Университет ИТМО, факультет ПИиКТ

Лабораторная работа №4

Дисциплина: Вычислительная математика

Вариант 19

Выполнил: Щелыкалов Виктор

Группа: Р3214

Преподаватель:

Малышева Татьяна Алексеевна

Санкт-Петербург, 2020 год

**Цель работы:** найти наилучшую аппроксимирующую функцию, используя метод наименьших квадратов.

**Текст задания:**

**Исходные данные:**

* линейная функция;
* полиномиальная функция 2-й степени;
* экспоненциальная функция;
* логарифмическая функция;
* степенная функция.

**Программная реализация задачи:**

* Предусмотреть ввод исходных данных из файла.
* Реализовать метод наименьших квадратов, исследуя все функции.
* Предусмотреть вывод результатов в файл.
* Программа должна выбирать наилучшую аппроксимирующую функцию.
* Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом).

**Ход работы:**

**Линейная аппроксимация:**

*Идея метода:* рассмотрим в качестве эмпирической формулы линейную функцию:

Сумма квадратов отклонений запишется следующим образом:

Для нахождения *а* и *b* необходимо найти минимум функции *S(a,b).* Необходимое условие существования минимума для функции *S*:

Упростим полученную систему:

Введем обозначения:

Получим систему уравнений для нахождения параметров и :

**Квадратичная аппроксимация:**

*Идея метода:* рассмотрим в качестве эмпирической формулы квадратичную функцию:

Сумма квадратов отклонений запишется следующим образом:

Приравниваем к нулю частные производные *S* по неизвестным параметрам, получаем систему линейных уравнений:

Введем обозначения:

Получим систему уравнений для нахождения параметров и , решая которую (в коде использовал библиотеку Apache Math) получаем необходимые параметры.

**Аппроксимация с помощью других функций:**

*Идея метода:* помимо линейных зависимостей для описания результатов эксперимента используют также показательные, степенные, логарифмические функции. Эти функции легко могут быть приведены к линейному виду, после чего для определения коэффициентов аппроксимирующей функции можно использовать описанный выше алгоритм.

**Аппроксимирующая функция задана степенной функцией вида:**

Для применения метода наименьших квадратов степенная функция линеаризуется:

Введем обозначения:

Получаем линейную зависимость: . После определения коэффициентов и вернемся к принятым ранее обозначениям: и .

**Аппроксимирующая функция задана экспоненциальной функцией вида:**

Для применения метода наименьших квадратов экспоненциальная функция линеаризуется:

Введем обозначения:

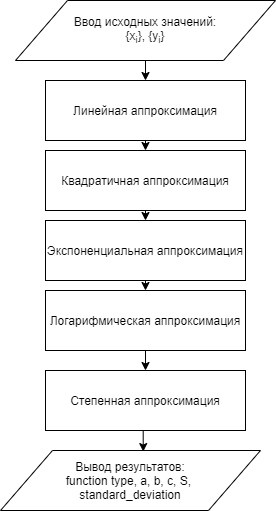
Получаем линейную зависимость: . После определения коэффициентов и вернемся к принятым ранее обозначениям: и .

**Аппроксимирующая функция задана логарифмической функцией вида:**

Введем обозначения:

Получаем линейную зависимость: . Откуда находим коэффициенты *a* и *b*.

*Соответствующая блок-схема:*



**Листинг:**

Main.java

import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.chart.LineChart;  
import javafx.scene.chart.NumberAxis;  
import javafx.scene.chart.XYChart;  
import javafx.stage.Stage;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.util.Scanner;  
  
public class Main extends Application {  
  
 public static void main(String[] args) {  
 launch(args);  
 }  
  
 @Override  
 public void start(Stage stage) throws Exception {  
 Scanner scanner = new Scanner(new File("input.txt"));  
 double[] xArr = new double[50];  
 double[] yArr = new double[50];  
 double[] xyArr = new double[100];  
 int n = 0;  
  
 while (scanner.hasNextDouble()) {  
 xyArr[n] = scanner.nextDouble();  
 n++;  
 }  
 for (int i = 0; i < n / 2; i++) { //собираем значения из файла  
 xArr[i] = xyArr[i];  
 yArr[i] = xyArr[i + n / 2];  
  
 }  
 String minName = "Линейная";  
  
 FileWriter writer = new FileWriter("output.txt", false);  
 // запись всей строки  
 String text = "Вид функции | a | b | c | S | Sigma |";  
 writer.write(text);  
 writer.append('\n');  
  
 double [] ansLine = Solver.linearApprox(xArr, yArr, n / 2);  
 double min = ansLine[3];  
 text = "f = a\*x + b | " + String.format("%.3f", ansLine[0]) + " | " + String.format("%.3f", ansLine[1])  
 + " | - | " + String.format("%.3f", ansLine[2])  
 + " | " + String.format("%.3f", ansLine[3]) + " |";  
 writer.write(text);  
 writer.append('\n');  
 writer.flush();  
  
 double[] ansQuad = Solver.quadraticApprox(xArr, yArr, n / 2);  
 text = "f = a\*x^2 + b\*x + c | " + String.format("%.3f", ansQuad[0]) + " | " + String.format("%.3f", ansQuad[1])  
 + " | " + String.format("%.3f", ansQuad[2]) +" | " + String.format("%.3f", ansQuad[3])  
 + " | " + String.format("%.3f", ansQuad[4]) + " |";  
 writer.write(text);  
 writer.append('\n');  
 writer.flush();  
 if(min > ansQuad[4]) {  
 min = ansQuad[4];  
 minName = "Квадратичная";  
 }  
  
 double[] ansPow = Solver.powApprox(xArr, yArr, n / 2);  
 text = "f = a\*x^b | " + String.format("%.3f", ansPow[0]) + " | " + String.format("%.3f", ansPow[1])  
 + " | - | " + String.format("%.3f", ansPow[2])  
 + " | " + String.format("%.3f", ansPow[3]) + " |";  
 writer.write(text);  
 writer.append('\n');  
 writer.flush();  
 if(min > ansPow[3]) {  
 min = ansPow[3];  
 minName = "Степенная";  
 }  
  
 double[] ansExp = Solver.expApprox(xArr, yArr, n / 2);  
 text = "f = a\*e^(bx) | " + String.format("%.3f", ansExp[0]) + " | " + String.format("%.3f", ansExp[1])  
 + " | - | " + String.format("%.3f", ansExp[2])  
 + " | " + String.format("%.3f", ansExp[3]) + " |";  
 writer.write(text);  
 writer.append('\n');  
 writer.flush();  
 if(min > ansExp[3]) {  
 min = ansExp[3];  
 minName = "Эксопоненциальная";  
 }  
  
 double [] ansLog = Solver.logApprox(xArr, yArr, n / 2);  
 text = "f = a\*lnx + b | " + String.format("%.3f", ansLog[0]) + " | " + String.format("%.3f", ansLog[1])  
 + " | - | " + String.format("%.3f", ansLog[2])  
 + " | " + String.format("%.3f", ansLog[3]) + " |";  
 writer.write(text);  
 writer.append('\n');  
 writer.flush();  
 if(min > ansExp[3]) {  
 min = ansExp[3];  
 minName = "Логарифмическая";  
 }  
 writer.write("Наилучшая аппроксимирующая функция - " + minName + "!");  
 writer.append('\n');  
 writer.flush();  
 writer.close();  
  
  
  
 //вывод графика  
 stage.setTitle("График");  
  
 final NumberAxis xAxis = new NumberAxis();  
 final NumberAxis yAxis = new NumberAxis();  
 xAxis.setLabel("X");  
 yAxis.setLabel("Y");  
  
 final LineChart<Number, Number> lineChart =  
 new LineChart<Number, Number>(xAxis, yAxis);  
  
 XYChart.Series seriesLine = new XYChart.Series();  
 XYChart.Series seriesQuad = new XYChart.Series();  
 XYChart.Series seriesPow = new XYChart.Series();  
 XYChart.Series seriesExp = new XYChart.Series();  
 XYChart.Series seriesLog = new XYChart.Series();  
 XYChart.Series seriesInput = new XYChart.Series();  
 seriesLine.setName("Линейный");  
 seriesQuad.setName("Квадратичный");  
 seriesPow.setName("Степенной");  
 seriesExp.setName("Экспонента");  
 seriesLog.setName("Логарифмический");  
 seriesInput.setName("Искомый");  
  
  
 for (int i = 0; i < n/2; i++) {  
 seriesLine.getData().add(new XYChart.Data(xArr[i], ansLine[0]\*xArr[i] +ansLine[1]));  
 seriesQuad.getData().add(new XYChart.Data(xArr[i], ansQuad[0]\*xArr[i]\*xArr[i] +ansLine[1]\*xArr[i] + ansLine[2]));  
 seriesPow.getData().add(new XYChart.Data(xArr[i], Math.pow((ansPow[0] \* Math.pow(xArr[i], ansPow[1]) - yArr[i]), 2)));  
 seriesExp.getData().add(new XYChart.Data(xArr[i], ansLine[0]\*Math.pow(Math.E, xArr[i]\*ansLine[1])));  
 seriesLog.getData().add(new XYChart.Data(xArr[i], ansLine[0]\*Math.log(xArr[i]) +ansLine[1]));  
 seriesInput.getData().add(new XYChart.Data(xArr[i], yArr[i]));  
 }  
  
 Scene scene = new Scene(lineChart, 1300, 1300);  
 lineChart.getData().add(seriesLine);  
 lineChart.getData().add(seriesQuad);  
 lineChart.getData().add(seriesPow);  
 lineChart.getData().add(seriesExp);  
 lineChart.getData().add(seriesLog);  
 lineChart.getData().add(seriesInput);  
  
 stage.setScene(scene);  
 stage.show();  
  
 }  
  
}

Solver.java

import org.apache.commons.math3.linear.Array2DRowRealMatrix;  
import org.apache.commons.math3.linear.ArrayRealVector;  
import org.apache.commons.math3.linear.DecompositionSolver;  
import org.apache.commons.math3.linear.LUDecomposition;  
import org.apache.commons.math3.linear.RealMatrix;  
import org.apache.commons.math3.linear.RealVector;  
  
public class Solver {  
  
  
 public static double getSX(double[] arr, int n) {  
 double s = 0;  
 for (int i = 0; i < n; i++) {  
 s += arr[i];  
 }  
 return s;  
 }  
  
 public static double getSXX(double[] xArr, int n) {  
 double sxx = 0;  
 for (int i = 0; i < n; i++) {  
 sxx += Math.pow(xArr[i], 2);  
 }  
 return sxx;  
 }  
  
 public static double getSXXX(double[] xArr, int n) {  
 double sxxx = 0;  
 for (int i = 0; i < n; i++) {  
 sxxx += Math.pow(xArr[i], 3);  
 }  
 return sxxx;  
 }  
  
 public static double getSXXXX(double[] xArr, int n) {  
 double sxxxx = 0;  
 for (int i = 0; i < n; i++) {  
 sxxxx += Math.pow(xArr[i], 4);  
 }  
 return sxxxx;  
 }  
  
 public static double getSXY(double[] xArr, double[] yArr, int n) {  
 double sxy = 0;  
 for (int i = 0; i < n; i++) {  
 sxy += xArr[i] \* yArr[i];  
 }  
 return sxy;  
 }  
  
 public static double getSXXY(double[] xArr, double[] yArr, int n) {  
 double sxxy = 0;  
 for (int i = 0; i < n; i++) {  
 sxxy += Math.pow(xArr[i], 2) \* yArr[i];  
 }  
 return sxxy;  
 }  
  
 public static double[] linearApprox(double[] xArr, double[] yArr, int n) {  
 double a, b, s = 0, cigma;  
 a = (getSXY(xArr, yArr, n) \* n - getSX(xArr, n) \* getSX(yArr, n)) / (getSXX(xArr, n) \* n - Math.pow(getSX(xArr, n), 2));  
 b = (getSXX(xArr, n) \* getSX(yArr, n) - getSX(xArr, n) \* getSXY(xArr, yArr, n)) / (getSXX(xArr, n) \* n - Math.pow(getSX(xArr, n), 2));  
 for (int i = 0; i < n; i++) {  
 s += Math.pow((a \* xArr[i] + b - yArr[i]), 2);  
 }  
 cigma = Math.sqrt(s/n);  
 return new double[]{a, b, s, cigma};  
 }  
  
 public static double[] quadraticApprox(double[] xArr, double[] yArr, int n) {  
 double a, b, c, s = 0, cigma;  
 RealMatrix coefficients =  
 new Array2DRowRealMatrix(new double[][]{  
 {n, getSX(xArr, n), getSXX(xArr, n)},  
 {getSX(xArr, n), getSXX(xArr, n), getSXXX(xArr, n)},  
 {getSXX(xArr, n), getSXXX(xArr, n), getSXXXX(xArr, n)}  
 },  
 false);  
 DecompositionSolver solver = new LUDecomposition(coefficients).getSolver();  
 RealVector constants = new ArrayRealVector(new double[]{getSX(yArr, n), getSXY(xArr, yArr, n), getSXXY(xArr, yArr, n)}, false);  
 RealVector solution = solver.solve(constants);  
 a = solution.getEntry(2);  
 b = solution.getEntry(1);  
 c = solution.getEntry(0);  
 for (int i = 0; i < n; i++) {  
 s += Math.pow((a \* xArr[i] \* xArr[i] + b \* xArr[i] + c - yArr[i]), 2);  
 }  
 cigma = Math.sqrt(s/n);  
 return new double[]{a, b, c, s, cigma};  
 }  
  
  
 public static double[] powApprox(double[] xArr, double[] yArr, int n) {  
 double a, b, s = 0, cigma;  
 for (int i = 0; i < n; i++) {  
 xArr[i] = Math.log(xArr[i]);  
 yArr[i] = Math.log(yArr[i]);  
 }  
 double[] linansw = linearApprox(xArr, yArr, n);  
 a = Math.pow(Math.E, linansw[1]);  
 b = linansw[0];  
 for (int i = 0; i < n; i++) {  
 xArr[i] = Math.pow(Math.E, xArr[i]);  
 yArr[i] = Math.pow(Math.E, yArr[i]);  
 s += Math.pow((a \* Math.pow(xArr[i], b) - yArr[i]), 2);  
 }  
 cigma = Math.sqrt(s/n);  
 return new double[]{a, b, s, cigma};  
 }  
  
 public static double[] expApprox(double[] xArr, double[] yArr, int n) {  
 double a, b, s = 0, cigma;  
 for (int i = 0; i < n; i++) {  
 yArr[i] = Math.log(yArr[i]);  
 }  
 double[] linansw = linearApprox(xArr, yArr, n);  
 a = Math.pow(Math.E, linansw[1]);  
 b = linansw[0];  
 for (int i = 0; i < n; i++) {  
 yArr[i] = Math.pow(Math.E, yArr[i]);  
 s += Math.pow((a \* Math.pow(Math.E, b \* xArr[i]) - yArr[i]), 2);  
 }  
 cigma = Math.sqrt(s/n);  
 return new double[]{a, b, s, cigma};  
 }  
  
 public static double[] logApprox(double[] xArr, double[] yArr, int n) {  
 double a, b, s = 0, cigma;  
 for (int i = 0; i < n; i++) {  
 xArr[i] = Math.log(xArr[i]);  
 }  
 double[] linansw = linearApprox(xArr, yArr, n);  
 a = linansw[0];  
 b = linansw[1];  
 for (int i = 0; i < n; i++) {  
 xArr[i] = Math.pow(Math.E, xArr[i]);  
 s += Math.pow((a \* Math.log(xArr[i]) + b - yArr[i]), 2);  
 }  
 cigma = Math.sqrt(s/n);  
 return new double[]{a, b, s, cigma};  
 }  
}

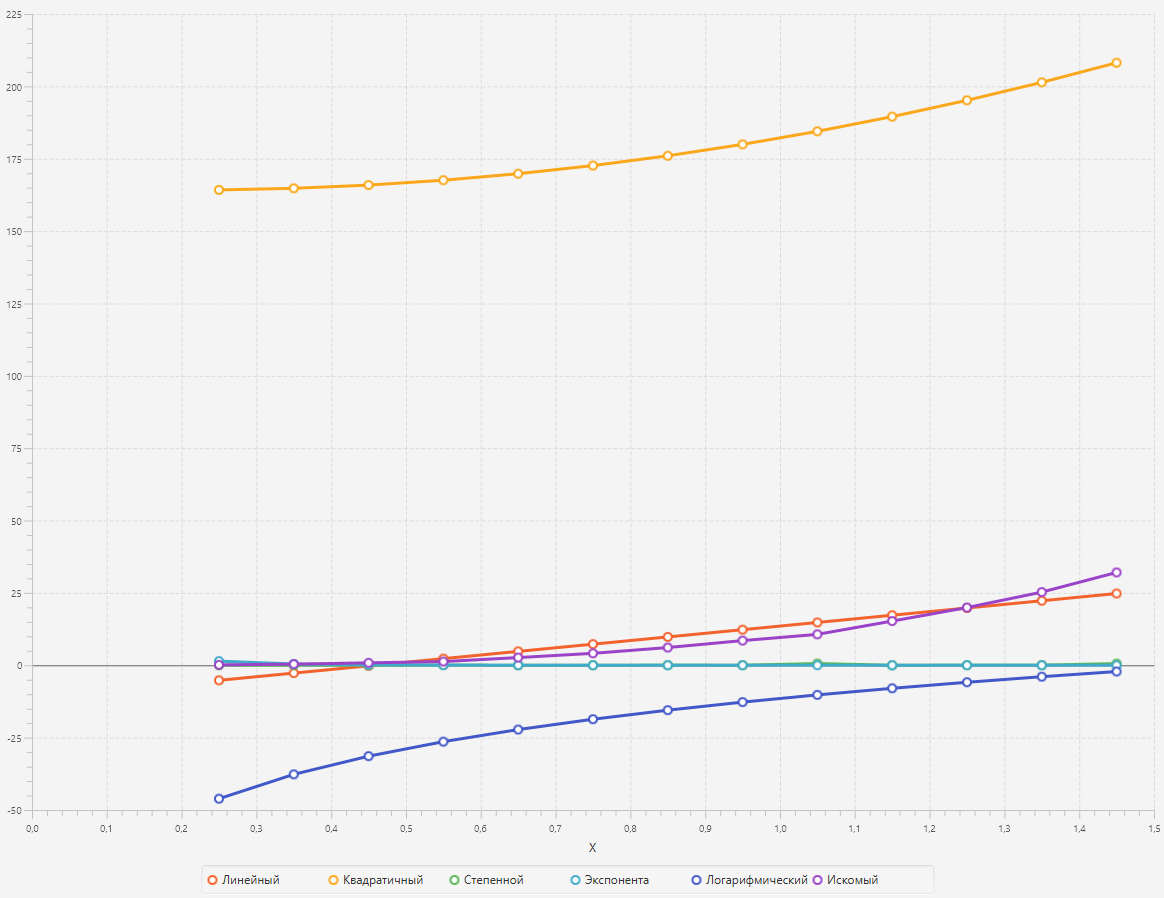
**Примеры работы программы:**

*Входные данные:*

0,25 0,35 0,45 0,55 0,65 0,75 0,85 0,95 1,05 1,15 1,25 1,35 1,45

0,134 0,381 0,845 1,278 2,659 4,125 6,1237 8,524 10,678 15,271 19,932 25,278 32,067

*Пример вывода графика:*



По графику видно, что функций должно быть 6, однако экспонента и степенная накладываются. Данное явление связано с масштабом графика, уменьшая который мы теряем квадратичную функцию. Придется чем-то пожертвовать

*Таблица в файле:*

Вид функции | a | b | c | S | Sigma |

f = a\*x + b | 24,992 | -11,451 | - | 165,320 | 3,566 |

f = a\*x^2 + b\*x + c | 28,269 | -23,065 | 5,016 | 5,336 | 0,641 |

f = a\*x^b | 9,831 | 3,117 | - | 1,418 | 0,330 |

f = a\*e^(bx) | 0,109 | 4,255 | - | 504,537 | 6,230 |

f = a\*lnx + b | 15,912 | 14,298 | - | 397,277 | 5,528 |

Наилучшая аппроксимирующая функция - Степенная!

Ссылка на github - [https://github.com/numvc/Comp\_Mathematics\_lab](https://github.com/numvc/Comp_Mathematics_lab3)4

**Вывод:** в результате выполнения лабораторной работы я научился искать приближение для какой-либо функции методом наименьших квадратов. За исходные данный может браться как сама функция, так и точки, соответствующие её координатам на плоскости. Также вспомнил статистические характеристики(отклонения), исходя из которых выбирается наилучшая аппроксимирующая функция.