БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №3

**Метод наименьших квадратов**

**Выполнил:**

Крючков Василий

2 курс 9 группа

**Преподаватель:**

Горбачева Ю.Н.

Минск, 2022

**Постановка задачи**

На отрезкезадана таблица значений функции с шагом . По заданной таблице значений найти наилучшие среднеквадратичные приближения при .

Найти .

**Краткие теоретические сведения**

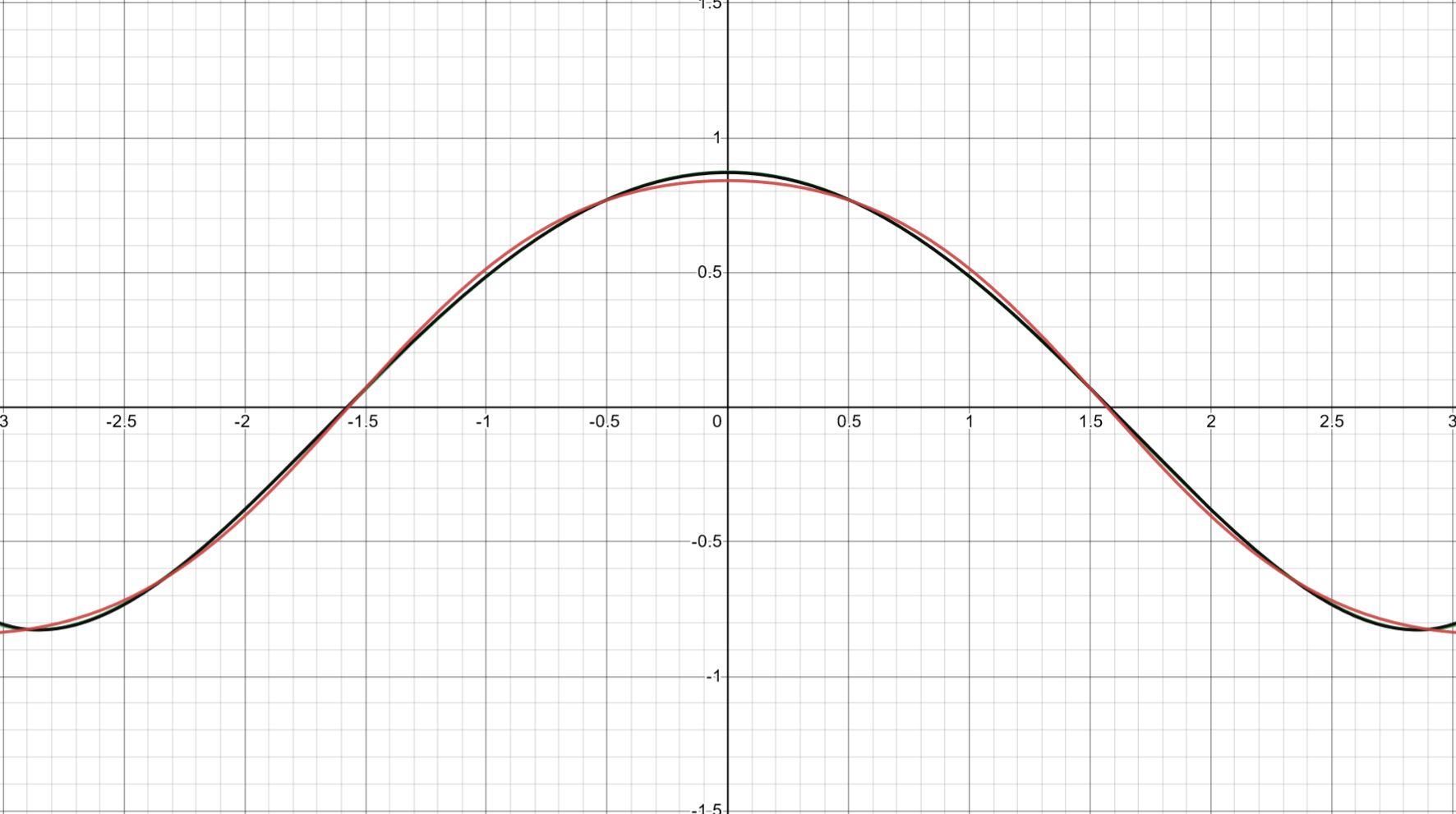
Приближение методом неменьших квадратов алгебраическими многочленами:

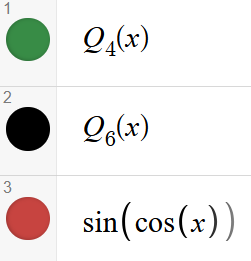
**Листинг программы**

**Lab.java**

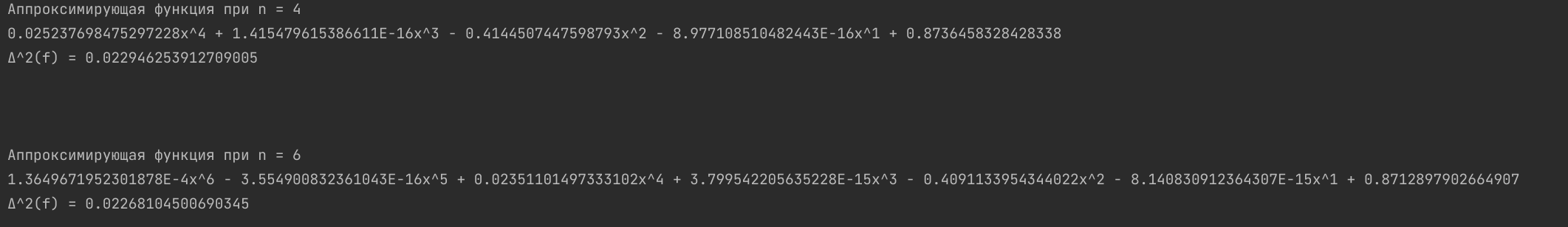
import static java.lang.Math.*abs*;  
  
public class Lab {  
 public static void main(String[] args) {  
 Program pr = new Program();  
 try {  
 pr.base();  
 } catch (Error e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}  
  
class Program {  
 void base() {  
 double a = -3;  
 double b = 3;  
 double h = 0.1;  
 methodOfLeastSquares(4, a, b, h);  
 System.*out*.println("");  
 System.*out*.println("");  
 System.*out*.println("");  
 methodOfLeastSquares(6, a, b, h);  
 }  
  
 private void methodOfLeastSquares(int n, double a, double b, double h) {  
 int fn = (int) (*abs*(b - a) / h) + 1;  
 double[] xk = new double[fn];  
 double[] fk = new double[fn];  
 double t = a;  
 for (int i = 0; i < fn; i++) {  
 xk[i] = t;  
 fk[i] = f(t);  
 t += h;  
 }  
 double[][] g = new double[n + 1][n + 1];  
 double[] m = new double[n + 1];  
 for (int i = 0; i < n + 1; i++) {  
 double sum = 0.0;  
 for (int j = 0; j < fn; j++) {  
 sum += Math.*pow*(xk[j], i) \* fk[j];  
 }  
 m[i] = sum;  
 }  
 double[] sn = new double[2 \* n + 1];  
 for (int i = 0; i < 2 \* n + 1; i++) {  
 double sum = 0.0;  
 for (int j = 0; j < fn; j++) {  
 sum += Math.*pow*(xk[j], i);  
 }  
 sn[i] = sum;  
 }  
 for (int i = 0; i < n + 1; i++) {  
 for (int j = i; j < n + 1; j++) {  
 if (i == j)  
 g[i][j] = sn[2 \* i];  
 else {  
 g[i][j] = sn[i + j];  
 g[j][i] = sn[i + j];  
 }  
 }  
 }  
 double[] c = *gaussSelectionByColumn*(n + 1, g, m);  
 System.*out*.println("Аппроксимирующая функция при n = " + n);  
 System.*out*.printf(c[n] + "x^%d", n);  
 for (int i = n - 1; i != 0; i--) {  
 if (c[i] >= 0.0 || c[i] == -0.0)  
 System.*out*.printf(" + " + Math.*abs*(c[i]) + "x^%d", i);  
 else  
 System.*out*.printf(" - " + Math.*abs*(c[i]) + "x^%d", i);  
 }  
 if (c[0] >= 0)  
 System.*out*.println(" + " + Math.*abs*(c[0]));  
 else  
 System.*out*.println(" - " + Math.*abs*(c[0]));  
 double del = 0.0;  
 for (int i = 0; i < fn; i++) {  
 del += Math.*pow*(fk[i] - qnf(xk[i], c, n), 2);  
 }  
 System.*out*.println("Δ^2(f) = " + del);  
 }  
  
  
 private double f(double x) {  
 return Math.*sin*(Math.*cos*(x));  
 }  
  
 private double qnf(double x, double[] c, int n) {  
 double t = 0.0;  
 for (int i = n; i > -1; i--) {  
 t += Math.*pow*(x, i) \* c[i];  
 }  
 return t;  
 }  
  
 public static double[] gaussSelectionByColumn(int matrixOrder, double[][] matrixA, double[] columnF) {  
 double[] columnX = new double[matrixOrder];  
 int maxIndex;  
 double maxInColumn;  
 double temp;  
 for (int i = 0; i < matrixOrder - 1; i++) {  
 maxIndex = i;  
 maxInColumn = Math.*abs*(matrixA[i][i]);  
 for (int j = i; j < matrixOrder; j++) {  
 if (maxInColumn < Math.*abs*(matrixA[j][i])) {  
 maxInColumn = Math.*abs*(matrixA[j][i]);  
 maxIndex = j;  
 }  
 if (maxInColumn == 0)  
 throw new ArithmeticException("Нулевой столбец - решение данной слау невозможно");  
 }  
 for (int k = i; k < matrixOrder; k++) {  
 temp = matrixA[i][k];  
 matrixA[i][k] = matrixA[maxIndex][k];  
 matrixA[maxIndex][k] = temp;  
 }  
 temp = columnF[i];  
 columnF[i] = columnF[maxIndex];  
 columnF[maxIndex] = temp;  
 for (int j = i + 1; j < matrixOrder; j++)  
 matrixA[i][j] = matrixA[i][j] / matrixA[i][i];  
 columnF[i] = columnF[i] / matrixA[i][i];  
 matrixA[i][i] = 1.0;  
 for (int j = i + 1; j < matrixOrder; j++) {  
 for (int k = i + 1; k < matrixOrder; k++)  
 matrixA[j][k] = matrixA[j][k] + matrixA[i][k] \* matrixA[j][i] \* (-1.0);  
 columnF[j] = columnF[j] + columnF[i] \* matrixA[j][i] \* (-1.0);  
 matrixA[j][i] = 0;  
 }  
 }  
 columnF[matrixOrder - 1] = columnF[matrixOrder - 1] / matrixA[matrixOrder - 1][matrixOrder - 1];  
 matrixA[matrixOrder - 1][matrixOrder - 1] = 1;  
 columnX[matrixOrder - 1] = columnF[matrixOrder - 1];  
 for (int i = matrixOrder - 2; i > -1; i--) {  
 columnX[i] = columnF[i];  
 for (int k = i; k < matrixOrder - 1; k++)  
 columnX[i] = columnX[i] + matrixA[i][k + 1] \* columnX[k + 1] \* (-1.0);  
 }  
 return columnX;  
 }  
}

**Результаты**





**Результаты вычисления**

****