 𝒀𝒊 , 𝜺𝒊→min.

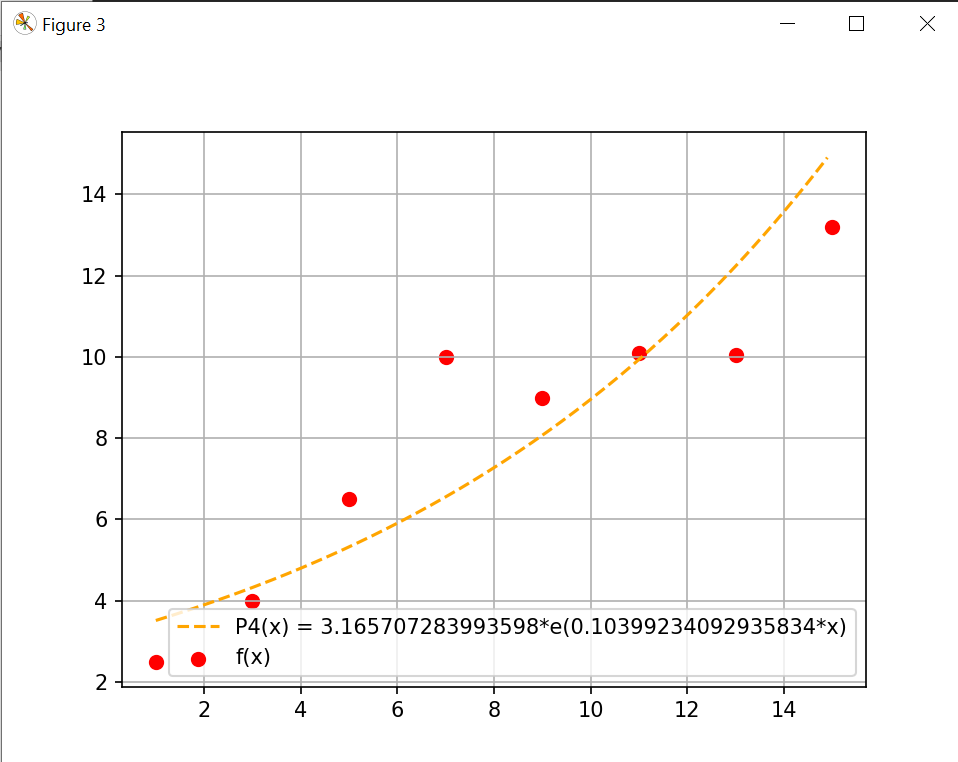
***4. Листинг программы:***

from numpy import sqrt, log, exp  
  
  
class Approximation:  
  
 def \_\_init\_\_(self, x\_arr, y\_arr):  
 self.\_\_x\_array = x\_arr  
 self.\_\_y\_array = y\_arr  
 self.\_\_n = len(self.\_\_x\_array)  
 self.\_\_p\_x = []  
 self.\_\_epsilon = []  
  
 def getArrayX(self):  
 return self.\_\_x\_array  
  
 def setArrayX(self, arr):  
 self.\_\_x\_array = arr  
  
 def getArrayY(self):  
 return self.\_\_y\_array  
  
 def setArrayY(self, arr):  
 self.\_\_y\_array = arr  
  
 def getN(self):  
 return self.\_\_n  
  
 def getEpsilon(self):  
 return self.\_\_epsilon  
  
 def \_setEpsilon(self, eps):  
 self.\_\_epsilon = eps  
  
 def getPX(self):  
 return self.\_\_p\_x  
  
 def \_setPX(self, p\_x):  
 self.\_\_p\_x = p\_x  
  
 def calculateSumX(self, degree):  
 return sum([x \*\* degree for x in self.\_\_x\_array])  
  
 def calculateSumY(self):  
 return sum(self.\_\_y\_array)  
  
 def calculateSumXY(self, degree):  
 return sum([self.\_\_x\_array[i] \*\* degree \* self.\_\_y\_array[i] for i in range(self.\_\_n)])  
  
 def \_calculateEpsilon(self):  
 return [self.\_\_p\_x[i] - self.\_\_y\_array[i] for i in range(self.\_\_n)]  
  
 def \_calculateExponentialValues(self, a, b):  
 return [a \* x \*\* b for x in self.\_\_x\_array]  
  
 def \_calculateLogarithmValues(self, a, b):  
 return [a \* log(x) + b for x in self.\_\_x\_array]  
  
 def \_calculateExponentValues(self, a, b):  
 return [a \* exp(b \* x) for x in self.\_\_x\_array]  
  
 def \_calculateLinearValues(self, a, b):  
 return [a \* x + b for x in self.\_\_x\_array]  
  
 def \_calculatePolynomialSecondValues(self, a0, a1, a2):  
 return [a0 + a1 \* x + a2 \* x \*\* 2 for x in self.\_\_x\_array]  
  
 def \_calculatePolynomialThirdValues(self, a0, a1, a2, a3):  
 return [a0 + a1 \* x + a2 \* x \*\* 2 + a3 \* x \*\* 3 for x in self.\_\_x\_array]  
  
 def \_deviationMeasure(self):  
 return sum([eps \*\* 2 for eps in self.\_\_epsilon])  
  
 def \_standardDeviation(self):  
 return sqrt(self.\_deviationMeasure() / self.\_\_n)  
  
 def \_printTable(self):  
 print('| X | Y | P(x) | epsilon |')  
 for i in range(self.\_\_n):  
 print(f"| {self.\_\_x\_array[i]} | {self.\_\_y\_array[i]} | {self.\_\_p\_x[i]} | {self.\_\_epsilon[i]} |")  
  
 def \_coefficientCorrelation(self):  
 x\_mean = self.\_calculateMeanX()  
 y\_mean = self.\_calculateMeanY()  
 return sum([(self.\_\_x\_array[i] - x\_mean) \* (self.\_\_y\_array[i] - y\_mean)  
 for i in range(self.\_\_n)]) / sqrt(  
 sum([(self.\_\_x\_array[i] - x\_mean) \*\* 2 for i in range(self.\_\_n)]) \*  
 sum([(self.\_\_y\_array[i] - y\_mean) \*\* 2 for i in range(self.\_\_n)]))  
  
 def \_calculateMeanX(self):  
 return sum(self.\_\_x\_array) / self.\_\_n  
  
 def \_calculateMeanY(self):  
 return sum(self.\_\_y\_array) / self.\_\_n

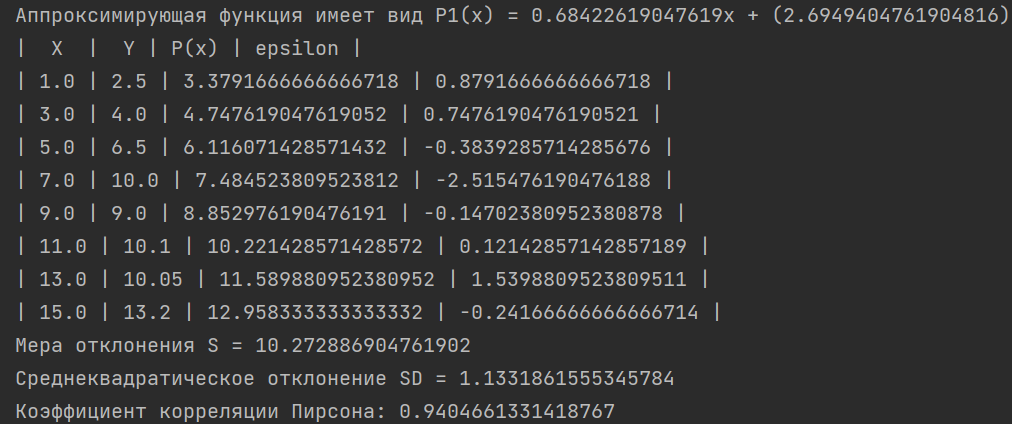
import math  
  
from numpy import exp, log  
  
from Approximation import Approximation  
import numpy as np  
  
  
class ApproximationFunctions(Approximation):  
  
 def \_\_calculateLinearFunctionCoefficient(self):  
 SX = self.calculateSumX(1)  
 SXX = self.calculateSumX(2)  
 SY = self.calculateSumY()  
 SXY = self.calculateSumXY(1)  
 N = self.getN()  
 delta = SXX \* N - SX \* SX  
 delta1 = SXY \* N - SX \* SY  
 delta2 = SXX \* SY - SX \* SXY  
 a = delta1 / delta  
 b = delta2 / delta  
 self.\_setPX(self.\_calculateLinearValues(a, b))  
 self.\_setEpsilon(self.\_calculateEpsilon())  
 S = self.\_deviationMeasure()  
 SD = self.\_standardDeviation()  
 return {'SX': SX, 'SXX': SXX, 'SY': SY,  
 'SXY': SXY, 'N': N, 'delta': delta,  
 'delta1': delta1, 'delta2': delta2,  
 'a': a, 'b': b, 'S': S, 'SD': SD}  
  
 def linearFunction(self):  
 coefficients = self.\_\_calculateLinearFunctionCoefficient()  
 print('\t\t\tЛинейная аппроксимация')  
 print(f"\tCистема уравнений:\n"  
 f"\t\t{coefficients['SXX']}a + {coefficients['SX']}b = {coefficients['SXY']}\n"  
 f"\t\t{coefficients['SX']}a + {coefficients['N']}b = {coefficients['SY']}")  
 print(f"\tИз нее находим:\n"  
 f"\t\tdelta = {coefficients['SXX']}\*{coefficients['N']} - {coefficients['SX']}\*{coefficients['SX']} = {coefficients['delta']}\n"  
 f"\t\tdelta1 = {coefficients['SXY']}\*{coefficients['N']} - {coefficients['SX']}\*{coefficients['SY']} = {coefficients['delta1']}\n"  
 f"\t\tdelta2 = {coefficients['SXX']}\*{coefficients['SY']} - {coefficients['SX']}\*{coefficients['SXY']} = {coefficients['delta2']}\n"  
 f"\t\t\ta = {coefficients['delta1']}/{coefficients['delta']} = {coefficients['a']}\n"  
 f"\t\t\tb = {coefficients['delta2']}/{coefficients['delta']} = {coefficients['b']}\n")  
 function = f"P1(x) = {coefficients['a']}x + ({coefficients['b']})"  
 self.\_\_printCoefficients(coefficients, function)  
 print(f"Коэффициент корреляции Пирсона: {self.\_coefficientCorrelation()}")  
 return function, {'a': coefficients['a'], 'b': coefficients['b']}, coefficients['S'], coefficients['SD']  
  
 def \_\_calculatePolynomialSecondFunctionCoefficient(self):  
 SX = self.calculateSumX(1)  
 SXX = self.calculateSumX(2)  
 SXXX = self.calculateSumX(3)  
 SXXXX = self.calculateSumX(4)  
 SY = self.calculateSumY()  
 SXY = self.calculateSumXY(1)  
 SXXY = self.calculateSumXY(2)  
 N = self.getN()  
 M1 = np.array([[N, SX, SXX], [SX, SXX, SXXX], [SXX, SXXX, SXXXX]]) # Матрица (левая часть системы)  
 V1 = np.array([SY, SXY, SXXY]) # Вектор (правая часть системы)  
 a0, a1, a2 = np.linalg.solve(M1, V1)  
 self.\_setPX(self.\_calculatePolynomialSecondValues(a0, a1, a2))  
 self.\_setEpsilon(self.\_calculateEpsilon())  
 S = self.\_deviationMeasure()  
 SD = self.\_standardDeviation()  
 return {'SX': SX, 'SXX': SXX, 'SXXX': SXXX, 'SXXXX': SXXXX,  
 'SY': SY, 'SXY': SXY, 'SXXY': SXXY, 'N': N,  
 'a0': round(a0, 5), 'a1': round(a1, 5), 'a2': round(a2, 5), 'S': S, 'SD': SD}  
  
 def polynomialSecondFunction(self):  
 print('\t\t\tПолиномиальная функция 2-ой степени')  
 coefficients = self.\_\_calculatePolynomialSecondFunctionCoefficient()  
 function = f"P2(x) = {coefficients['a0']} + ({coefficients['a1']}x)+ ({coefficients['a2']}x^2)"  
 print(f"\tПолучаем систему уравнений:\n"  
 f"\t\t{coefficients['N']}a0 + {coefficients['SX']}a1 + {coefficients['SXX']}a2 = {coefficients['SY']}\n"  
 f"\t\t{coefficients['SX']}a0 + {coefficients['SXX']}a1 + {coefficients['SXXX']}a2= {coefficients['SXY']}\n"  
 f"\t\t{coefficients['SXX']}a0 + {coefficients['SXXX']}a1 + {coefficients['SXXXX']}a2= {coefficients['SXXY']}\n")  
 print(f"\tНайденные коэффициенты:\n"  
 f"\t\ta0 = {coefficients['a0']}\n"  
 f"\t\ta1 = {coefficients['a1']}\n"  
 f"\t\ta2 = {coefficients['a2']}\n")  
 self.\_\_printCoefficients(coefficients, function)  
 return function, {'a0': coefficients['a0'], 'a1': coefficients['a1'], 'a2': coefficients['a2']}, \  
 coefficients['S'], coefficients['SD']  
  
 def \_\_calculatePolynomialThirdFunctionCoefficient(self):  
 SX = self.calculateSumX(1)  
 SXX = self.calculateSumX(2)  
 SXXX = self.calculateSumX(3)  
 SXXXX = self.calculateSumX(4)  
 SXXXXX = self.calculateSumX(5)  
 SXXXXXX = self.calculateSumX(6)  
 SY = self.calculateSumY()  
 SXY = self.calculateSumXY(1)  
 SXXY = self.calculateSumXY(2)  
 SXXXY = self.calculateSumXY(3)  
 N = self.getN()  
 M1 = np.array([[N, SX, SXX, SXXX], [SX, SXX, SXXX, SXXXX], [SXX, SXXX, SXXXX, SXXXXX],  
 [SXXX, SXXXX, SXXXXX, SXXXXXX]]) # Матрица (левая часть системы)  
 V1 = np.array([SY, SXY, SXXY, SXXXY]) # Вектор (правая часть системы)  
 a0, a1, a2, a3 = np.linalg.solve(M1, V1)  
 self.\_setPX(self.\_calculatePolynomialThirdValues(a0, a1, a2, a3))  
 self.\_setEpsilon(self.\_calculateEpsilon())  
 S = self.\_deviationMeasure()  
 SD = self.\_standardDeviation()  
 return {'SX': SX, 'SXX': SXX, 'SXXX': SXXX, 'SXXXX': SXXXX, 'SXXXXX': SXXXXX, 'SXXXXXX': SXXXXXX,  
 'SY': SY, 'SXY': SXY, 'SXXY': SXXY, 'SXXXY': SXXXY, 'N': N,  
 'a0': round(a0, 5), 'a1': round(a1, 5), 'a2': round(a2, 5), 'a3': round(a3, 5), 'S': S, 'SD': SD}  
  
 def polynomialThirdFunction(self):  
 print('\t\t\tПолиномиальная функция 3-ей степени')  
 coefficients = self.\_\_calculatePolynomialThirdFunctionCoefficient()  
 function = f"P3(x) = {coefficients['a0']} + ({coefficients['a1']}x)+ ({coefficients['a2']}x^2) + ({coefficients['a3']}x^3)"  
 print(f"\tПолучим систему уравнений:\n"  
 f"\t\t{coefficients['N']}a0 + {coefficients['SX']}a1 + {coefficients['SXX']}a2 + {coefficients['SXXX']}a3 = {coefficients['SY']}\n"  
 f"\t\t{coefficients['SX']}a0 + {coefficients['SXX']}a1 + {coefficients['SXXX']}a2 + {coefficients['SXXXX']}a3= {coefficients['SXY']}\n"  
 f"\t\t{coefficients['SXX']}a0 + {coefficients['SXXX']}a1 + {coefficients['SXXXX']}a2 + {coefficients['SXXXXX']}a3= {coefficients['SXXY']}\n"  
 f"\t\t{coefficients['SXXX']}a0 + {coefficients['SXXXX']}a1 + {coefficients['SXXXXX']}a2 + {coefficients['SXXXXXX']}a3= {coefficients['SXXXY']}\n")  
 print(f"\tНайденные коэффициенты:\n"  
 f"\t\ta0 = {coefficients['a0']}\n"  
 f"\t\ta1 = {coefficients['a1']}\n"  
 f"\t\ta2 = {coefficients['a2']}\n"  
 f"\t\ta3 = {coefficients['a3']}\n")  
 self.\_\_printCoefficients(coefficients, function)  
 return function, {'a0': coefficients['a0'], 'a1': coefficients['a1'], 'a2': coefficients['a2'],  
 'a3': coefficients['a3']}, \  
 coefficients['S'], coefficients['SD']  
  
 # Экспоненциальная  
 def \_\_calculateExponentFunctionCoefficient(self):  
 y\_arr = self.getArrayY()  
 self.setArrayY([log(y) for y in y\_arr])  
 A, B, a, b = self.calculateLinearForOtherFunctions()  
 self.setArrayY(y\_arr)  
 self.\_setPX(self.\_calculateExponentValues(a, b))  
 self.\_setEpsilon(self.\_calculateEpsilon())  
 S = self.\_deviationMeasure()  
 SD = self.\_standardDeviation()  
 return {'A': A, 'B': B, 'a': a,  
 'b': b, 'S': S, 'SD': SD}  
  
 def exponentFunction(self):  
 print('\t\t\tЭкспоненциальная функция')  
 coefficients = self.\_\_calculateExponentFunctionCoefficient()  
 function = f"P4(x) = {coefficients['a']}\*e({coefficients['b']}\*x)"  
 self.\_\_printCoefficients(coefficients, function)  
 return function, {'a': coefficients['a'], 'b': coefficients['b']}, \  
 coefficients['S'], coefficients['SD']  
  
 # Cтепенная  
 def \_\_calculateExponentialFunctionCoefficient(self):  
 y\_arr, x\_arr = self.getArrayY(), self.getArrayX()  
 self.setArrayY([log(y) for y in y\_arr])  
 self.setArrayX([log(x) for x in x\_arr])  
 A, B, a, b = self.calculateLinearForOtherFunctions()  
 self.setArrayY(y\_arr)  
 self.setArrayX(x\_arr)  
 self.\_setPX(self.\_calculateExponentialValues(a, b))  
 self.\_setEpsilon(self.\_calculateEpsilon())  
 S = self.\_deviationMeasure()  
 SD = self.\_standardDeviation()  
 return {'A': A, 'B': B, 'a': a,  
 'b': b, 'S': S, 'SD': SD}  
  
 def exponentialFunction(self):  
 print('\t\t\tСтепенная функция')  
 coefficients = self.\_\_calculateExponentialFunctionCoefficient()  
 function = f"P5(x) = {coefficients['a']}\*x^({coefficients['b']})"  
 self.\_\_printCoefficients(coefficients, function)  
 return function, {'a': coefficients['a'], 'b': coefficients['b']}, \  
 coefficients['S'], coefficients['SD']  
  
 # Логарифмическая  
 def \_\_calculateLogarithmFunctionCoefficient(self):  
 x\_arr = self.getArrayX()  
 self.setArrayX([log(x) for x in x\_arr])  
 coefficient = self.\_\_calculateLinearFunctionCoefficient()  
 A, B = coefficient['a'], coefficient['b']  
 a, b = A, B  
 self.setArrayX(x\_arr)  
 self.\_setPX(self.\_calculateLogarithmValues(a, b))  
 self.\_setEpsilon(self.\_calculateEpsilon())  
 S = self.\_deviationMeasure()  
 SD = self.\_standardDeviation()  
 return {'A': A, 'B': B, 'a': a,  
 'b': b, 'S': S, 'SD': SD}  
  
 def logarithmFunction(self):  
 print('\t\t\tЛогарифмическая функция')  
 coefficients = self.\_\_calculateLogarithmFunctionCoefficient()  
 function = f"P6(x) = {coefficients['a']}\*ln(x)+{coefficients['b']}"  
 self.\_\_printCoefficients(coefficients, function)  
 return function, {'a': coefficients['a'], 'b': coefficients['b']}, \  
 coefficients['S'], coefficients['SD']  
  
 def \_\_printCoefficients(self, coefficients, function):  
 for key, val in coefficients.items():  
 print(f" {key} = {val}.")  
 print(f"Аппроксимирующая функция имеет вид {function}")  
 self.\_printTable()  
 print(f"Мера отклонения S = {coefficients['S']}")  
 print(f"Среднеквадратическое отклонение SD = {coefficients['SD']}")  
  
 def calculateLinearForOtherFunctions(self):  
 coefficient = self.\_\_calculateLinearFunctionCoefficient()  
 A, B = coefficient['b'], coefficient['a']  
 a, b = math.exp(A), B  
 return A, B, a, b

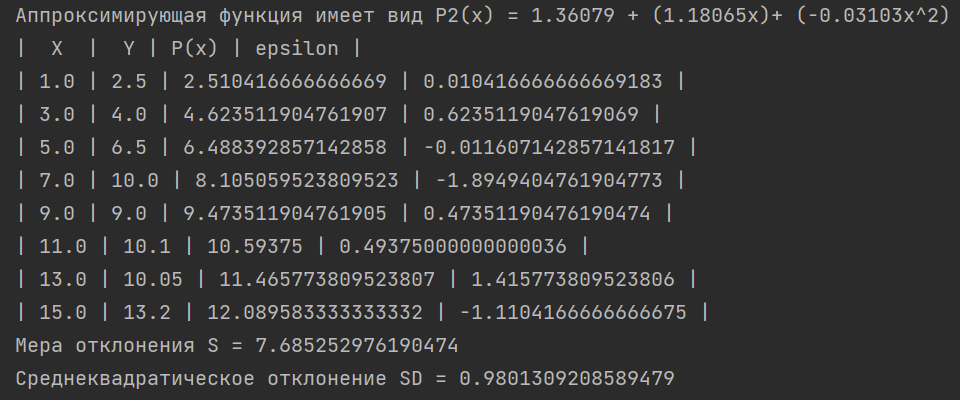
***5. Графики:***

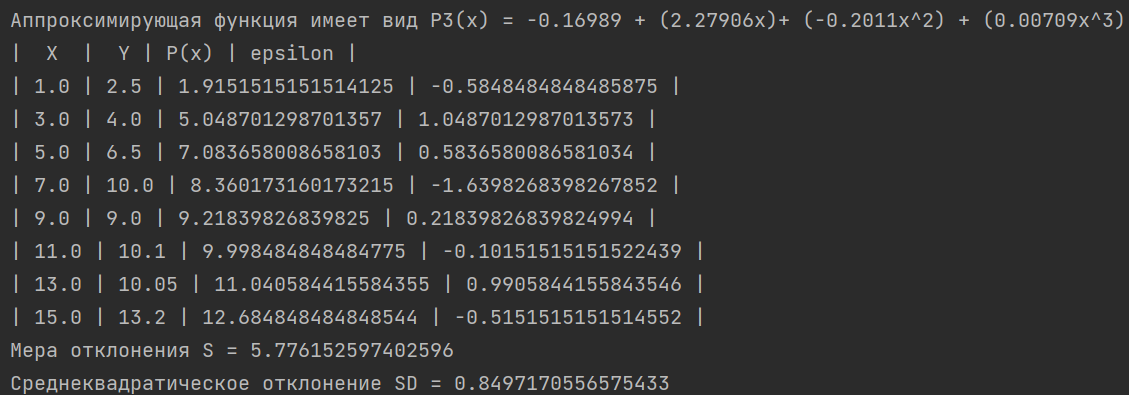
******

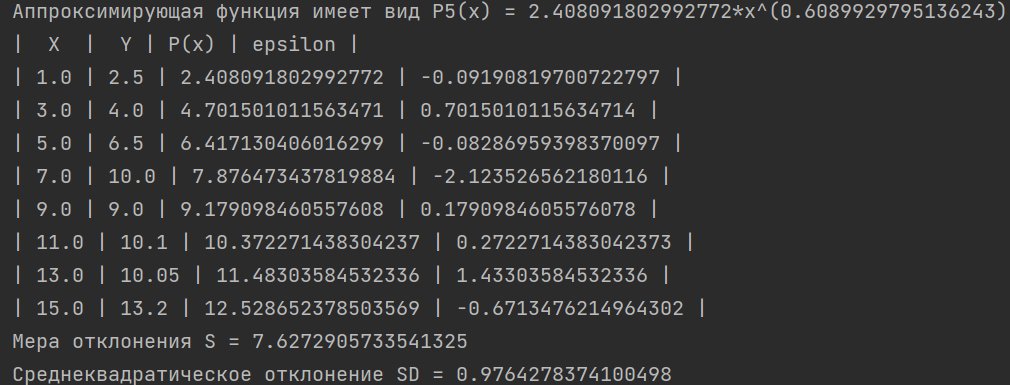
******

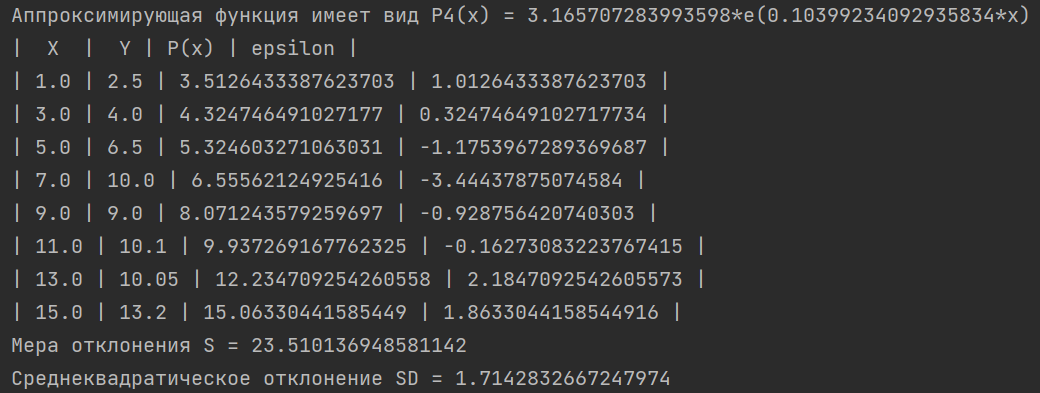
***6. Результаты выполнения программы:***

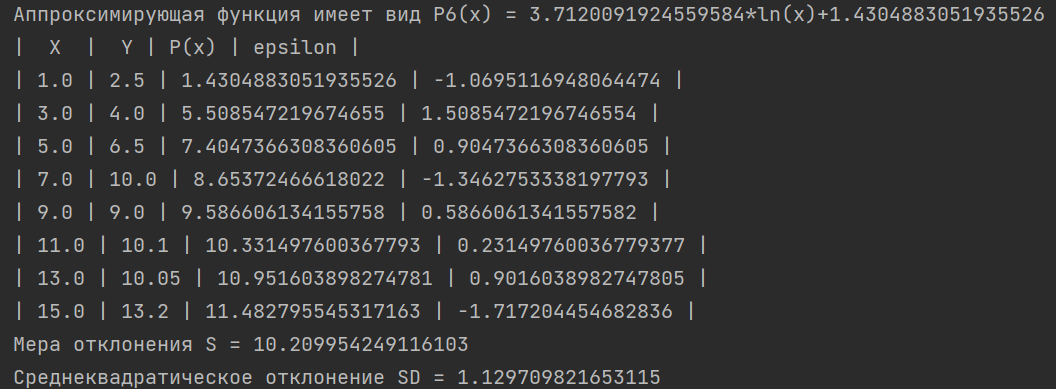
******

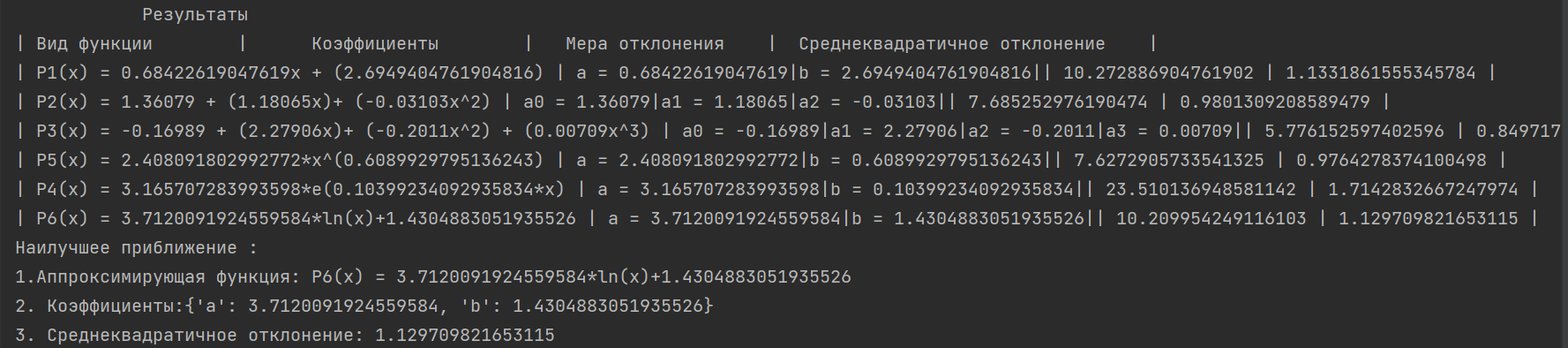
******

******

******

******

******

******

***7. Выводы***:

Я научилась находить функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.