Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №4**

по «Вычислительной математике»

Аппроксимация функции методом наименьших квадратов

Вариант 4

Выполнил:

Студент группы P3213

Ваховский П.А.

Преподаватель:

Малышева Т.А.

Санкт-Петербург

2022

**Цель лабораторной работы**: Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

***Вычислительная реализация***

Таблица значений:

|  |  |
| --- | --- |
| X | Y |
| -2 | -0,588 |
| -1.8 | -0,783 |
| -1.6 | -1,059 |
| -1.4 | -1,446 |
| -1.2 | -1,952 |
| -1.0 | -2,500 |
| -0.8 | -2,838 |
| -0.6 | -2,656 |
| -0.4 | -1,950 |
| -0.2 | -0,998 |
| 0 | 0 |

**Линейная аппроксимация**

**-**11

Чтобы определить коэффициенты аппроксимирующей функции



Решим систему:

Изображение выглядит как текст

Автоматически созданное описание

Полученные результаты:

a = -0,177

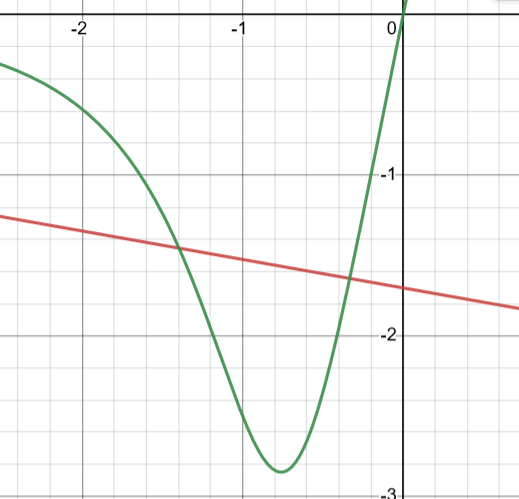
b = -1,702

**Полученная функция аппроксимации: -0,17727x-1,70181**

|  |  |  |
| --- | --- | --- |
| X | Y | eps |
| -2 | -1,347 | 0,759 |
| -1.8 | -1,383 | 0,600 |
| -1.6 | -1,418 | 0,359 |
| -1.4 | -1,454 | 0,008 |
| -1.2 | -1,489 | 0,463 |
| -1.0 | -1,525 | 0,975 |
| -0.8 | -1,560 | 1,278 |
| -0.6 | -1,595 | 1,060 |
| -0.4 | -1,631 | 0,319 |
| -0.2 | -1,666 | 0,668 |
| 0 | -1,702 | 1,702 |

Отклонение: 0.8755

График



**Квадратичная аппроксимация**

**-**11

Решим следующую систему:

Изображение выглядит как текст, человек

Автоматически созданное описание

Полученные результаты:

a0 = 2,22

a1 = 4,264

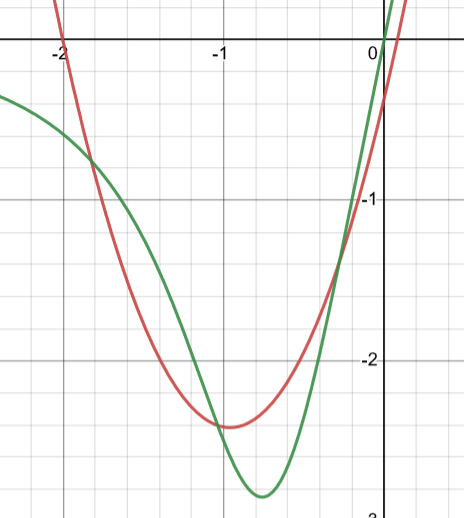
a2 = -0,369

**Полученная функция аппроксимации: 2,220x^2+4,264x-0,369**

|  |  |  |
| --- | --- | --- |
| X | Y | eps |
| -2 | -0,015 | 0,573 |
| -1.8 | -0,850 | 0,067 |
| -1.6 | -1,507 | 0,448 |
| -1.4 | -1,987 | 0,541 |
| -1.2 | -2,289 | 0,336 |
| -1.0 | -2,413 | 0,087 |
| -0.8 | -2,359 | 0,478 |
| -0.6 | -2,128 | 0,527 |
| -0.4 | -1,720 | 0,230 |
| -0.2 | -1,133 | 0,135 |
| 0 | -0,369 | 0,369 |

Отклонение: 0.389

График



Листинг класса Approximation:

import java.util.\*;  
import java.util.stream.Collectors;  
  
import org.apache.commons.math3.linear.\*;  
  
public class Approximation {  
 FunctionType type;  
 public double sd = 1000;  
 public static Approximation *bestApp*;  
 int power;  
 Function appFunction;  
 public List<Double> X, Y;  
 private double coefficients[][], constants[], solution[];  
 String wolfRequest;  
 public Approximation(FunctionType type, List<Double> X, List<Double> Y){  
 this.type = type;  
  
 switch (type){  
 case *Square*:  
 this.power = 2;  
 break;  
 case *Cube*:  
 this.power = 3;  
 break;  
 default:  
 this.power = 1;  
 }  
  
 this.X = new ArrayList<>(X);  
 this.Y = new ArrayList<>(Y);  
  
 this.coefficients = new double[power+1][power+1];  
 this.constants = new double[power+1];  
 }  
  
  
  
 public void formLinearSystem(){  
 List<Double> Xt = new ArrayList<>(X);  
 List<Double> Yt = new ArrayList<>(Y);  
  
 switch (type){  
 case *Exp*:  
 Yt = Yt.stream().mapToDouble(Math::*log*).boxed().collect(Collectors.*toList*());  
 break;  
 case *Log*:  
 Xt = Xt.stream().mapToDouble(Math::*log*).boxed().collect(Collectors.*toList*());  
 break;  
 case *Power*:  
 Xt = Xt.stream().mapToDouble(Math::*log*).boxed().collect(Collectors.*toList*());  
 Yt = Yt.stream().mapToDouble(Math::*log*).boxed().collect(Collectors.*toList*());  
 }  
// for(Double x : Xt){  
// System.out.println(x);  
// }  
// for(Double y : Yt){  
// System.out.println(y);  
// }  
 for(int i = 0; i <= power; i++){  
 int p = power\*2-i;  
 for(int j = 0; j <= power; j++){  
 int finalP = p - j;  
 coefficients[i][j] = Xt.stream().mapToDouble(x->Math.*pow*(x, finalP)).sum();  
 System.*out*.printf("%.3f ", coefficients[i][j]);  
 }  
 for(int k = 0; k < Xt.size(); k++){  
 constants[i] += Math.*pow*(Xt.get(k), power-i)\*Yt.get(k);  
 }  
 System.*out*.printf("\t %.3f\n", constants[i]);  
 }  
 System.*out*.println();  
 }  
  
 public void solveLinearSystem() {  
 DecompositionSolver solver = new LUDecomposition(new Array2DRowRealMatrix(coefficients)).getSolver();  
 solution = solver.solve(new ArrayRealVector(constants)).toArray();  
 }  
  
 public void setAppFunc(){  
 switch (type){  
 case *Exp*:  
 appFunction = x -> Math.*exp*(solution[1])\*Math.*exp*(solution[0]\*x);  
 System.*out*.printf("%.5f %.5f\n", Math.*exp*(solution[1]), solution[0]);  
 wolfRequest = String.*format*("%.5f", Math.*exp*(solution[1])) + "e^(+" + String.*format*("%.5f", solution[0]) + "x)";  
 break;  
 case *Log*:  
 appFunction = x -> solution[0]\*Math.*log*(x)+solution[1];  
 System.*out*.printf("%.5f %.5f\n", solution[0], solution[1]);  
 wolfRequest = String.*format*("%.5f", solution[0]) + "ln(x)+" + String.*format*("%.5f", solution[1]);  
 break;  
 case *Power*:  
 appFunction = x -> Math.*exp*(solution[0])\*Math.*pow*(x, solution[1]);  
 System.*out*.printf("%.5f %.5f\n", solution[0], solution[1]);  
 wolfRequest = String.*format*("%.5f", Math.*exp*(solution[0])) + "x^(" + String.*format*("%.5f", solution[1]) + ")";  
 break;  
 case *Linear*:  
 appFunction = x -> solution[0]\*x+solution[1];  
 System.*out*.printf("%.5f %.5f\n", solution[0], solution[1]);  
 wolfRequest = String.*format*("%.5f", solution[0]) + "x+" + String.*format*("%.5f", solution[1]);  
 break;  
 case *Square*:  
 appFunction = x -> solution[0]\*x\*x+solution[1]\*x+solution[2];  
 System.*out*.printf("%.5f %.5f %.5f\n", solution[0], solution[1], solution[2]);  
 wolfRequest = String.*format*("%.5f", solution[0])+"x^2+" + String.*format*("%.5f", solution[1])+"x+"+String.*format*("%.5f", solution[2]);  
 break;  
 case *Cube*:  
 appFunction = x -> solution[0]\*x\*x\*x+solution[1]\*x\*x+solution[2]\*x+solution[3];  
 System.*out*.printf("%.5f %.5f %.5f %.5f\n", solution[0], solution[1], solution[2], solution[3]);  
 wolfRequest = String.*format*("%.5f", solution[0]) +"x^3+" + String.*format*("%.5f", solution[1]) +"x^2+" + String.*format*("%.5f", solution[2]) +"x+" +String.*format*("%.5f", solution[3]);  
 }  
 }  
 public void setStandartDeviation(){  
 double s2 = 0;  
 for(int i = 0; i < X.size(); i++){  
 s2 += Math.*pow*(appFunction.calcFunction(X.get(i))-Y.get(i), 2);  
 }  
 sd = Math.*sqrt*(s2/X.size());  
 if(*bestApp* == null || sd < *bestApp*.sd){  
 *bestApp* = this;  
 }  
 }  
  
}

**input.txt:**

Изображение выглядит как текст

Автоматически созданное описание

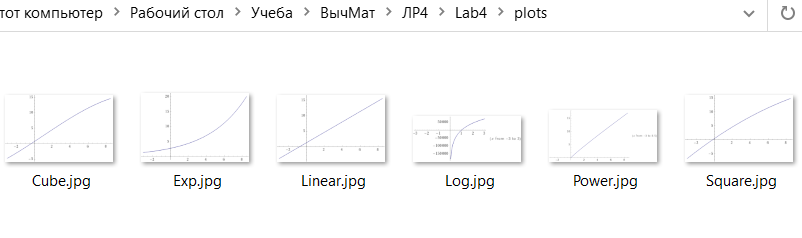
**Вывод программы:**

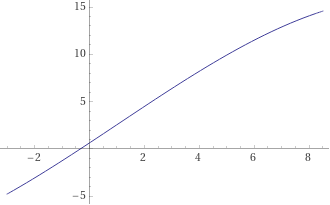
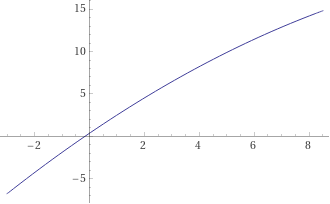
Изображение выглядит как текст

Автоматически созданное описание

**Графики:**

Изображение выглядит как текст

Автоматически созданное описание 

**Вывод**: выполняя эту ЛР я научился аппроксировать функцию через метод наименьших квадратов, используя шесть разных видов апр. функций.