Университет ИТМО, факультет ПИиКТ

Лабораторная работа №4

Дисциплина: Вычислительная математика

Вариант 17

Выполнил: Чайка Алексей

Группа: Р3214

Преподаватель: Малышева Т. А.

Санкт-Петербург, 2020 год

**Текст задания:**

**Лабораторная работа 4. «Аппроксимация функции методом наименьших квадратов»**

**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).

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

**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 использовался метод Крамера) 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*.

**Листинг:**

Main.java

**package** pepe.lmao;  
  
**import** pepe.lmao.method.SmallSquare;  
  
**import** java.io.\*;  
**import** java.util.Random;  
  
**public class** Main {  
 **static final double**[] ***x*** = **new double**[14];  
 **static final double**[] ***y*** = **new double**[14];  
  
 **public static void** main(String[] args) **throws** IOException {  
 *readFromFile*(**"input.txt"**);  
 SmallSquare smallSquare = **new** SmallSquare(***x***,***y***);  
 smallSquare.solve();  
 smallSquare.plot();  
 *writeToFile*(smallSquare.print());  
  
 }  
   
 **public static void** readFromFile(String filename) **throws** IOException {  
 String row;  
 String[] data;  
 BufferedReader reader = **new** BufferedReader(**new** FileReader(filename));  
 **while** ((row = reader.readLine()) != **null**) {  
 data = row.split(**","**);  
 **for** (**int** i = 0; i < data.**length**; i++) {  
 **if** (***x***[i] == 0.0)  
 ***x***[i] = Double.*parseDouble*(data[i]);  
 **else  
 *y***[i] = Double.*parseDouble*(data[i]);  
 }  
 }  
 }  
  
 **private static void** writeToFile(String data) **throws** IOException {  
 Random random = **new** Random();  
 String filename = **"output\_"** + Math.*abs*(random.nextInt()) + **".txt"**;  
 BufferedWriter writer = **new** BufferedWriter(**new** FileWriter(filename));  
 writer.write(data);  
 writer.close();  
 System.***out***.println(**"-->Done! Check the file: "** + filename);  
 }  
}

SmallSquare.java

**package** pepe.lmao.method;  
  
**import** lombok.Data;  
**import** pepe.lmao.approximation.\*;  
**import** pepe.lmao.chart.XYChart;  
  
**import** java.util.\*;  
  
@Data  
**public class** SmallSquare {  
 **private double**[] **x**;  
 **private double**[] **y**;  
 **private** String **key** = **"OriginalFunction"**;  
 **static** LinearApproximation *linearApproximation*;  
 **static** PolynomialApproximation *polynomialApproximation*;  
 **static** ExponentialApproximation *exponentialApproximation*;  
 **static** LogarithmicApproximation *logarithmicApproximation*;  
 **static** PowApproximation *powApproximation*;  
  
 **public** SmallSquare(**double**[] x, **double**[] y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 *linearApproximation* = **new** LinearApproximation(x, y);  
 *polynomialApproximation* = **new** PolynomialApproximation(x, y);  
 *exponentialApproximation* = **new** ExponentialApproximation(x, y);  
 *logarithmicApproximation* = **new** LogarithmicApproximation(x, y);  
 *powApproximation* = **new** PowApproximation(x, y);  
 }  
  
 **public void** solve() {  
 *linearApproximation*.result();  
 *polynomialApproximation*.result();  
 *exponentialApproximation*.result();  
 *logarithmicApproximation*.result();  
 *powApproximation*.result();  
 }  
  
 **public** String print() {  
 StringBuilder stringBuilder = **new** StringBuilder();  
 Formatter formatter = **new** Formatter(stringBuilder);  
 formatter.format(**"%25s %25s %25s %25s %25s %25s"**, **"| FUNCTION TYPE |"**, **"| a |"**, **"| b |"**, **"| c |"**, **"| DEVIATION MEASURE (S) |"**, **"| STANDARD DEVIATION |\n"**);  
 formatter.format(**"%25s %25s %25s %25s %25s %25s"**, **"f = ax + b"**, *linearApproximation*.getA(), *linearApproximation*.getB(), **"-"**,  
 *linearApproximation*.getDeviation\_measure(), *linearApproximation*.getStandard\_deviation() + **"\n"**);  
   
 formatter.format(**"%25s %25s %25s %25s %25s %25s"**, **"f = ax^b"**, *powApproximation*.getA(), *powApproximation*.getB(), **"-"**,  
 *powApproximation*.getDeviation\_measure(), *powApproximation*.getStandard\_deviation() + **"\n"**);  
   
 formatter.format(**"%25s %25s %25s %25s %25s %25s"**, **"f = ae^(bx)"**, *exponentialApproximation*.getA(), *exponentialApproximation*.getA(), **"-"**,  
 *exponentialApproximation*.getDeviation\_measure(), *exponentialApproximation*.getStandard\_deviation() + **"\n"**);  
   
 formatter.format(**"%25s %25s %25s %25s %25s %25s"**, **"f = alnx + b"**, *logarithmicApproximation*.getA(), *logarithmicApproximation*.getA(), **"-"**,  
 *logarithmicApproximation*.getDeviation\_measure(), *logarithmicApproximation*.getStandard\_deviation() + **"\n"**);  
   
 formatter.format(**"%25s %25s %25s %25s %25s %25s"**, **"f = ax^2bx + c"**, *polynomialApproximation*.getA(), *polynomialApproximation*.getB(), *polynomialApproximation*.getC(),  
 *polynomialApproximation*.getDeviation\_measure(), *polynomialApproximation*.getStandard\_deviation());  
  
 HashMap<String, Double> tmp = **new** HashMap<>();  
 tmp.put(*linearApproximation*.getKey(), *linearApproximation*.getStandard\_deviation());  
 tmp.put(*polynomialApproximation*.getKey(), *polynomialApproximation*.getStandard\_deviation());  
 tmp.put(*exponentialApproximation*.getKey(), *exponentialApproximation*.getStandard\_deviation());  
 tmp.put(*logarithmicApproximation*.getKey(), *logarithmicApproximation*.getStandard\_deviation());  
 tmp.put(*powApproximation*.getKey(), *powApproximation*.getStandard\_deviation());  
 Map.Entry<String, Double> min = **null**;  
 **for** (Map.Entry<String, Double> entry : tmp.entrySet()) {  
 **if** (min == **null** || min.getValue() > entry.getValue()) {  
 min = entry;  
 }  
 }  
 **assert** min != **null**;  
 stringBuilder.append(**"\nBest approximate result is shown by: "**)  
 .append(min.getKey())  
 .append(**" \nwith the least standard deviation equals to: "**)  
 .append(min.getValue());  
 **return** stringBuilder.toString();  
 }  
  
 **public void** plot() {  
 XYChart xyChart = **new** XYChart();  
 xyChart.prepareChart(getY(), getKey());  
 xyChart.prepareChart(*linearApproximation*.getApproximateResult(), *linearApproximation*.getKey());  
 xyChart.prepareChart(*polynomialApproximation*.getApproximateResult(), *polynomialApproximation*.getKey());  
 xyChart.prepareChart(*exponentialApproximation*.getApproximateResult(), *exponentialApproximation*.getKey());  
 xyChart.prepareChart(*logarithmicApproximation*.getApproximateResult(), *logarithmicApproximation*.getKey());  
 xyChart.prepareChart(*powApproximation*.getApproximateResult(), *powApproximation*.getKey());  
 xyChart.pack();  
 xyChart.setVisible(**true**);  
 }  
}

LinearApproximation.java

**package** pepe.lmao.approximation;  
  
**import** lombok.Data;  
**import** org.apache.commons.math3.util.FastMath;  
  
@Data  
**public class** LinearApproximation {  
 **private double**[] **x**;  
 **private double**[] **y**;  
 **private int n**;  
   
 *//output variables* **private double a**;  
 **private double b**;  
 **private double deviation\_measure** = 0, **standard\_deviation** = 0;  
 **private double**[] **approximateResult**;  
 **private** String **key** = **"LinearApproximation"**;  
  
 **public** LinearApproximation(**double**[] x, **double**[] y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 **this**.**n** = x.**length**;  
 **this**.**approximateResult** = **new double**[**n**];  
 }  
  
 **public void** result() {  
 findSolution();  
 }  
  
 **private void** findSolution() {  
 **double** SX = 0, SXX = 0, SY = 0, SXY = 0;  
 **double**[] eps = **new double**[**n**];  
 **double** correlation;  
 **for** (**int** i = 0; i < **n**; i++) {  
 SX += **x**[i];  
 SXX += FastMath.*pow*(**x**[i], 2);  
 SY += **y**[i];  
 SXY += **x**[i] \* **y**[i];  
 }  
 correlation = CorrelationUtil.*calcCorrelation*(**x**, **y**);  
 **a** = (SXY \* **n** - SX \* SY) / (SXX \* **n** - FastMath.*pow*(SX, 2));  
 **b** = (SXX \* SY - SX \* SXY) / (SXX \* **n** - FastMath.*pow*(SX, 2));  
 **for** (**int** i = 0; i < **n**; i++) {  
 **approximateResult**[i] = f(**x**[i]);  
 eps[i] = **approximateResult**[i] - **y**[i];  
 **deviation\_measure** += FastMath.*pow*(eps[i], 2);  
 }  
 **standard\_deviation** = FastMath.*sqrt*(**deviation\_measure**/**n**);  
 }  
  
 **public double** f(**double** x) {  
 **return a** \* x + **b**;  
 }  
}

PolynomialApproximation.java

**package** pepe.lmao.approximation;  
  
**import** lombok.Data;  
**import** org.apache.commons.math3.util.FastMath;  
**import** pepe.lmao.cramer.CramersRuleImplementation;  
  
@Data  
**public class** PolynomialApproximation {  
 **private double**[] **x**;  
 **private double**[] **y**;  
 **private int n**;  
 *//output variables* **private double a**, **b**, **c**;  
 **double deviation\_measure** = 0, **standard\_deviation** = 0;  
 **private double**[] **approximateResult**;  
 **private** String **key** = **"PolynomialApproximation"**;  
 **public** PolynomialApproximation(**double**[] x, **double**[] y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 **this**.**n** = x.**length**;  
 **this**.**approximateResult** = **new double**[**n**];  
 }  
 **public void** result() {  
 findSolution();  
 }  
 **private void** findSolution() {  
 **double** SX = 0, SXX = 0, SXXX = 0, SXXXX = 0, SY = 0, SXY = 0, SXXY = 0;  
 **int** n = **x**.**length**;  
 **double**[] eps = **new double**[n];  
 **double** correlation;  
 **for** (**int** i = 0; i < n; i++) {  
 SX += **x**[i];  
 SXX += FastMath.*pow*(**x**[i], 2);  
 SXXX += FastMath.*pow*(**x**[i], 3);  
 SXXXX += FastMath.*pow*(**x**[i], 4);  
 SY += **y**[i];  
 SXY += **x**[i] \* **y**[i];  
 SXXY += FastMath.*pow*(**x**[i], 2) \* **y**[i];  
 }  
 **double**[][] coefficients = {  
 {n, SX, SXX, SY},  
 {SX, SXX, SXXX, SXY},  
 {SXX, SXXX, SXXXX, SXXY},  
 };  
 CramersRuleImplementation cramersRuleImplementation = **new** CramersRuleImplementation();  
 cramersRuleImplementation.findSolution(coefficients);  
 **a** = CramersRuleImplementation.*a0*;  
 **b** = CramersRuleImplementation.*a1*;  
 **c** = CramersRuleImplementation.*a2*;  
 **for** (**int** i = 0; i < n; i++) {  
 **approximateResult**[i] = f(**x**[i]);  
 eps[i] = **approximateResult**[i] - **y**[i];  
 **deviation\_measure** += FastMath.*pow*(eps[i], 2);  
 }  
 **standard\_deviation** = FastMath.*sqrt*(**deviation\_measure**/n);  
 correlation = CorrelationUtil.*calcCorrelation*(**x**,**y**);  
 }  
  
 **private double** f(**double** x) {  
 **return a** + **b**\*x + **c** \* FastMath.*pow*(x,2);  
 }  
}

ExponentialApproximation.java

**package** pepe.lmao.approximation;  
  
**import** lombok.Data;  
**import** org.apache.commons.math3.util.FastMath;  
  
@Data  
**public class** ExponentialApproximation {  
 **private double**[] **x**;  
 **private double**[] **y**;  
 **private double A**;  
 **private double B**;  
 **private int n**;  
 *//output variables* **private double a**;  
 **private double b**;  
 **double deviation\_measure** = 0, **standard\_deviation** = 0;  
 **private double**[] **approximateResult**;  
 **private** String **key** = **"ExponentialApproximation"**;  
  
 **public** ExponentialApproximation(**double**[] x, **double**[] y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 **this**.**n** = x.**length**;  
 **this**.**approximateResult** = **new double**[**n**];  
 }  
  
 **public void** result() {  
 findSolution();  
 }  
  
 **private void** findSolution() {  
 **double** SX = 0, SXX = 0, SY = 0, SXY = 0;  
 **double**[] y\_log = **new double**[**n**];  
 **double**[] eps = **new double**[**n**];  
 **double** correlation;  
 **for** (**int** i = 0; i < **n**; i++) {  
 y\_log[i] = FastMath.*log*(**y**[i]);  
 }  
 **for** (**int** i = 0; i < **n**; i++) {  
 SX += **x**[i];  
 SXX += FastMath.*pow*(**x**[i], 2);  
 SY += y\_log[i];  
 SXY += **x**[i] \* y\_log[i];  
 }  
 correlation = CorrelationUtil.*calcCorrelation*(**x**, **y**);  
 **A** = (SXY \* **n** - SX \* SY) / (SXX \* **n** - FastMath.*pow*(SX, 2));  
 **B** = (SXX \* SY - SX \* SXY) / (SXX \* **n** - FastMath.*pow*(SX, 2));  
 **a** = FastMath.*exp*(**B**);  
 **b** = **A**;  
 **for** (**int** i = 0; i < **n**; i++) {  
 **approximateResult**[i] = f(**x**[i]);  
 eps[i] = **approximateResult**[i] - **y**[i];  
 **deviation\_measure** += FastMath.*pow*(eps[i], 2);  
 }  
 **standard\_deviation** = FastMath.*sqrt*(**deviation\_measure**/**n**);  
 }  
 **private double** f(**double** x) {  
 **return a**\*FastMath.*pow*(FastMath.***E***, **b**\*x);  
 }  
}

LogarithmicApproximation.java

**package** pepe.lmao.approximation;  
  
**import** lombok.Data;  
**import** org.apache.commons.math3.util.FastMath;  
  
@Data  
**public class** LogarithmicApproximation {  
 **private double**[] **x**;  
 **private double**[] **y**;  
 **private int n**;  
 *//output variables* **private double a**;  
 **private double b**;  
 **double deviation\_measure** = 0, **standard\_deviation** = 0;  
 **private double**[] **approximateResult**;  
 **private** String **key** = **"LogarithmicApproximation"**;  
  
 **public** LogarithmicApproximation(**double**[] x, **double**[] y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 **this**.**n** = x.**length**;  
 **this**.**approximateResult** = **new double**[**n**];  
 }  
 **public void** result() {  
 findSolution();  
 }  
 **private void** findSolution() {  
 **double** SX = 0, SXX = 0, SY = 0, SXY = 0;  
 **int** n = **x**.**length**;  
 **double**[] x\_log = **new double**[n];  
 **double**[] eps = **new double**[n];  
 **double** correlation;  
 **for** (**int** i = 0; i < n; i++) {  
 x\_log[i] = FastMath.*log*(**x**[i]);  
 }  
 **for** (**int** i = 0; i < n; i++) {  
 SX += x\_log[i];  
 SXX += FastMath.*pow*(x\_log[i], 2);  
 SY += **y**[i];  
 SXY += x\_log[i] \* **y**[i];  
 }  
 correlation = CorrelationUtil.*calcCorrelation*(**x**, **y**);  
 **a** = (SXY \* n - SX \* SY) / (SXX \* n - FastMath.*pow*(SX, 2));  
 **b** = (SXX \* SY - SX \* SXY) / (SXX \* n - FastMath.*pow*(SX, 2));  
 **for** (**int** i = 0; i < n; i++) {  
 **approximateResult**[i] = f(**x**[i]);  
 eps[i] = **approximateResult**[i] - **y**[i];  
 **deviation\_measure** += FastMath.*pow*(eps[i], 2);  
 }  
 **standard\_deviation** = FastMath.*sqrt*(**deviation\_measure**/n);  
 }  
  
 **private double** f(**double** x) {  
 **return a**\*FastMath.*log*(x) + **b**;  
 }  
}

PowApproximation.java

**package** pepe.lmao.approximation;  
  
**import** lombok.Data;  
**import** org.apache.commons.math3.util.FastMath;  
  
**import** java.util.Arrays;  
@Data  
**public class** PowApproximation {  
 **private double**[] **x**;  
 **private double**[] **y**;  
 **private double A**;  
 **private double B**;  
 **private int n**;  
 *//output variables* **private double a**;  
 **private double b**;  
 **private double deviation\_measure** = 0, **standard\_deviation** = 0;  
 **private double**[] **approximateResult**;  
 **private** String **key** = **"PowApproximation"**;  
 **public** PowApproximation(**double**[] x, **double**[] y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 **this**.**n** = x.**length**;  
 **this**.**approximateResult** = **new double**[**n**];  
 }  
  
 **public void** result() {  
 findSolution();  
 }  
  
 **private void** findSolution() {  
 **double** SX = 0, SXX = 0, SY = 0, SXY = 0;  
 **int** n = **x**.**length**;  
 **double**[] x\_log = **new double**[n];  
 **double**[] y\_log = **new double**[n];  
 **double**[] eps = **new double**[n];  
 **double** correlation;  
 **for** (**int** i = 0; i < n; i++) {  
 x\_log[i] = FastMath.*log*(**x**[i]);  
 y\_log[i] = FastMath.*log*(**y**[i]);  
 }  
 **for** (**int** i = 0; i < n; i++) {  
 SX += x\_log[i];  
 SXX += FastMath.*pow*(x\_log[i], 2);  
 SY += y\_log[i];  
 SXY += x\_log[i] \* y\_log[i];  
 }  
 correlation = CorrelationUtil.*calcCorrelation*(**x**, **y**);  
 **A** = (SXY \* n - SX \* SY) / (SXX \* n - FastMath.*pow*(SX, 2));  
 **B** = (SXX \* SY - SX \* SXY) / (SXX \* n - FastMath.*pow*(SX, 2));  
 **a** = FastMath.*exp*(**B**);  
 **b** = **A**;  
 **for** (**int** i = 0; i < n; i++) {  
 **approximateResult**[i] = f(**x**[i]);  
 eps[i] = **approximateResult**[i] - **y**[i];  
 **deviation\_measure** += FastMath.*pow*(eps[i], 2);  
 }  
 **standard\_deviation** = FastMath.*sqrt*(**deviation\_measure**/n);  
 }  
  
 **private double** f(**double** x) {  
 **return a**\*FastMath.*pow*(x,**b**);  
 }  
}

CorrelationUtil.java

**package** pepe.lmao.approximation;  
  
**import** org.apache.commons.math3.util.FastMath;  
  
**public class** CorrelationUtil {  
 **public static double** calcCorrelation(**double**[] x, **double**[] y) {  
 **double** SX = 0, SY = 0, avg\_x = 0, avg\_y = 0;  
 **double** cor\_sum = 0, cor\_x = 0, cor\_y = 0;  
 **int** n = x.**length**;  
 **for** (**int** i = 0; i < n; i++) {  
 SX += x[i];  
 SY += y[i];  
 }  
 avg\_x = SX / n;  
 avg\_y = SY / n;  
 **for** (**int** i = 0; i < n; i++) {  
 cor\_sum += (x[i] - avg\_x) \* (y[i] - avg\_y);  
 cor\_x += FastMath.*pow*((x[i] - avg\_x), 2);  
 cor\_y += FastMath.*pow*((y[i] - avg\_y), 2);  
 }  
 **return** cor\_sum / (FastMath.*sqrt*(cor\_x \* cor\_y));  
 }  
}

CramersRuleImplementation.java

**package** pepe.lmao.cramer;  
  
**import** lombok.Data;  
  
@Data  
**public class** CramersRuleImplementation {  
 **public static double** *a0*;  
 **public static double** *a1*;  
 **public static double** *a2*;  
  
 **private static double** determinantOfMatrix(**double**[][] mat) {  
 **double** ans;  
 ans = mat[0][0] \* (mat[1][1] \* mat[2][2] - mat[2][1] \* mat[1][2])  
 - mat[0][1] \* (mat[1][0] \* mat[2][2] - mat[1][2] \* mat[2][0])  
 + mat[0][2] \* (mat[1][0] \* mat[2][1] - mat[1][1] \* mat[2][0]);  
 **return** ans;  
 }  
  
 **public void** findSolution(**double**[][] coeff) {  
 **double**[][] d = {  
 {coeff[0][0], coeff[0][1], coeff[0][2]},  
 {coeff[1][0], coeff[1][1], coeff[1][2]},  
 {coeff[2][0], coeff[2][1], coeff[2][2]},  
 };  
 **double**[][] d1 = {  
 {coeff[0][3], coeff[0][1], coeff[0][2]},  
 {coeff[1][3], coeff[1][1], coeff[1][2]},  
 {coeff[2][3], coeff[2][1], coeff[2][2]},  
 };  
 **double**[][] d2 = {  
 {coeff[0][0], coeff[0][3], coeff[0][2]},  
 {coeff[1][0], coeff[1][3], coeff[1][2]},  
 {coeff[2][0], coeff[2][3], coeff[2][2]},  
 };  
 **double**[][] d3 = {  
 {coeff[0][0], coeff[0][1], coeff[0][3]},  
 {coeff[1][0], coeff[1][1], coeff[1][3]},  
 {coeff[2][0], coeff[2][1], coeff[2][3]},  
 };  
 **double** D = *determinantOfMatrix*(d);  
 **double** D1 = *determinantOfMatrix*(d1);  
 **double** D2 = *determinantOfMatrix*(d2);  
 **double** D3 = *determinantOfMatrix*(d3);  
 **if** (D != 0) {  
 *a0* = D1 / D;  
 *a1* = D2 / D;  
 *a2* = D3 / D;  
 } **else** {  
 **if** (D1 == 0 && D2 == 0 && D3 == 0)  
 System.***out***.println(**"Infinite solutions"**);  
 **else if** (D1 != 0 || D2 != 0 || D3 != 0)  
 System.***out***.println(**"No solutions"**);  
 }  
 }  
}

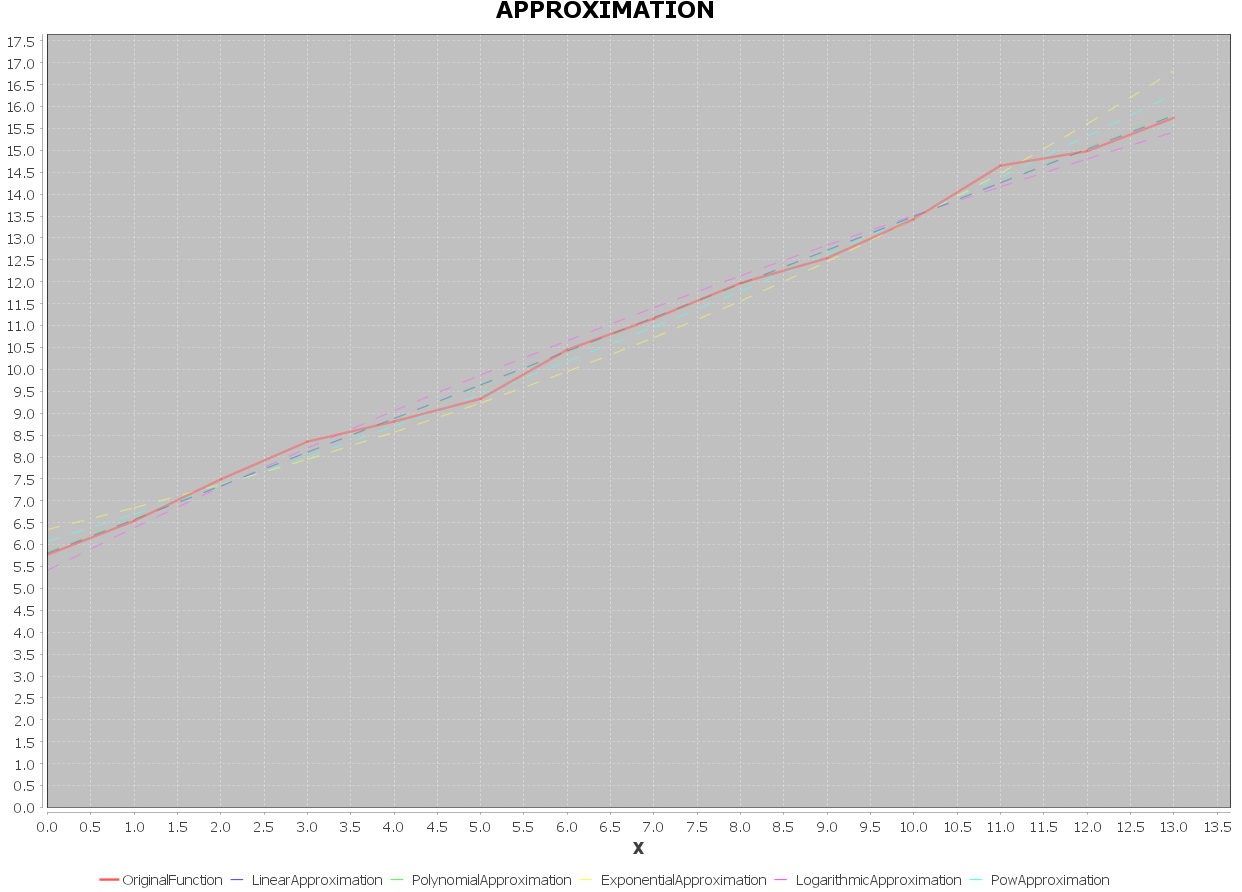
XYChart.java

**package** pepe.lmao.chart;  
  
**import** org.jfree.chart.ChartFactory;  
**import** org.jfree.chart.ChartPanel;  
**import** org.jfree.chart.JFreeChart;  
**import** org.jfree.chart.plot.XYPlot;  
**import** org.jfree.chart.renderer.AbstractRenderer;  
**import** org.jfree.data.xy.XYSeries;  
**import** org.jfree.data.xy.XYSeriesCollection;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** XYChart **extends** JFrame {  
 **final** XYSeriesCollection **dataset** = **new** XYSeriesCollection();  
 **public** XYPlot prepareChart(**double**[] f, String key) {  
 **int** i = 0;  
 XYSeries series = **new** XYSeries(key);  
 **for** (**double** d : f) {  
 series.add(i++, d);  
 }  
 **dataset**.addSeries(series);  
 JFreeChart chart = ChartFactory.*createXYLineChart*(**"APPROXIMATION"**, **"X"**, **"Y"**, **dataset**);  
 ChartPanel chartPanel = **new** ChartPanel(chart);  
 chartPanel.setPreferredSize(**new** java.awt.Dimension(1280, 900));  
 setContentPane(chartPanel);  
 XYPlot plot = chart.getXYPlot();  
 ((AbstractRenderer) plot.getRenderer()).setAutoPopulateSeriesStroke(**false**);  
 ((AbstractRenderer) plot.getRenderer()).setSeriesStroke(0, **new** BasicStroke(2.0f));  
 **for** (**int** j = 1; j < 6; j++) {  
 ((AbstractRenderer) plot.getRenderer()).setSeriesStroke(j, **new** BasicStroke(1.0f,BasicStroke.***CAP\_BUTT***, BasicStroke.***JOIN\_MITER***, 10.0f, **new float**[] {10.0f}, 0.0f));  
 }  
 plot.setForegroundAlpha(0.5f);  
 **return** plot;  
 }  
}

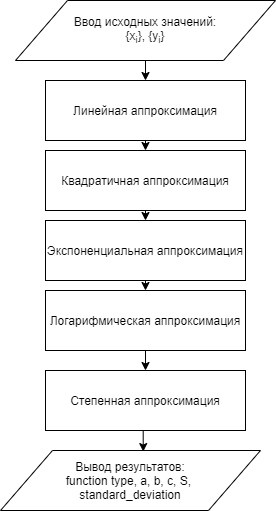
**Примеры работы программы:**

|  |
| --- |
| input.txt  2,2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9,3,3.1,3.2,3.3 5.77,6.54,7.49,8.35,8.81,9.33,10.45,11.16,11.97,12.54,13.43,14.65,14.98,15.74  console  -->Done! Check the file: output\_1426006342.txt  output\_1426006342.txt |

**Пример графика:**



**Блок-схема:**

****

**Вывод:** в результате выполнения работы я познакомился с аппроксимацией функции методом наименьших квадратов путем реализации Java-программы. В ходе выполнения работы, встречал череду тривиальных проблем, решение которых не занимало много времени.