**Министерство образования и науки Российской Федерации**

**Федеральное государственное автономное образовательное**

**учреждение высшего образования**

**«Университет ИТМО»**

**Мегафакультет:** Компьютерных технологий и управления

**Направление:** 09.03.04 «Программная инженерия»

**Лабораторная работа 4**

По дисциплине:

«Вычислительная математика»

Вариант 10

На тему:

«Аппроксимация функций»

**Выполнила:**

студентка группы **P3214**

Маньшина Елена

Витальевна

**Преподаватель:**

Малышева Татьяна

Алексеевна

Санкт-Петербург

2020

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вариант  10 | X | 1,1 | 2,2 | 3,3 | 4,4 | 5,5 | 6,6 | 7,7 | 8,8 | 9,9 | 11 | 12,1 |
| Y | 7,42 | 5,12 | 4,42 | 4,78 | 5,75 | 6,92 | 8,03 | 10,18 | 13,17 | 15,7 | 18,87 |

**«Аппроксимация функции методом наименьших квадратов»**

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

Для исследования использовать:

1. **линейную функцию;**

Рассмотрим в качестве эмпирической формулы линейную функцию:

Необходимое условие существования минимума для функции S:

После упрощения:

Тогда

1. **полиномиальную функцию 2-й степени;**

Рассмотрим в качестве эмпирической формулы квадратичную функцию:

Необходимое условие существования минимума для функции S:

После упрощения:

Вычислим:

Полученная система уравнений:

1. **экспоненциальную функцию;**

Аппроксимирующая функция задана экспоненциальной функцией вида:

Линеаризируем:

=

Поиск неизвестных коэффициентов осуществляется по методу наименьших квадратов в соответствии со следующей системой уравнений.

1. **логарифмическую функцию;**

Аппроксимирующая функция задана логарифмической функцией вида:

Поиск неизвестных коэффициентов осуществляется по методу наименьших квадратов в соответствии со следующей системой уравнений.

1. **степенную функцию.**

Аппроксимирующая функция задана экспоненциальной функцией вида:

Линеаризируем:

=

Поиск неизвестных коэффициентов осуществляется по методу наименьших квадратов в соответствии со следующей системой уравнений.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Вид *φ(х)*** | | **Линейная** | **Полиномиальная** | **Экспоненциальная** | **Логарифмическая** | **Степенная** |
| **Х** | **Y** | *F=ax+b* | *F=ax2+bx+c* | *F=aebx* | *F=* | *F=axb* |
| **1,1** | **7,42** | 2,810 | 6,633 | 4,247 | 2,202 | 3,993 |
| **2,2** | **5,12** | 4,073 | 5,600 | 4,836 | 5,218 | 5,439 |
| **3,3** | **4,42** | 5,335 | 5,075 | 5,505 | 6,982 | 6,517 |
| **4,4** | **4,78** | 6,598 | 5,059 | 6,268 | 8,233 | 7,410 |
| **5,5** | **5,75** | 7,861 | 5,551 | 7,136 | 9,204 | 8,185 |
| **6,6** | **6,92** | 9,124 | 6,550 | 8,125 | 9,998 | 8,879 |
| **7,7** | **8,03** | 10,386 | 8,059 | 9,250 | 10,668 | 9,510 |
| **8,8** | **10,18** | 11,649 | 10,075 | 10,532 | 11,249 | 10,094 |
| **9,9** | **13,17** | 12,912 | 12,599 | 11,991 | 11,762 | 10,638 |
| **11** | **15,7** | 14,175 | 15,632 | 13,652 | 12,220 | 11,150 |
| **12,1** | **18,87** | 15,437 | 19,173 | 15,543 | 12,635 | 11,634 |
| **S** | | 57,693 | 1,967 | 35,184 | 128,206 | 114,589 |
| **δ** | | 2,290 | 0,423 | 1,788 | 3,414 | 3,228 |
| **a** | | 1,148 | 0,210 | 3,731 | 4,351 | 3,827 |
| **b** | | 1,547 | -1,632 | 0,118 | 1,787 | 0,446 |
| **c** | | - | 8,174 | - | - | - |

Наилучшая аппроксимация осуществляется с помощью квадратичной функции.

1. Программная реализация задачи:
   1. Предусмотреть ввод исходных данных (таблица *y=f(x)*) из файла.
   2. Реализовать метод наименьших квадратов, исследуя все функции п.1.
   3. Предусмотреть вывод результатов (см. табл. ниже) в файл.
   4. Программа должна выбирать наилучшую аппроксимирующую функцию.
   5. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом).

Реализованные методы

**public class** Function {  
 **double**[][] **table**;  
 **protected int n**;  
 **protected double a**;  
 **protected double b**;  
 **protected double c**;  
 **protected double**[] **f**;  
 **protected double S** = 0;  
 **protected double δ**;  
  
 **public** Function(**double**[][] table) {  
 **this**.**table** = table;  
 **n** = table[0].**length**;  
 **f** = **new double**[**n**];  
 }  
  
 **protected void** countFANDParam(java.util.function.Function<Double, Double> fun) {  
 **for** (**int** i = 0; i < **table**[0].**length**; i++) {  
 **f**[i] = fun.apply(**table**[0][i]);  
 **S** = **S** + Math.*pow*(**f**[i] - **table**[1][i], 2);  
 }  
 **δ** = Math.*sqrt*(**S** / **n**);  
 }  
  
 **public double**[] getF() {  
 **return f**;  
 }  
 **public double** getS() {  
 **return S**;  
 }  
 **public double** getδ() {  
 **return δ**;  
 }  
 **public double** getA() {  
 **return a**;  
 }  
 **public double** getB() {  
 **return b**;  
 }  
 **public double** getC() {  
 **return c**;  
 }  
}

**public class** LinearFunction **extends** Function {  
 **private double SX** = 0;  
 **private double SXX** = 0;  
 **private double SY** = 0;  
 **private double SXY** = 0;  
  
 **public** LinearFunction(**double**[][] table) {  
 **super**(table);  
 init();  
 }  
 **private void** init() {  
 **n** = **table**[0].**length**;  
 **for** (**int** i = 0; i < **n**; i++) {  
 **SX** = **SX** + **table**[0][i];  
 **SXX** = **SXX** + **table**[0][i] \* **table**[0][i];  
 **SY** = **SY** + **table**[1][i];  
 **SXY** = **SXY** + **table**[0][i] \* **table**[1][i];  
 }  
 countA();  
 countB();  
 countFANDParam((Double x) -> **a** \* x + **b**);  
 }  
  
 **private void** countA() {  
 **a** = (**SXY** \* **n** - **SX** \* **SY**) / (**SXX** \* **n** - **SX** \* **SX**);  
 }  
 **private void** countB() {  
 **b** = (**SXX** \* **SY** - **SX** \* **SXY**) / (**SXX** \* **n** - **SX** \* **SX**);  
 }  
  
}

**import** Gauss.GaussMethod;  
  
**public class** QuadraticFunction **extends** Function {  
 **private double SX** = 0;  
 **private double SXX** = 0;  
 **private double SXXX** = 0;  
 **private double SXXXX** = 0;  
 **private double SY** = 0;  
 **private double SXY** = 0;  
 **private double SXXY** = 0;  
  
 **private double**[][] **matrix** = **new double**[3][4];  
  
 **public** QuadraticFunction(**double**[][] table) {  
 **super**(table);  
 init();  
 }  
 **private void** init() {  
 **n** = **table**[0].**length**;  
 **for** (**int** i = 0; i < **n**; i++) {  
 **SX** = **SX** + **table**[0][i];  
 **SXX** = **SXX** + **table**[0][i] \* **table**[0][i];  
 **SXXX** = **SXXX** + **table**[0][i] \* **table**[0][i] \* **table**[0][i];  
 **SXXXX** = **SXXXX** + **table**[0][i] \* **table**[0][i] \* **table**[0][i] \* **table**[0][i];  
 **SY** = **SY** + **table**[1][i];  
 **SXY** = **SXY** + **table**[0][i] \* **table**[1][i];  
 **SXXY** = **SXXY** + **table**[0][i] \* **table**[0][i] \* **table**[1][i];  
 }  
 countABC();  
 countFANDParam((Double x) -> **a** \* x \* x + **b** \* x + **c**);  
  
 }  
  
 **private void** countABC() {  
 **matrix**[0][0] = **n**;  
 **matrix**[1][0] = **SX**;  
 **matrix**[0][1] = **SX**;  
 **matrix**[2][0] = **SXX**;  
 **matrix**[1][1] = **SXX**;  
 **matrix**[0][2] = **SXX**;  
 **matrix**[2][1] = **SXXX**;  
 **matrix**[1][2] = **SXXX**;  
 **matrix**[2][2] = **SXXXX**;  
 **matrix**[0][3] = **SY**;  
 **matrix**[1][3] = **SXY**;  
 **matrix**[2][3] = **SXXY**;  
 GaussMethod gaussMethod = **new** GaussMethod(**matrix**);  
 **double**[] answer = gaussMethod.getAnswer();  
 **a** = answer[2];  
 **b** = answer[1];  
 **c** = answer[0];  
 }  
  
}

**public class** ExponentialFunction **extends** Function {  
 **private double SX** = 0;  
 **private double SXX** = 0;  
 **private double SLNY** = 0;  
 **private double SLNYX** = 0;  
  
  
 **public** ExponentialFunction(**double**[][] table) {  
 **super**(table);  
 init();  
 }  
 **private void** init() {  
 **n** = **table**[0].**length**;  
 **for** (**int** i = 0; i < **n**; i++) {  
 **SX** = **SX** + **table**[0][i];  
 **SXX** = **SXX** + **table**[0][i] \* **table**[0][i];  
 **SLNY** = **SLNY** + Math.*log*(**table**[1][i]);  
 **SLNYX** = **SLNYX** + Math.*log*(**table**[1][i]) \* **table**[0][i];  
 }  
 countA();  
 countB();  
 countFANDParam((Double x) -> **a** \* Math.*exp*(**b** \* x));  
 }  
  
 **private void** countA() {  
 **double** LNA = (**SLNY** \* **SXX** - **SLNYX** \* **SX**) / (**n** \* **SXX** - **SX** \* **SX**);  
 **a** = Math.*exp*(LNA);  
 }  
  
 **private void** countB() {  
 **b** = (**n** \* **SLNYX** - **SLNY** \* **SX**) / (**n** \* **SXX** - **SX** \* **SX**);  
 }  
  
}

**public class** LogarithmicFunction **extends** Function {  
 **private double SY** = 0;  
 **private double SLNX** = 0;  
 **private double SLNXLNX** = 0;  
 **private double SYLNX** = 0;  
  
 **public** LogarithmicFunction(**double**[][] table) {  
 **super**(table);  
 init();  
 }  
  
 **private void** init() {  
 **n** = **table**[0].**length**;  
 **for** (**int** i = 0; i < **n**; i++) {  
 **SY** = **SY** + **table**[1][i];  
 **SLNX** = **SLNX** + Math.*log*(**table**[0][i]);  
 **SLNXLNX** = **SLNXLNX** + Math.*log*(**table**[0][i]) \* Math.*log*(**table**[0][i]);  
 **SYLNX** = **SYLNX** + **table**[1][i] \* Math.*log*(**table**[0][i]);  
 }  
 countA();  
 countB();  
 countFANDParam((Double x) -> **a** \* Math.*log*(x) + **b**);  
 }  
  
 **private void** countA() {  
 **a** = (**SY** \* **SLNXLNX** - **SYLNX** \* **SLNX**) / (**n** \* **SLNXLNX** - **SLNX** \* **SLNX**);  
 }  
  
 **private void** countB() {  
 **b** = (**n** \* **SYLNX** - **SY** \* **SLNX**) / (**n** \* **SLNXLNX** - **SLNX** \* **SLNX**);  
 }  
  
}

**public class** PowerFunction **extends** Function {  
 **private double SLNY** = 0;  
 **private double SLNX** = 0;  
 **private double SLNYLNX** = 0;  
 **private double SLNXLNX** = 0;  
  
 **public** PowerFunction(**double**[][] table) {  
 **super**(table);  
 init();  
 }  
 **private void** init() {  
 **n** = **table**[0].**length**;  
 **for** (**int** i = 0; i < **n**; i++) {  
 **SLNY** = **SLNY** + Math.*log*(**table**[1][i]);  
 **SLNX** = **SLNX** + Math.*log*(**table**[0][i]);  
 **SLNYLNX** = **SLNYLNX** + Math.*log*(**table**[1][i]) \* Math.*log*(**table**[0][i]);  
 **SLNXLNX** = **SLNXLNX** + Math.*log*(**table**[0][i]) \* Math.*log*(**table**[0][i]);  
 }  
 countA();  
 countB();  
 countFANDParam((Double x) -> **a** \* Math.*pow*(x, **b**));  
 }  
  
 **private void** countA() {  
 **double** LNA = (**SLNY** \* **SLNXLNX** - **SLNYLNX** \* **SLNX**) / (**n** \* **SLNXLNX** - **SLNX** \* **SLNX**);  
 **a** = Math.*exp*(LNA);  
 }  
  
 **private void** countB() {  
 **b** = (**n** \* **SLNYLNX** - **SLNY** \* **SLNX**) / (**n** \* **SLNXLNX** - **SLNX** \* **SLNX**);  
 }  
  
}

**Решение системы уравнений методом Гаусса**

**public class** GaussMethod {  
 **private double**[][] **matrix**;  
 **private double**[][] **firstMatrix**;  
 **private double**[] **x**;  
  
 **public** GaussMethod(**double**[][] matrix) {  
 **this**.**matrix** = matrix;  
 }  
  
 **public void** forwardStroke() {  
 *//копирование массива для проверки невязок* **double** changes = 0;  
 **firstMatrix** = **new double**[**matrix**.**length**][**matrix**[0].**length**];  
 **for** (**int** i = 0; i < **matrix**.**length**; i++) {  
 **for** (**int** j = 0; j < **matrix**[0].**length**; j++) {  
 **firstMatrix**[i][j] = **matrix**[i][j];  
 }  
 }  
  
 **for** (**int** j = 0; j < **matrix**.**length**; j++) {  
 **int** maxElemInColumn = maxElem(**matrix**, j);  
 **if** (maxElemInColumn != j) {*//если диагональный элемент не максимален в столбце, то происходит обмен строками* **matrix** = changeRows(**matrix**, j, maxElemInColumn);  
 changes += 1;  
 }  
  
 *//прямой метод* **for** (**int** row = j + 1; row < **matrix**.**length**; row++) {  
 **double** с = (**matrix**[row][j] / **matrix**[j][j]);  
 **matrix**[row][j] = 0; *//обнуляем значения элементов ниже главной диагонали* **for** (**int** column = j + 1; column < **matrix**[0].**length**; column++) {  
 **matrix**[row][column] = **matrix**[row][column] - с \* **matrix**[j][column];  
 }  
 }  
  
 }  
 }  
  
 **private double**[] returnStroke() {  
 *//обратный ход* **x** = **new double**[**matrix**.**length**];  
 **for** (**int** i = 0; i < **matrix**.**length**; i++) {  
 **x**[i] = 0;  
 }  
 **for** (**int** i = **matrix**.**length** - 1; i >= 0; i--) {  
 **double** s = 0;  
 **for** (**int** j = i + 1; j < **matrix**.**length**; j++) {  
 s += **matrix**[i][j] \* **x**[j];  
 }  
 **x**[i] = (**matrix**[i][**matrix**[0].**length** - 1] - s) / **matrix**[i][i];  
 }  
  
 **return x**;  
 }  
  
  
 *//выбор макс элемента по столбцу* **private int** maxElem(**double**[][] matrix, **int** column) {  
 **double** max = 0;  
 **int** maxElemInColumn = 0;  
 **for** (**int** i = column; i < matrix.**length**; i++) { *//начинаем с column, чтобы менять строку местами только с нижними строками* **if** (Math.*abs*(matrix[i][column]) - max > 0) {  
 max = Math.*abs*(matrix[i][column]);  
 maxElemInColumn = i;  
 }  
 }  
 **return** maxElemInColumn;  
 }  
  
 **private double**[][] changeRows(**double**[][] matrix, **int** i, **int** maxElemInColumn) {  
  
 **double**[] savingRow = **new double**[matrix[0].**length**];  
 **for** (**int** j = 0; j < matrix[0].**length**; j++) {  
 savingRow[j] = matrix[i][j];  
 matrix[i][j] = matrix[maxElemInColumn][j];  
 matrix[maxElemInColumn][j] = savingRow[j];  
 }  
  
 **return** matrix;  
  
 }  
  
 **public double**[] getAnswer() {  
 forwardStroke();  
 **return** returnStroke();  
 }  
}

**Построение графика**

**public class** LineChart **extends** ApplicationFrame {  
 **private static final long *serialVersionUID*** = 1L;  
 **private** String **description**;  
 **private double**[] **f**;  
 **private double**[] **x**;  
 **private double**[] **y**;  
 XYSeriesCollection **seriesNewDataset** = **new** XYSeriesCollection();  
 XYSeriesCollection **firstSeriesDataset** = **new** XYSeriesCollection();  
  
 **public** LineChart(**final** String title, **double**[] f, **double**[] x,**double**[] y,String description) {  
 **super**(title);  
 **this**.**description**=description;  
 **this**.**f**=f;  
 **this**.**x**=x;  
 **this**.**y**=y;  
 start();  
 }  
  
 **private void** start() {  
 JFreeChart chart = createChart();  
 ChartPanel chartPanel = **new** ChartPanel(chart);  
 chartPanel.setPreferredSize(**new** Dimension(500, 400));  
 setContentPane(chartPanel);  
 }  
  
  
 **private** JFreeChart createChart() {  
 **final** JFreeChart chart = ChartFactory.*createXYLineChart*(  
 **description**,  
 **null**, *// x axis label* **null**, *// y axis label* **null**, *// data* PlotOrientation.***VERTICAL***,  
 **true**, *// include legend* **false**, *// tooltips* **false** *// urls* );  
  
 chart.setBackgroundPaint(Color.***white***);  
  
 **final** XYPlot plot = chart.getXYPlot();  
 plot.setBackgroundPaint(**new** Color(159, 190, 237));  
  
 plot.setDomainGridlinePaint(Color.***gray***);*//сетка* plot.setRangeGridlinePaint(Color.***gray***);  
  
 *// Определение отступа меток делений* plot.setAxisOffset(**new** RectangleInsets(1.0, 1.0, 1.0, 1.0));  
  
 *// Скрытие осевых линий и меток делений* ValueAxis axis = plot.getDomainAxis();  
 axis.setAxisLineVisible(**false**); *// осевая линия  
  
 // Настройка NumberAxis* **final** NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();  
 rangeAxis.setAxisLineVisible(**false**);  
 rangeAxis.setStandardTickUnits(NumberAxis.*createIntegerTickUnits*());  
  
 XYSeries seriesNew = **new** XYSeries(**description**);  
 XYSeries firstSeries = **new** XYSeries(**"Табличная функция"**);  
 **for** (**int** i = 0; i < **f**.**length**; i++) {  
 seriesNew.add(**x**[i], **f**[i]);*// График апроксимирующей функции* firstSeries.add(**x**[i],**y**[i]);  
 }  
 **seriesNewDataset**.addSeries(seriesNew);  
 XYDataset newDataset = **seriesNewDataset**;  
  
 *// Точки набора данных* **firstSeriesDataset**.addSeries(firstSeries);  
 XYDataset firstDataset = **firstSeriesDataset**;  
  
 *// Настройка XYSplineRenderer  
 // Precision: the number of line segments between 2 points [default: 5]* XYSplineRenderer r0 = **new** XYSplineRenderer();  
 *//характеристики графика аппроксимации* r0.setPrecision(8);  
 r0.setSeriesShapesVisible(0, **true**); *//точки на графике* r0.setSeriesPaint(0, Color.***blue***);  
 XYSplineRenderer r1 = **new** XYSplineRenderer();  
 *//характеристики табличной функции* r1.setSeriesPaint(0, Color.***red***);  
 r1.setSeriesShapesVisible(0, **true**);  
 r1.setSeriesLinesVisible (0, **false**);  
  
 plot.setDataset(0, newDataset);  
 plot.setDataset(1, firstDataset);  
  
 *// Подключение Spline Renderer к набору данных* plot.setRenderer(0, r0);  
 plot.setRenderer(1, r1);  
  
 **return** chart;  
 }  
  
}

**Чтение и запись данных(как в файл, так и в консоль)**

**import** java.io.\*;  
  
**public class** DataReceiver {  
  
 BufferedReader **in** = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
  
 **public double**[][] receiveData() {  
 **while** (**true**) {  
 System.***out***.println(**"Нахождение функции, являющейся наилучшим приближением заданной табличной\n функции по методу наименьших квадратов"**);  
 System.***out***.println(**"Для ввода данных с консоли нажмите 1, для ввода из файла нажмите 2"**);  
 **try** {  
 **switch** (**in**.readLine()) {  
 **case "1"**:  
 **return** readFromConsole();  
 **case "2"**:  
 **return** readFromFile();  
 **default**:  
 System.***out***.println(**"Произошла ошибка, повторите ввод еще раз"**);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 System.***out***.println(**"Неверный ввод!"**);  
 }  
 }  
  
 }  
  
 **public void** answer(String s) {  
 **while** (**true**) {  
 System.***out***.println(**"Для вывода ответа в консоль нажмите 1\n"** + **"Для вывода в файл нажмите 2"**);  
 **try** {  
 **switch** (**in**.readLine()) {  
 **case "1"**:  
 System.***out***.println(s);  
 **return**;  
 **case "2"**:  
 writeToFile(s);  
 **return**;  
  
 **default**:  
 System.***out***.println(**"Произошла ошибка, повторите ввод еще раз"**);  
 }  
 } **catch** (IOException e) {  
 System.***out***.println(**"Неверный ввод!"**);  
 }  
 }  
 }  
  
 **private void** writeToFile(String s) {  
 **try** (FileWriter writer = **new** FileWriter(**"answer.txt"**, **false**)) {  
 writer.write(s);  
 writer.flush();  
 } **catch** (IOException ex) {  
 System.***out***.println(**"Произошла ошибка записи в файл"**);  
 }  
 }  
  
 **private double**[][] readFromConsole() {  
 **while** (**true**) {  
 **try** {  
 System.***out***.println(**"Для записи дробной части используйте точку. Чтобы разделить аргументы используйте пробел. Сначала строка X, затем Y"**);  
 **return** getTableXY(**in**);  
 } **catch** (IOException e) {  
 System.***out***.println(**"Неверный ввод!"**);  
 }  
 }  
 }  
  
 **private double**[][] readFromFile() {  
 **while** (**true**) {  
 **try** {  
 System.***out***.println(**"Введите полный путь к файлу, в котором хранится таблица y=f(x)"**);  
 System.***out***.println(**"Для записи дробной части используйте точку. Чтобы разделить аргументы используйте пробел"**);  
 String path = **in**.readLine();  
 BufferedReader reader = **new** BufferedReader(**new** FileReader(**new** File(path)));  
 **return** getTableXY(reader);  
 } **catch** (FileNotFoundException e) {  
 System.***out***.println(**"Файл не найден!"**);  
 } **catch** (ArrayIndexOutOfBoundsException e) {  
 System.***out***.println(**"Данные некорректны.Проверьте содержимое файла!"**);  
 } **catch** (IOException e) {  
 System.***out***.println(**"Произошла ошибка загрузки данных из файла!"**);  
 }  
 }  
 }  
  
 **private double**[][] getTableXY(BufferedReader reader) **throws** IOException {  
 **double**[][] table = **new double**[2][];  
 **int** length = -1;  
 **for** (**int** i = 0; i < 2; i++) {  
 String line = reader.readLine();  
 String[] parts = line.trim().replace(**","**, **"."**).split(**" "**);  
 **if** (length == -1 || length == parts.**length**) {  
 table[i] = **new double**[parts.**length**];  
 **for** (**int** j = 0; j < parts.**length**; j++) {  
 table[i][j] = Double.*parseDouble*(parts[j].trim());  
 }  
 length = parts.**length**;  
 } **else** {  
 **throw new** IOException(**"разная длина..."**);  
 }  
 }  
 **return** table;  
 }  
}

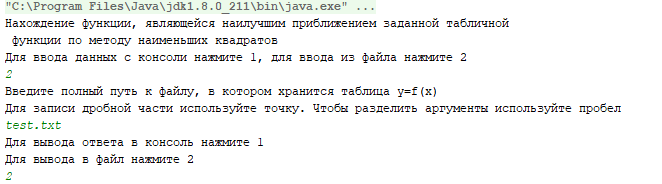
**public class** TheBestApproximatingFunction {  
 DataReceiver **dataReceiver** = **new** DataReceiver();  
  
 Function[] **functions** = **new** Function[5];  
 String[][] **answer**;  
 **double**[][] **table**;  
 **int n**;  
 StringBuffer **buffAnswer** = **new** StringBuffer();  
 String[] **title**;  
 String[] **functionsDescription**;  
  
  
 TheBestApproximatingFunction() {  
 init();  
 }  
  
 **private void** init() {  
 **table** = **dataReceiver**.receiveData();  
 }  
  
  
 **void** getTheBestApproximatingFunction() {  
 createAnswerTable();  
 **dataReceiver**.answer(*valueOf*(**buffAnswer**));  
 printGraphics();  
  
 }  
  
  
 **private void** createDescriptions() {  
 **title** = **new** String[]{**"Линейная функция"**, **"Полиноминальная функция"**, **"Экспоненциальная функция"**, **"Логарифмическая функция"**, **"Степенная функция"**};  
 **functionsDescription** = **new** String[]{String.*format*(**"F=%.4fx+%.4f"**, **functions**[0].getA(), **functions**[0].getB()),  
 String.*format*(**"F=%.4fx^2+%.4fx+%.4f"**, **functions**[1].getA(), **functions**[1].getB(), **functions**[1].getC()),  
 String.*format*(**"F=%.4fe^(%.4fx)"**, **functions**[2].getA(), **functions**[2].getB()),  
 String.*format*(**"F=%.4flnx+%.4f"**, **functions**[3].getA(), **functions**[3].getB()),  
 String.*format*(**"F=%.4fx^%.4f"**, **functions**[4].getA(), **functions**[4].getB())};  
 }  
  
 **private void** printGraphics() {  
 Double[] horizontalPercent = {0.0, 1.0, 1.0, 0.0, 0.5};  
 Double[] verticalPercent = {0.0, 0.0, 1.1, 1.1, 0.5};  
 **for** (**int** i = 0; i < 5; i++) {  
 LineChart lineChart0 = **new** LineChart(**title**[i], **functions**[i].getF()  
 , **table**[0], **table**[1], **functionsDescription**[i]);  
 lineChart0.pack();  
 RefineryUtilities.*positionFrameOnScreen*(lineChart0, horizontalPercent[i], verticalPercent[i]);  
 lineChart0.setVisible(**true**);  
 }  
  
 }  
  
 **private void** createAnswerTable() {  
 **n** = **table**[0].**length**;*//количество точек* **answer** = **new** String[5 + **n**][7];  
 **for** (**int** i = 0; i < **n**; i++) {  
 **answer**[i][0] = String.*format*(**"|%10.2f "**, **table**[0][i]); *//столбец Х в итоговой таблице* **answer**[i][1] = String.*format*(**"|%10.2f "**, **table**[1][i]); *// столбец Y в итоговой таблице* }  
 **answer**[**n**][0] = String.*format*(**"| S "**);  
 **answer**[**n**][1] = String.*format*(**"| = "**);  
 **answer**[**n** + 1][0] = String.*format*(**"| δ "**);  
 **answer**[**n** + 1][1] = String.*format*(**"| = "**);  
 **answer**[**n** + 2][0] = String.*format*(**"| A "**);  
 **answer**[**n** + 2][1] = String.*format*(**"| = "**);  
 **answer**[**n** + 3][0] = String.*format*(**"| B "**);  
 **answer**[**n** + 3][1] = String.*format*(**"| = "**);  
 **answer**[**n** + 4][0] = String.*format*(**"| C "**);  
 **answer**[**n** + 4][1] = String.*format*(**"| = "**);  
 **functions**[0] = **new** LinearFunction(**table**);  
 **functions**[1] = **new** QuadraticFunction(**table**);  
 **functions**[2] = **new** ExponentialFunction(**table**);  
 **functions**[3] = **new** LogarithmicFunction(**table**);  
 **functions**[4] = **new** PowerFunction(**table**);  
 **for** (**int** j = 2; j < 7; j++) {  
 **double**[] f = **functions**[j - 2].getF();  
 **for** (**int** i = 0; i < **n**; i++) {  
 **answer**[i][j] = String.*format*(**"|%10.2f "**, f[i]); *//столбец каждой из функций в итоговой таблице* }  
 **answer**[**n**][j] = String.*format*(**"|%10.2f "**, **functions**[j - 2].getS());  
 **answer**[**n** + 1][j] = String.*format*(**"|%10.2f "**, **functions**[j - 2].getδ());  
 **answer**[**n** + 2][j] = String.*format*(**"|%10.2f "**, **functions**[j - 2].getA());  
 **answer**[**n** + 3][j] = String.*format*(**"|%10.2f "**, **functions**[j - 2].getB());  
 }  
 **answer**[**n** + 4][2] = String.*format*(**"|-----------"**);  
 **answer**[**n** + 4][3] = String.*format*(**"|%10.2f "**, **functions**[1].getC());  
 **for** (**int** i = 4; i < 7; i++) {  
 **answer**[**n** + 4][i] = String.*format*(**"|-----------"**);  
 }  
  
 **buffAnswer**.append(**"| Вид f(x) | Линейная | Полином. | Экспонен. | Логарифм. | Степенная |\n"**)  
 .append(**"-------------------------------------------------------------------------------------\n"**)  
 .append(**"| X | Y | F=ax+b |F=ax^2+bx+c| F=ae^(bx) | F=alnx+b | F=ax^b |\n"**)  
 .append(**"-------------------------------------------------------------------------------------\n"**);  
  
 **for** (**int** i = 0; i < 5 + **table**[0].**length**; i++) {  
 **for** (**int** j = 0; j < 7; j++) {  
 **buffAnswer**.append(**answer**[i][j]);  
 }  
 **buffAnswer**.append(**"|\n"**);  
 **buffAnswer**.append(**"-------------------------------------------------------------------------------------\n"**);  
 }  
  
 createDescriptions();  
 *//поиск лучшей аппроксимирующей функции* **double** min = 1000000;  
 **int** k = 0;  
 **for** (**int** i = 0; i < 5; i++) {  
 **if** (**functions**[i].getδ() < min) {  
 min = **functions**[i].getδ();  
 k = i;  
 }  
 }  
 **buffAnswer**.append(**title**[k] + **" "** + **functionsDescription**[k] + **" является наилучшим приближением заданной табличной функции"**);  
 }  
}

1. Анализ результатов работы: апробация и тестирование.

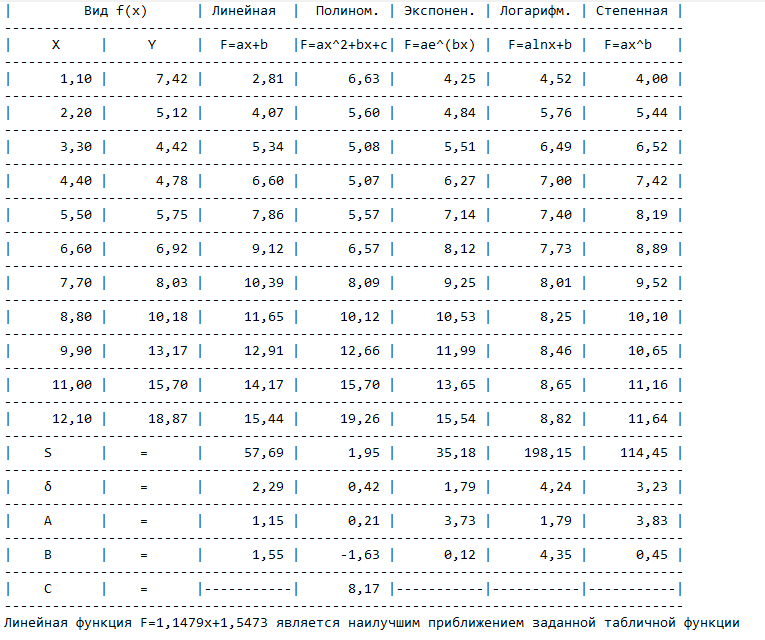
Входные данные



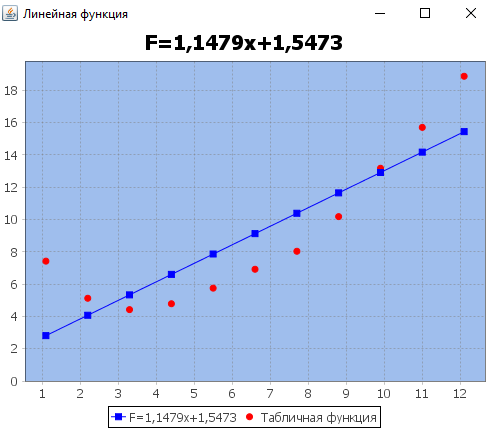
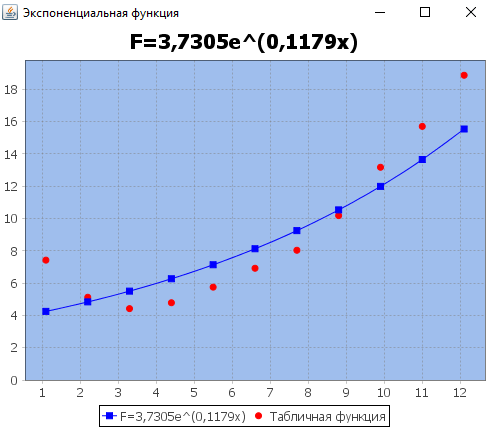
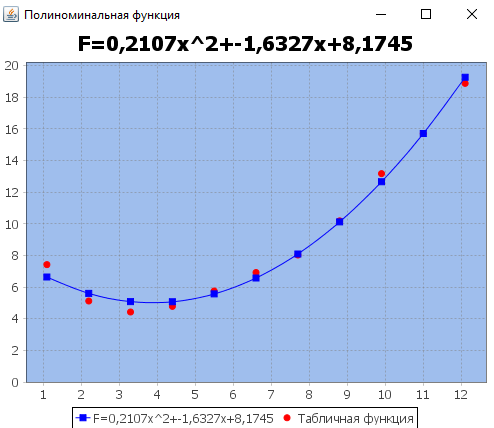
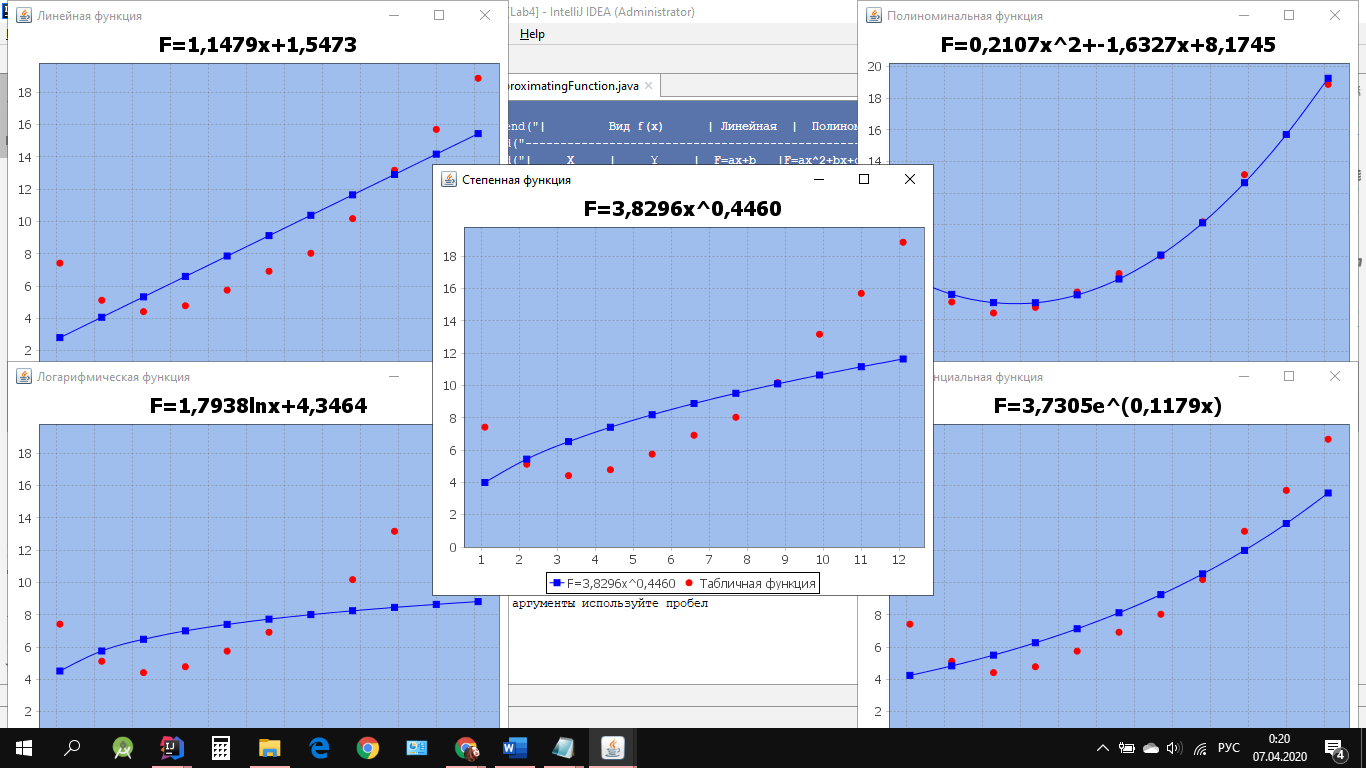
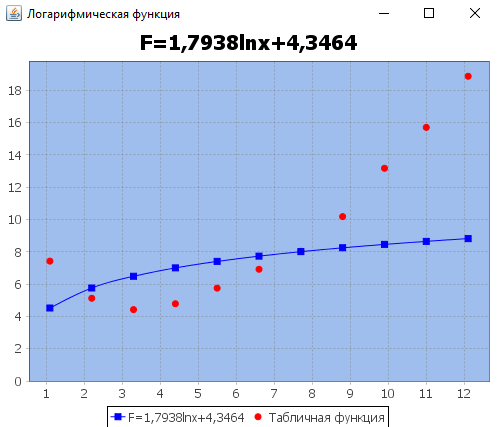
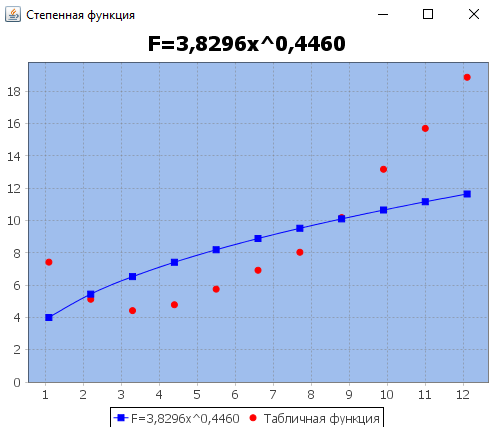
Выходные данные



Содержимое файла



Графики функций

Вывод: В ходе лабораторной работы я научилась находить функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.