Федеральное государственное автономное
образовательное учреждение высшего образования
«Научно-образовательная корпорация ИТМО»

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия

Отчёт по лабораторной работе №4

По дисциплине «Вычислительная математика» (4 семестр)

Студент:
Дениченко Александр P3212
Практик:
Наумова Надежда Александровна

Санкт-Петербург
2024 г.

1 Цель работы

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

Вариант – 8

2 Вычислительная часть

1. Сформировать таблицу табулирования заданной функции на указанном интервале (см. табл. 1)
2. Построить линейное и квадратичное приближения по 11 точкам заданного интервала;
3. Найти среднеквадратические отклонения для каждой аппроксимирующей функции. Ответы дать с тремя знаками после запятой;
4. Выбрать наилучшее приближение;
5. Построить графики заданной функции, а также полученные линейное и квадратичное приближения;

Функция по варианту:

$$y = \frac{3x}{x^4 + 8},\ x \in [-2,\ 0],\ h = 0.2$$

График функции:

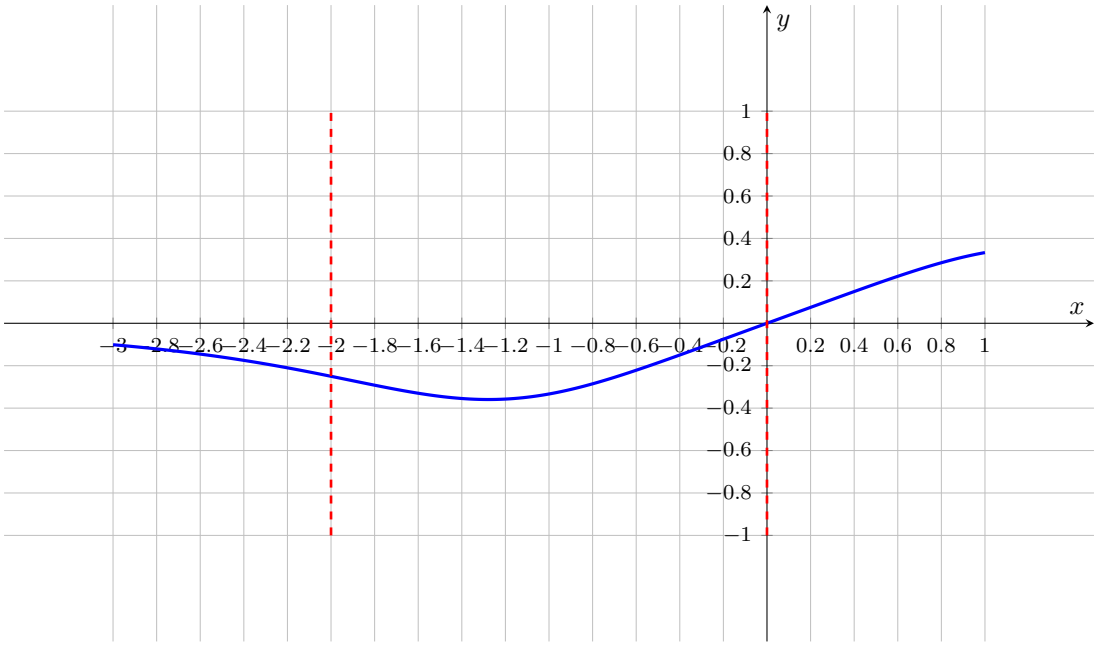


Рис. 1: График функции $y = \frac{3x}{x^4+8}$

Таблица табулирования функции:

x	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y	-0.25	-0.292	-0.330	-0.355	-0.357	-0.333	-0.285	-0.221	-0.150	-0.075	0.0

Таблица 1: Трассировка

3 Линейная аппроксимация

Рассмотрим в качестве эмпирической формулы линейную функцию:

$$\phi(a, x, b) = ax + b$$

Сумма квадратов отклонений запишется следующим образом:

$$S = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (\phi(x_i) - y_i)^2 = \sum_{i=1}^{n} (ax_i^2 + b - y_i)^2 \ \ -> \ \ min$$

Для нахождения a и b необходимо найти минимум функции S. Необходимое условие существования минимума для функции S:

$$\begin{cases} \frac{\partial S}{\partial a} = 0 \\ \frac{\partial S}{\partial b} = 0 \end{cases} => \begin{cases} 2\sum_{i=1}^{n}(ax_i + b - y_i)x_i = 0 \\ 2\sum_{i=1}^{n}(ax_i + b - y_i) = 0 \end{cases} => \begin{cases} a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \\ a\sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i \end{cases}$$

Проведём подсчёты:

$$\sum_{i=1}^{n} x_i = (-2.0) + (-1.8) + (-1.6) + (-1.4) + (-1.2) + (-1.0) + (-0.8) + (-0.6) + (-0.4) + (-0.2) + 0.0 = -11.0$$

$$\sum_{i=1}^{n} x_i^2 = (-2.0)^2 + (-1.8)^2 + (-1.6)^2 + (-1.4)^2 + (-1.2)^2 + (-1.0)^2 + (-0.8)^2 + (-0.6)^2 + (-0.4)^2 + (-0.2)^2 + 0.0^2 = 15.4$$

$$\sum_{i=1}^{n} y_i = (-0.25)+(-0.292)+(-0.330)+(-0.355)+(-0.357)+(-0.333)+(-0.285)+(-0.221)+(-0.150)+(-0.075)+0.0 = -2.648$$

$$\sum_{i=1}^{n} x_i y_i = (-2.0)\cdot(-0.25)+(-1.8)\cdot(-0.292)+(-1.6)\cdot(-0.330)+(-1.4)\cdot(-0.355)+(-1.2)\cdot(-0.357)+(-1.0)\cdot(-0.333)+(-0.8)\cdot$$

$$\cdot(-0.285) + (-0.6)\cdot(-0.221) + (-0.4)\cdot(-0.150) + (-0.2)\cdot(-0.075) + (0.0)\cdot(0.0) = 3.248$$

Получим систему:

$$\begin{cases} 15.4a + (-11.0)b = 3.248 \\ -11.0a + 11b = -2.648 \end{cases}$$

из которой находим:

$$\Delta = 15.4 \cdot 11 - 11^2 = 48.4$$

$$\Delta_1 = 3.248 \cdot 11 - 11 \cdot 2.648 = 6.6$$

$$\Delta_2 = 15.4 \cdot (-2.648) - (-11.0) \cdot 3.248 = -5.051$$

Подставим найденные значения:

$$a = \frac{6.6}{48.4} \approx 0.136; \quad b = \frac{-5.0512}{48.4} \approx -0.104$$

Тогда аппроксимирующая функция будет иметь вид:

$$\phi(x) = 0.136x - 0.104$$

Дополним таблицу:

x	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y	-0.25	-0.292	-0.330	-0.355	-0.357	-0.333	-0.285	-0.221	-0.150	-0.075	0.0
$\phi(x)$	-0.377	-0.350	-0.323	-0.295	-0.268	-0.241	-0.214	-0.186	-0.159	-0.132	-0.105
$\lvert y - \phi \rvert$	0.127	0.058	0.007	0.060	0.089	0.092	0.071	0.035	0.009	0.057	0.105
$\lvert y - \phi \rvert^2$	0.016	0.003	0	0.003	0.007	0.008	0.005	0.001	0	0.003	0.011

Таблица 2: Линейная аппроксимация

Среднеквадратические отклонения по формуле:

$$\delta = \sqrt{\frac{\sum_{i=1}^{n}(\phi(x_i) - y_i)^2}{n}} = 0.0739$$

4 Квадратичная аппроксимация

Рассмотрим в качестве эмпирической формулы квадратичную функцию:

$$\phi(x, a_0, a_1, a_2) = a_0 + a_1 x + a_2 x^2$$

Сумма квадратов отклонений записывается следующим образом:

$$S = S(a_0, a_1, a_2) = \sum_{i=1}^{n} (a_0 + a_1 x_i + a_2 x_i^2 - y_i)^2 \to \min$$

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

$$\begin{cases} \frac{\partial S}{\partial a_0} = 2\sum(a_2 x_i^2 + a_1 x_i + a_0 - y_i) = 0 \\ \frac{\partial S}{\partial a_1} = 2\sum(a_2 x_i^2 + a_1 x_i + a_0 - y_i)x_i = 0 \\ \frac{\partial S}{\partial a_2} = 2\sum(a_2 x_i^2 + a_1 x_i + a_0 - y_i)x_i^2 = 0 \end{cases}$$

$$\begin{cases} a_0 n + a_1 \sum x_i + a_2 \sum x_i^2 = \sum y_i \\ a_0 \sum x_i + a_1 \sum x_i^2 + a_2 \sum x_i^3 = \sum x_i y_i \\ a_0 \sum x_i^2 + a_1 \sum x_i^3 + a_2 \sum x_i^4 = \sum x_i^2 y_i \end{cases}$$

$$\sum_{i=1}^{n} x_i = -11.0 \quad \sum_{i=1}^{n} x_i^2 = 15.4 \quad \sum_{i=1}^{n} y_i = -2.648 \quad \sum_{i=1}^{n} x_i y_i = 3.248$$

$$\sum_{i=1}^{n} x_i^3 = -24.2 \quad \sum_{i=1}^{n} x_i^4 = 40.5328 \quad \sum_{i=1}^{n} x_i^2 y_i = -4.62272$$

Подставим значения:

$$\begin{cases} 11 \cdot a_0 + (-11.0) \cdot a_1 + 15.4 \cdot a_2 = -2.648 \\ -11.0 \cdot a_0 + 15.4 \cdot a_1 + (-24.2) \cdot a_2 = 3.248 \\ 15.4 \cdot a_0 + (-24.2) \cdot a_1 + 40.5328 \cdot a_2 = -4.62272 \end{cases}$$

Решение системы уравнений:

$$\begin{cases} a_0 = 0.02 \\ a_1 = 0.55 \\ a_2 = 0.207 \end{cases}$$

Получим формулу для квадратичной аппроксимации

$$\phi(x) = 0.02 + 0.55 \cdot x + 0.207 \cdot x^2$$

x	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y	-0.25	-0.292	-0.330	-0.355	-0.357	-0.333	-0.285	-0.221	-0.150	-0.075	0.0
$\phi(x)$	-0.252	-0.299	-0.330	-0.344	-0.341	-0.323	-0.287	-0.235	-0.166	-0.081	0.02
$\lvert y - \phi \rvert$	0.002	0.007	0	0.01072	0.015	0.01	0.002	0.014	0.016	0.006	0.02
$\lvert y - \phi \rvert^2 \cdot 10^{-4}$	0.04	5	0.006	1.14	2.27	1	0.063	2.09	2.84	0.451	4

Таблица 3: Квадратичная аппроксимация

Среднеквадратические отклонения по формуле:

$$\delta = \sqrt{\frac{\sum_{i=1}^{n} (\phi(x_i) - y_i)^2}{n}} = 0.011$$

5 Построения результатов

Исходная:

$$y = \frac{3x}{x^4 + 8}$$

Линейная:

$$\zeta(x) = 0.136x - 0.104$$

Квадратичная:

$$\aleph(x) = 0.02 + 0.55 \cdot x + 0.207 \cdot x^2$$

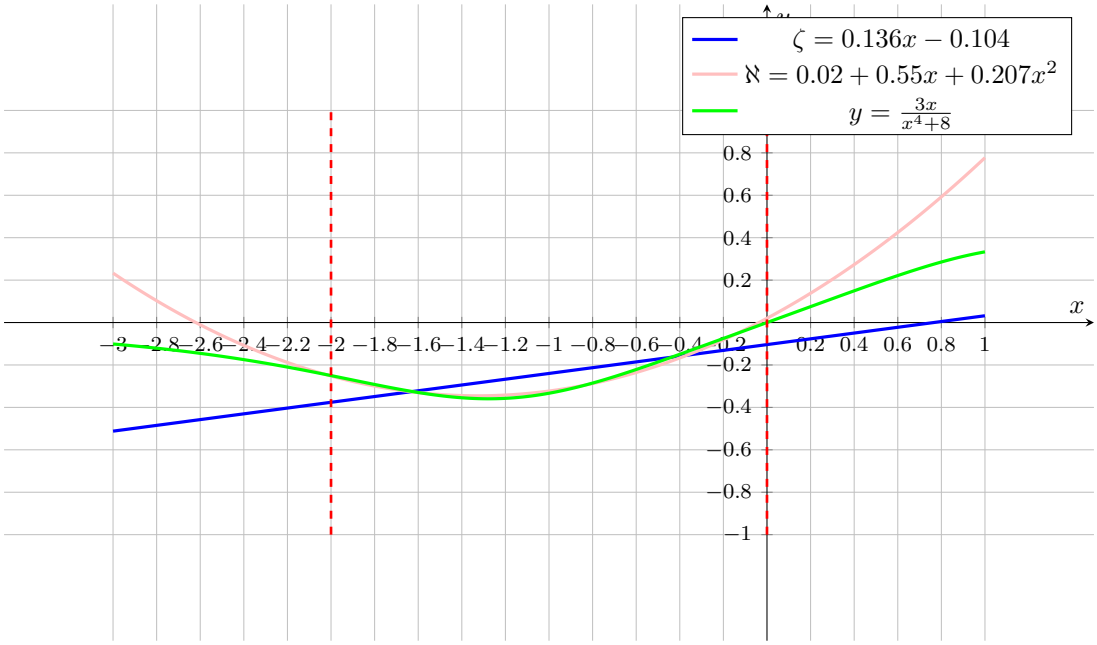


Рис. 2: Графики для сравнения

Самое хорошее приближении получилось квадратичное, оно наиболее чётко совпадает с графиком исходной функции в системе декартовых координат.

6 Машинная реализация

Листинг 1: Кубическая аппроксимация

```
1  package com.example.web4.math.approx;
2
3  import com.example.web4.dto.PointDto;
4  import com.example.web4.dto.RequestFuncUser;
5  import lombok.Getter;
6  import org.apache.commons.math3.linear.*;
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9
10 import java.util.ArrayList;
11 @Getter
12
13 public class CubicApprox extends Method{
```

```
14          private Double a;
15          private Double b;
16          private Double c;
17          private Double d;
18
19          private static final Logger logger = LoggerFactory.getLogger(CubicApprox.class);
20
21          public CubicApprox(RequestFuncUser requestFuncUser) {
22              super(requestFuncUser);
23          }
24
25          private double f(double x) {
26              return a * Math.pow(x, 3) + b * Math.pow(x, 2) + c * x + d;
27          }
28
29          @Override
30          public void calculate() {
31              int n = requestFuncUser.getPoints().size();
32              double[][] matrixData = new double[4][4];
33              double[] vector = new double[4];
34
35              for (PointDto point : requestFuncUser.getPoints()) {
36                  double x = point.getX();
37                  double y = point.getY();
38                  double x2 = Math.pow(x, 2);
39                  double x3 = Math.pow(x, 3);
40                  double x4 = Math.pow(x, 4);
41                  double x5 = Math.pow(x, 5);
42                  double x6 = Math.pow(x, 6);
43
44                  matrixData[0][0] += x6;
45                  matrixData[0][1] += x5;
46                  matrixData[0][2] += x4;
47                  matrixData[0][3] += x3;
48                  matrixData[1][0] += x5;
49                  matrixData[1][1] += x4;
50                  matrixData[1][2] += x3;
51                  matrixData[1][3] += x2;
52                  matrixData[2][0] += x4;
53                  matrixData[2][1] += x3;
54                  matrixData[2][2] += x2;
55                  matrixData[2][3] += x;
56                  matrixData[3][0] += x3;
57                  matrixData[3][1] += x2;
58                  matrixData[3][2] += x;
59                  matrixData[3][3] += 1;
60                  vector[0] += y * x3;
61                  vector[1] += y * x2;
62                  vector[2] += y * x;
63                  vector[3] += y;
64              }
65
66              RealMatrix coefficients = new Array2DRowRealMatrix(matrixData, false);
67              DecompositionSolver solver = new LUDecomposition(coefficients).getSolver();
68
69              RealVector constants = new ArrayRealVector(vector, false);
```

```
70              RealVector solution = solver.solve(constants);
71
72              a = solution.getEntry(0);
73              b = solution.getEntry(1);
74              c = solution.getEntry(2);
75              d = solution.getEntry(3);
76              logger.info("cubic: a = "+a+"; b = "+b+"; c = "+c+"; d = "+d);
77
78              for (PointDto pointDto : requestFuncUser.getPoints()){
79                  ArrayList<Double> tmp = new ArrayList<>();
80                  tmp.add(pointDto.getX());
81                  tmp.add(pointDto.getY());
82                  tmp.add(f(pointDto.getX()));
83                  tmp.add(f(pointDto.getX()) - pointDto.getY());
84                  table.add(tmp);
85                  S += Math.pow(f(pointDto.getX()) - pointDto.getY(), 2);
86              }
87              sko = Math.sqrt(S/n);
88
89              double meanY = requestFuncUser.getPoints().stream().mapToDouble(PointDto::getY).
                    average().orElse(0);
90              double ssTot = 0;
91              double ssRes = 0;
92
93              for (PointDto point : requestFuncUser.getPoints()) {
94                  double fi = f(point.getX());
95                  ssTot += Math.pow(point.getY() - meanY, 2);
96                  ssRes += Math.pow(point.getY() - fi, 2);
97              }
98              determ = 1 - (ssRes / ssTot);
99
100             korrelPirs = Math.sqrt(determ);
101         }
102         @Override
103         protected String getStringFun() {
104 //          a * Math.pow(x, 3) + b * Math.pow(x, 2) + c * x + d;
105             return "\\phi (x)="+formatScientificNotation(a.toString())+"\\cdot x^3 "+"+
                    formatScientificNotation(b.toString())+"\\cdot x^2+"+formatScientificNotation
                    (c.toString())+"\\cdot x "+formatScientificNotation(d.toString());
106         }
107     }
```

Листинг 2: Експоненциальная аппроксимация

```
1   package com.example.web4.math.approx;
2
3   import com.example.web4.dto.PointDto;
4   import com.example.web4.dto.RequestFuncUser;
5   import lombok.Getter;
6   import org.apache.commons.math3.linear.*;
7   import org.slf4j.Logger;
8   import org.slf4j.LoggerFactory;
9   import java.util.ArrayList;
10  @Getter
11  public class ExpApprox extends Method {
12      private Double a;
13      private Double b;
```

```
14
15          private static final Logger logger = LoggerFactory.getLogger(ExpApprox.class);
16
17          public ExpApprox(RequestFuncUser requestFuncUser) {
18              super(requestFuncUser);
19          }
20
21          private double f(double x) {
22              return a * Math.exp(b * x);
23          }
24
25          @Override
26          public void calculate() {
27              int n = requestFuncUser.getPoints().size();
28              double sumX = 0;
29              double sumYln = 0;
30              double sumX2 = 0;
31              double sumXYln = 0;
32
33              for (PointDto point : requestFuncUser.getPoints()) {
34                  double x = point.getX();
35                  double y = Math.log(point.getY());
36
37                  sumX += x;
38                  sumYln += y;
39                  sumX2 += x * x;
40                  sumXYln += x * y;
41              }
42
43              double b = (n * sumXYln - sumX * sumYln) / (n * sumX2 - sumX * sumX);
44              double aLn = (sumYln - b * sumX) / n; // ln(a)
45
46              this.a = Math.exp(aLn);
47              this.b = b;
48              logger.info("exp: a = "+a+"; b = "+b);
49
50              for (PointDto point : requestFuncUser.getPoints()) {
51                  double fi = f(point.getX());
52                  double eps = fi - point.getY();
53                  ArrayList<Double> row = new ArrayList<>();
54                  row.add(point.getX());
55                  row.add(point.getY());
56                  row.add(fi);
57                  row.add(eps);
58                  table.add(row);
59                  S += Math.pow(eps, 2);
60              }
61              sko = Math.sqrt(S/n);
62
63              double meanY = requestFuncUser.getPoints().stream().mapToDouble(PointDto::getY).
                    average().orElse(0);
64              double ssTot = 0;
65              double ssRes = 0;
66
67              for (PointDto point : requestFuncUser.getPoints()) {
68                  double fi = f(point.getX());
```

```
69                  ssTot += Math.pow(point.getY() - meanY, 2);
70                  ssRes += Math.pow(point.getY() - fi, 2);
71              }
72              determ = 1 - (ssRes / ssTot);
73
74              korrelPirs = Math.sqrt(determ);
75          }
76
77          @Override
78          protected String getStringFun() {
79      //          a * Math.exp(b * x);
80              return "\\phi (x)="+formatScientificNotation(a.toString())+"\\cdot e^{"+
                    formatScientificNotation(b.toString())+"\\cdot x}";
81          }
82      }
```

Листинг 3: Линейная аппроксимация

```
1       package com.example.web4.math.approx;
2
3       import com.example.web4.dto.PointDto;
4       import com.example.web4.dto.RequestFuncUser;
5
6       import java.util.ArrayList;
7
8       import lombok.Getter;
9       import org.slf4j.Logger;
10      import org.slf4j.LoggerFactory;
11      @Getter
12      public class LinApprox extends Method {
13          private Double a;
14          private Double b;
15
16          private static final Logger logger = LoggerFactory.getLogger(LinApprox.class);
17
18          public LinApprox(RequestFuncUser requestFuncUser) {
19              super(requestFuncUser);
20          }
21          private double f(double x){
22              return a*x+b;
23          }
24
25          @Override
26          public void calculate(){
27              double sumX = 0;
28              double sumXX = 0;
29              double sumY = 0;
30              double sumXY = 0;
31              double sumYY = 0;
32              int n = requestFuncUser.getSliderValue();
33              for (PointDto pointDto : requestFuncUser.getPoints()){
34                  sumX += pointDto.getX();
35                  sumXX += Math.pow(pointDto.getX(), 2);
36                  sumY += pointDto.getY();
37                  sumXY += pointDto.getX()*pointDto.getY();
38                  sumYY += pointDto.getY() * pointDto.getY();
39              }
```

```
40              a = (sumXY*n - sumX*sumY)/(sumXX*n - sumX*sumX);
41              b = (sumXX*sumY - sumX*sumXY)/(sumXX*n - sumX*sumX);
42
43              logger.info("lin: a = "+a+"; b = "+b);
44
45              for (PointDto pointDto : requestFuncUser.getPoints()){
46                  ArrayList<Double> tmp = new ArrayList<>();
47                  tmp.add(pointDto.getX());
48                  tmp.add(pointDto.getY());
49                  tmp.add(f(pointDto.getX()));
50                  tmp.add(f(pointDto.getX()) - pointDto.getY());
51                  table.add(tmp);
52                  S += Math.pow(f(pointDto.getX()) - pointDto.getY(), 2);
53              }
54
55              sko = Math.sqrt(S/n);
56
57              korrelPirs = (sumXY * n - sumX * sumY) / Math.sqrt((sumXX * n - sumX * sumX) * (
                    sumYY * n - sumY * sumY));
58              determ = Math.pow(korrelPirs ,2);
59          }
60
61
62          @Override
63          protected String getStringFun() {
64      //          a*x+b;
65              return "\\phi (x)="+formatScientificNotation(a.toString())+"\\cdot x+"+
                    formatScientificNotation(b.toString());
66          }
67      }
```

Листинг 4: Логарифмическая аппроксимация

```
1       package com.example.web4.math.approx;
2
3       import com.example.web4.dto.PointDto;
4       import com.example.web4.dto.RequestFuncUser;
5       import lombok.Getter;
6       import org.apache.commons.math3.linear.*;
7       import org.slf4j.Logger;
8       import org.slf4j.LoggerFactory;
9       import java.util.ArrayList;
10      @Getter
11      public class LogApprox extends Method {
12          private Double a;
13          private Double b;
14
15
16          private static final Logger logger = LoggerFactory.getLogger(LogApprox.class);
17
18          public LogApprox(RequestFuncUser requestFuncUser) {
19              super(requestFuncUser);
20          }
21
22          private double f(double x) {
23              return a * Math.log(x) + b;
24          }
```

```
25
26          @Override
27          public void calculate() {
28              int n = requestFuncUser.getPoints().size();
29              double sumLnX = 0;
30              double sumY = 0;
31              double sumLnX2 = 0;
32              double sumYLnX = 0;
33
34              for (PointDto point : requestFuncUser.getPoints()) {
35                  double x = point.getX();
36                  double y = point.getY();
37                  double lnX = Math.log(x);
38
39                  sumLnX += lnX;
40                  sumY += y;
41                  sumLnX2 += lnX * lnX;
42                  sumYLnX += y * lnX;
43              }
44
45              double b = (sumY * sumLnX2 - sumLnX * sumYLnX) / (n * sumLnX2 - sumLnX * sumLnX)
                    ;
46              double a = (n * sumYLnX - sumLnX * sumY) / (n * sumLnX2 - sumLnX * sumLnX);
47
48              this.a = a;
49              this.b = b;
50              logger.info("log: a = "+a+"; b = "+b);
51
52              for (PointDto point : requestFuncUser.getPoints()) {
53                  double fi = f(point.getX());
54                  double eps = fi - point.getY();
55                  ArrayList<Double> row = new ArrayList<>();
56                  row.add(point.getX());
57                  row.add(point.getY());
58                  row.add(fi);
59                  row.add(eps);
60                  table.add(row);
61                  S += Math.pow(eps, 2);
62              }
63              sko = Math.sqrt(S/n);
64
65              double meanY = requestFuncUser.getPoints().stream().mapToDouble(PointDto::getY).
                    average().orElse(0);
66              double ssTot = 0;
67              double ssRes = 0;
68
69              for (PointDto point : requestFuncUser.getPoints()) {
70                  double fi = f(point.getX());
71                  ssTot += Math.pow(point.getY() - meanY, 2);
72                  ssRes += Math.pow(point.getY() - fi, 2);
73              }
74
75              determ = 1 - (ssRes / ssTot);
76
77              korrelPirs = Math.sqrt(determ);
78          }
```

```
79
80
81          @Override
82          protected String getStringFun() {
83      //          a * Math.log(x) + b;
84              return "\\phi (x)="+formatScientificNotation(a.toString())+"\\cdot ln(x)+"+
                    formatScientificNotation(b.toString());
85          }
86      }
```

Листинг 5: Степенная аппроксимация

```
1       package com.example.web4.math.approx;
2
3       import com.example.web4.dto.PointDto;
4       import com.example.web4.dto.RequestFuncUser;
5       import lombok.Getter;
6       import org.slf4j.Logger;
7       import org.slf4j.LoggerFactory;
8       import java.util.ArrayList;
9       @Getter
10      public class PowerApprox extends Method {
11          private Double a;
12          private Double b;
13
14          private static final Logger logger = LoggerFactory.getLogger(PowerApprox.class);
15
16          public PowerApprox(RequestFuncUser requestFuncUser) {
17              super(requestFuncUser);
18          }
19
20          private double f(double x) {
21              return a * Math.pow(x, b);
22          }
23
24          @Override
25          public void calculate() {
26              int n = requestFuncUser.getPoints().size();
27              double sumLnX = 0;
28              double sumLnY = 0;
29              double sumLnX2 = 0;
30              double sumLnXLnY = 0;
31
32              for (PointDto point : requestFuncUser.getPoints()) {
33                  double lnX = Math.log(point.getX());
34                  double lnY = Math.log(point.getY());
35
36                  sumLnX += lnX;
37                  sumLnY += lnY;
38                  sumLnX2 += lnX * lnX;
39                  sumLnXLnY += lnX * lnY;
40              }
41
42              b = (n * sumLnXLnY - sumLnX * sumLnY) / (n * sumLnX2 - sumLnX * sumLnX);
43              double lnA = (sumLnY - b * sumLnX) / n;
44
45              this.a = Math.exp(lnA);
```

```
46              this.b = b;
47              logger.info("power: a = "+a+"; b = "+b);
48
49              for (PointDto point : requestFuncUser.getPoints()) {
50                  double fi = f(point.getX());
51                  double eps = fi - point.getY();
52                  ArrayList<Double> row = new ArrayList<>();
53                  row.add(point.getX());
54                  row.add(point.getY());
55                  row.add(fi);
56                  row.add(eps);
57                  table.add(row);
58                  S += Math.pow(eps, 2);
59              }
60              sko = Math.sqrt(S/n);
61
62              double meanY = requestFuncUser.getPoints().stream().mapToDouble(PointDto::getY).
                    average().orElse(0);
63              double ssTot = 0;
64              double ssRes = 0;
65
66              for (PointDto point : requestFuncUser.getPoints()) {
67                  double fi = f(point.getX());
68                  ssTot += Math.pow(point.getY() - meanY, 2);
69                  ssRes += Math.pow(point.getY() - fi, 2);
70              }
71
72              determ = 1 - (ssRes / ssTot);
73
74              korrelPirs = Math.sqrt(determ);
75          }
76
77
78          @Override
79          protected String getStringFun() {
80      //          a * Math.pow(x, b);
81              return "\\phi (x)="+formatScientificNotation(a.toString())+"\\cdot x^{"+
                    formatScientificNotation(b.toString())+"}";
82          }
83      }
```

Листинг 6: Квадратичная аппроксимация

```
1       package com.example.web4.math.approx;
2
3       import com.example.web4.dto.PointDto;
4       import com.example.web4.dto.RequestFuncUser;
5       import lombok.Getter;
6       import org.slf4j.Logger;
7       import org.slf4j.LoggerFactory;
8       import java.util.ArrayList;
9       import org.apache.commons.math3.linear.*;
10      @Getter
11      public class QuadApprox extends Method{
12          private Double a;
13          private Double b;
14          private Double c;
```

```
15
16          private static final Logger logger = LoggerFactory.getLogger(QuadApprox.class);
17
18          public QuadApprox(RequestFuncUser requestFuncUser) {
19              super(requestFuncUser);
20          }
21
22          private double f(double x) {
23              return a * Math.pow(x, 2) + b * x + c;
24          }
25
26          @Override
27          public void calculate() {
28              int n = requestFuncUser.getPoints().size();
29              double[] terms = new double[3];
30              double[][] matrixData = new double[3][3];
31              double[] vector = new double[3];
32
33              for (PointDto point : requestFuncUser.getPoints()) {
34                  double x = point.getX();
35                  double y = point.getY();
36                  terms[0] += Math.pow(x, 4);
37                  terms[1] += Math.pow(x, 3);
38                  terms[2] += Math.pow(x, 2);
39
40                  matrixData[0][0] += Math.pow(x, 4);
41                  matrixData[0][1] += Math.pow(x, 3);
42                  matrixData[0][2] += Math.pow(x, 2);
43                  matrixData[1][0] += Math.pow(x, 3);
44                  matrixData[1][1] += Math.pow(x, 2);
45                  matrixData[1][2] += x;
46                  matrixData[2][0] += Math.pow(x, 2);
47                  matrixData[2][1] += x;
48                  matrixData[2][2] += 1;
49
50                  vector[0] += x * x * y;
51                  vector[1] += x * y;
52                  vector[2] += y;
53              }
54
55              RealMatrix coefficients = new Array2DRowRealMatrix(matrixData, false);
56              DecompositionSolver solver = new LUDecomposition(coefficients).getSolver();
57
58              RealVector constants = new ArrayRealVector(vector, false);
59              RealVector solution = solver.solve(constants);
60
61              a = solution.getEntry(0);
62              b = solution.getEntry(1);
63              c = solution.getEntry(2);
64              logger.info("quad: a = "+a+"; b = "+b+"; c = "+c);
65
66              for (PointDto pointDto : requestFuncUser.getPoints()){
67                  ArrayList<Double> tmp = new ArrayList<>();
68                  tmp.add(pointDto.getX());
69                  tmp.add(pointDto.getY());
70                  tmp.add(f(pointDto.getX()));
```

```
71                  tmp.add(f(pointDto.getX()) - pointDto.getY());
72                  table.add(tmp);
73                  S += Math.pow(f(pointDto.getX()) - pointDto.getY(), 2);
74              }
75              sko = Math.sqrt(S/n);
76
77              double meanY = requestFuncUser.getPoints().stream().mapToDouble(PointDto::getY).
                    average().orElse(0);
78              double ssTot = 0;
79              double ssRes = 0;
80
81              for (PointDto point : requestFuncUser.getPoints()) {
82                  double fi = f(point.getX());
83                  ssTot += Math.pow(point.getY() - meanY, 2);
84                  ssRes += Math.pow(point.getY() - fi, 2);
85              }
86
87              determ = 1 - (ssRes / ssTot);
88
89              korrelPirs = Math.sqrt(determ);
90          }
91
92          @Override
93          protected String getStringFun() {
94      //          a * Math.pow(x, 2) + b * x + c;
95              return "\\phi (x)="+formatScientificNotation(a.toString())+"\\cdot x^2+"+
                    formatScientificNotation(b.toString())+"\\cdot x+"+formatScientificNotation(c
                    .toString());
96          }
97      }
```

7 Пример работы программы

Данные для ввода:

Вывод:

Кубическая аппроксимация

x	y	$\phi(x)$	ε
1.0	2.0	1.9999999999998979	$-1.021405182655144 \cdot 10^{-13}$
3.0	4.0	4.000000000000048	$4.796163466380676 \cdot 10^{-14}$
5.0	6.0	6.0000000000000835	$8.348877145181177 \cdot 10^{-14}$
7.0	8.0	8.000000000000048	$4.796163466380676 \cdot 10^{-14}$
9.0	10.0	9.999999999999982	$-1.7763568394002505 \cdot 10^{-14}$
11.0	12.0	11.999999999999934	$-6.572520305780927 \cdot 10^{-14}$
13.0	14.0	13.999999999999943	$-5.684341886080802 \cdot 10^{-14}$
15.0	16.0	16.00000000000006	$6.039613253960852 \cdot 10^{-14}$

Мера отклонения $S = 3.351791100193557 \cdot 10^{-26}$;

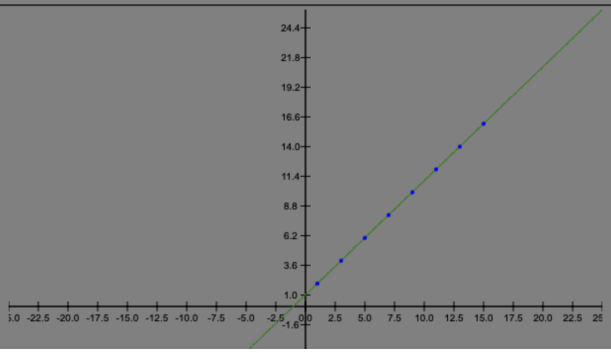
СКО $= 6.472819227540614 \cdot 10^{-14}$;

Корреляция $= 1.0$;

Коэффициент детерминации $= 1.0$.

Функция вида :

$$\phi(x) = 9.06450293866872 \cdot 10^{-16} \cdot x^3 - 2.2545242598976214 \cdot 10^{-14} \cdot x^2 + 1.0000000000001537 \cdot x + 0.9999999999997656$$



Экспоненциальная аппроксимация

x	y	$\phi(x)$	ε
1.0	2.0	2.872795018106004	0.8727950181060038
3.0	4.0	3.7831355506110493	−0.21686444938895066
5.0	6.0	4.981947721328535	−1.0180522786714654
7.0	8.0	6.560643351529315	−1.4393566484706852
9.0	10.0	8.63960113465177	−1.3603988653482304
11.0	12.0	11.377345751994978	−0.6226542480050217
13.0	14.0	14.982635696139175	0.9826356961391749
15.0	16.0	19.7303815227609	3.7303815227609007

Мера отклонения $S = 21.036681938726666$;

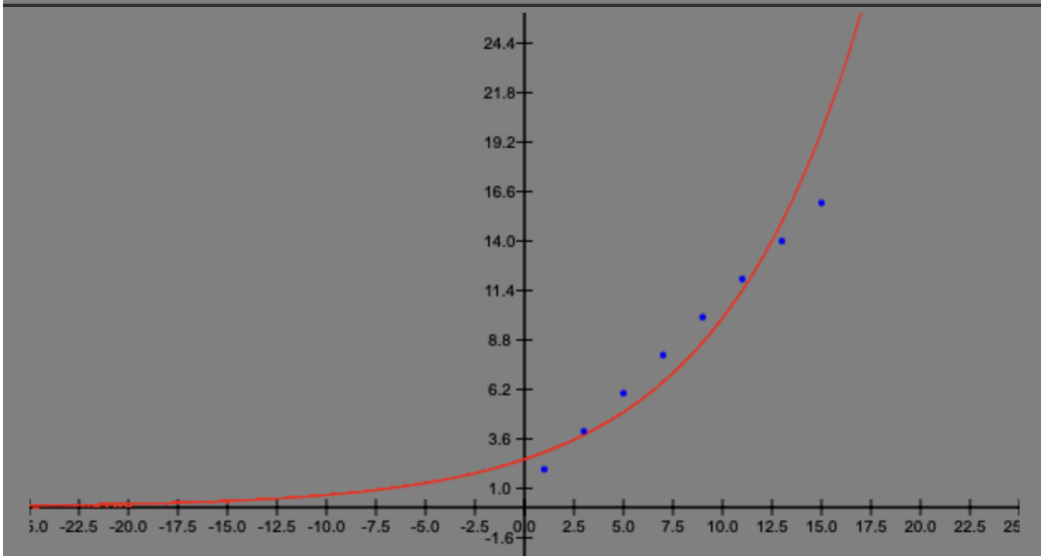
СКО $= 1.6215995937162888$;

Корреляция $= 0.93529762916765$;

Коэффициент детерминации $= 0.874781655126627$.

Функция вида :

$$\phi(x) = 2.503404377149598 \cdot e^{0.13763387337636754 \cdot x}$$



Линейная аппроксимация

x	y	$\phi(x)$	ε
1.0	2.0	2.0	0.0
3.0	4.0	4.0	0.0
5.0	6.0	6.0	0.0
7.0	8.0	8.0	0.0
9.0	10.0	10.0	0.0
11.0	12.0	12.0	0.0
13.0	14.0	14.0	0.0
15.0	16.0	16.0	0.0

Мера отклонения $S = 0.0$;

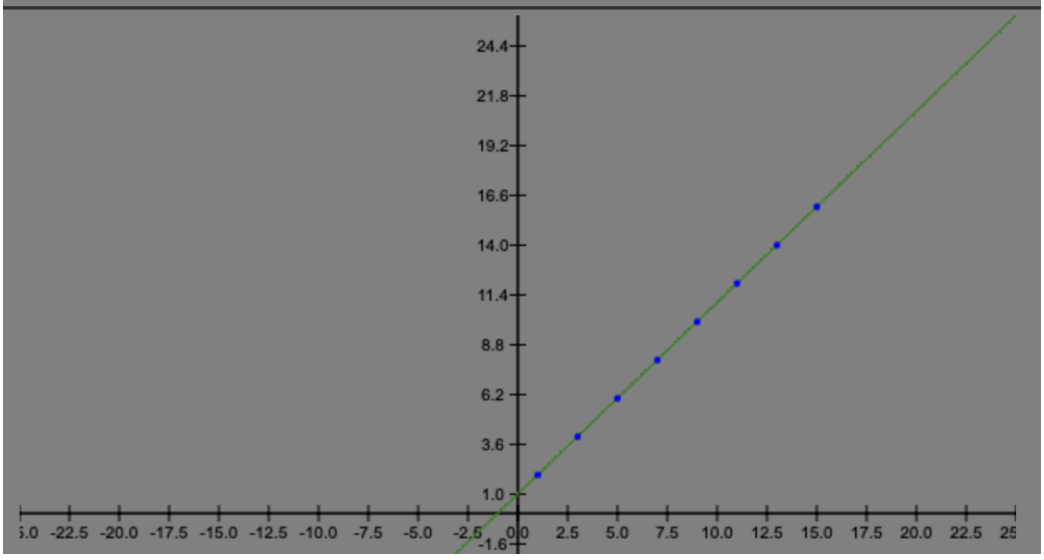
СКО $= 0.0$;

Корреляция $= 1.0$;

Коэффициент детерминации $= 1.0$.

Функция вида :

$$\phi(x) = 1.0 \cdot x + 1.0$$



Логарифмическая аппроксимация

x	y	$\phi(x)$	ε
1.0	2.0	−0.18480001061426307	−2.184800010614263
3.0	4.0	5.373926743777181	1.3739267437771812
5.0	6.0	7.958587493475508	1.958587493475508
7.0	8.0	9.661060006397944	1.6610600063979444
9.0	10.0	10.932653498168628	0.9326534981686283
11.0	12.0	11.948001292102491	−0.05199870789750882
13.0	14.0	12.79325672882558	−1.2067432711744193
15.0	16.0	13.517314247866954	−2.4826857521330457

Мера отклонения $S = 21.748715377738016$;

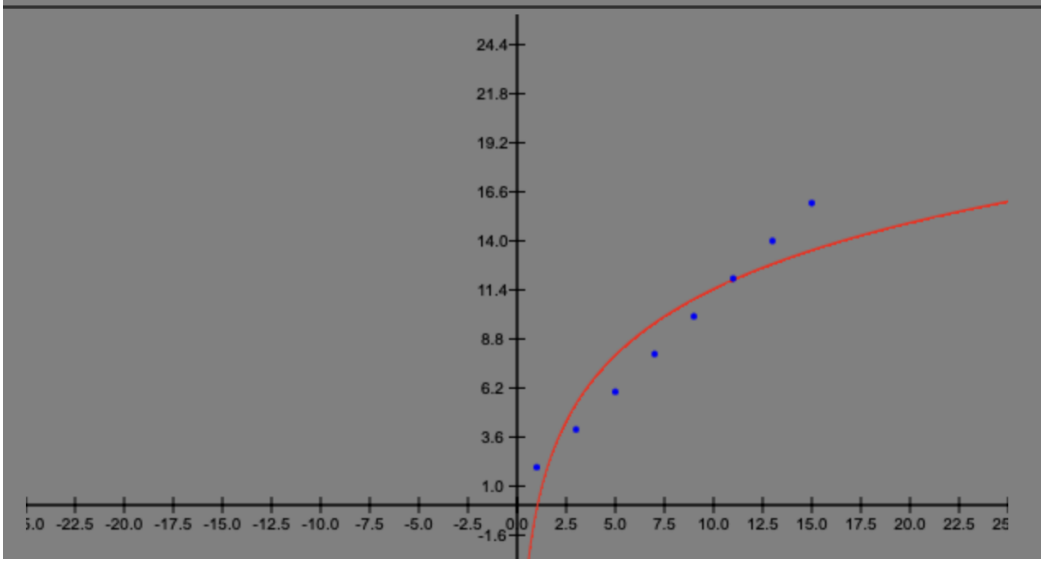
СКО $= 1.6488145505839193$;

Корреляция $= 0.9330291318317974$;

Коэффициент детерминации $= 0.8705433608467975$.

Функция вида :

$$\phi(x) = 5.059771141947179 \cdot ln(x) - 0.18480001061426307$$



Степенная аппроксимация

x	y	$\phi(x)$	ε
1.0	2.0	1.8410113078616224	−0.15898869213837763
3.0	4.0	4.317585861287266	0.31758586128726574
5.0	6.0	6.417516377254132	0.41751637725413193
7.0	8.0	8.331889186910649	0.33188918691064906
9.0	10.0	10.125710575477289	0.1257105754772887
11.0	12.0	11.83158152057966	−0.16841847942034072
13.0	14.0	13.468919470193542	−0.5310805298064576
15.0	16.0	15.050520253020942	−0.9494797469790583

Мера отклонения $S = 1.6383347932410877$;

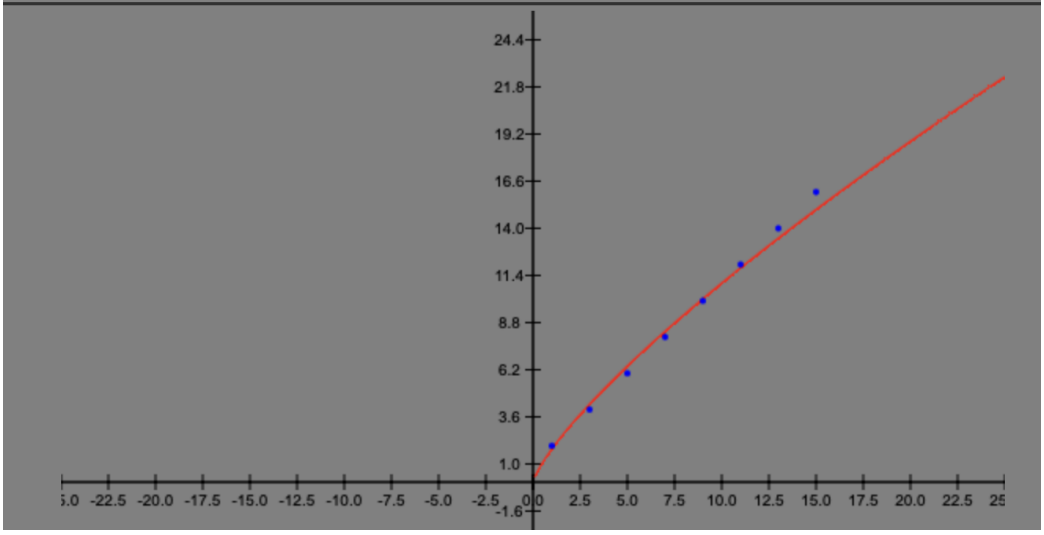
СКО $= 0.4525393343734177$;

Корреляция $= 0.9951120576010969$;

Коэффициент детерминации $= 0.9902480071830888$.

Функция вида :

$$\phi(x) = 1.8410113078616224 \cdot x^{0.7758709619114181}$$



Квадратичная аппроксимация

x	y	$\phi(x)$	ε
1.0	2.0	2.0000000000000124	$1.2434497875801753 \cdot 10^{-14}$
3.0	4.0	4.000000000000036	$3.552713678800501 \cdot 10^{-15}$
5.0	6.0	5.999999999999997	$-2.6645352591003757 \cdot 10^{-15}$
7.0	8.0	7.999999999999994	$-6.217248937900877 \cdot 10^{-15}$
9.0	10.0	9.999999999999993	$-7.105427357601002 \cdot 10^{-15}$
11.0	12.0	11.999999999999996	$-3.552713678800501 \cdot 10^{-15}$
13.0	14.0	14.000000000000002	$1.7763568394002505 \cdot 10^{-15}$
15.0	16.0	16.000000000000007	$7.105427357601002 \cdot 10^{-15}$

Мера отклонения $S = 3.2974385838238293 \cdot 10^{-28}$;

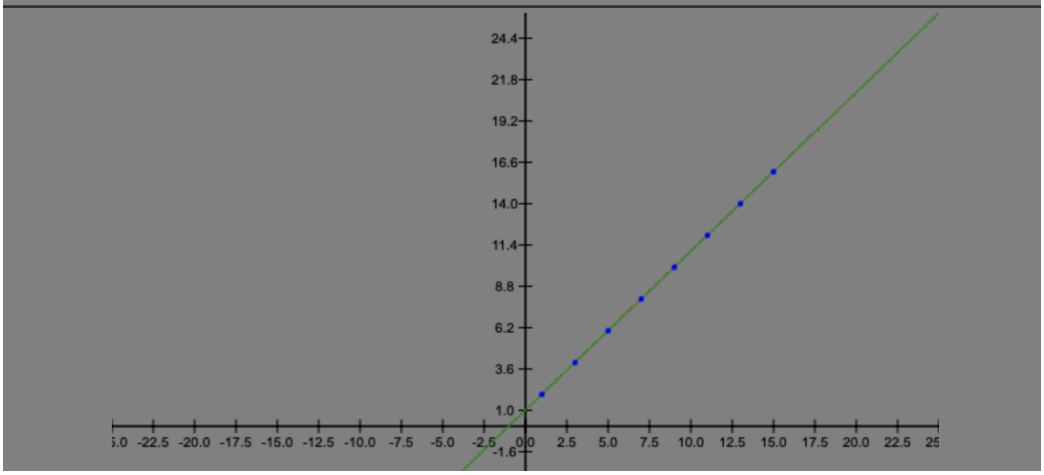
СКО $= 6.420123230733026 \cdot 10^{-15}$;

Корреляция $= 1.0$;

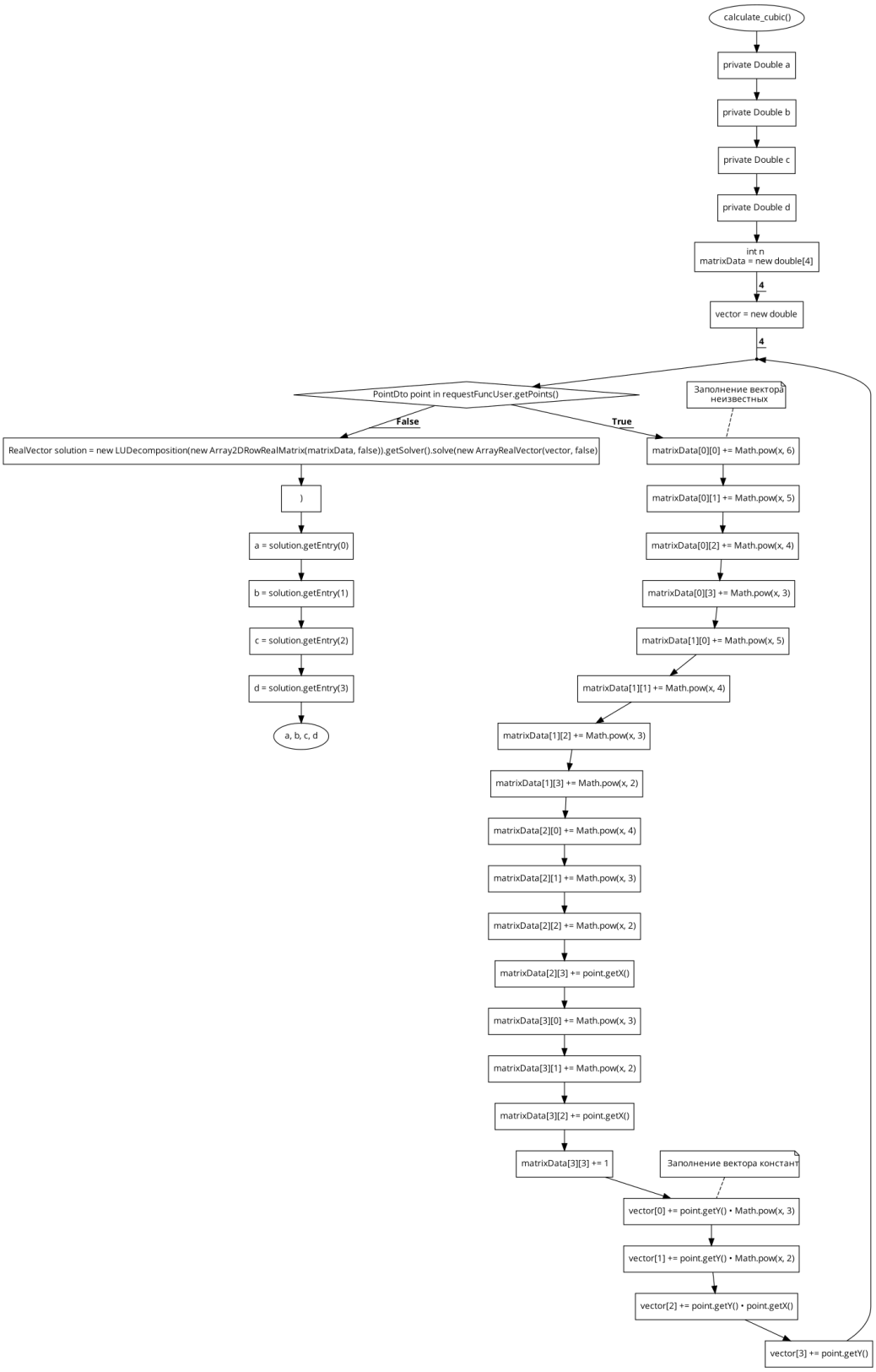
Коэффициент детерминации $= 1.0$.

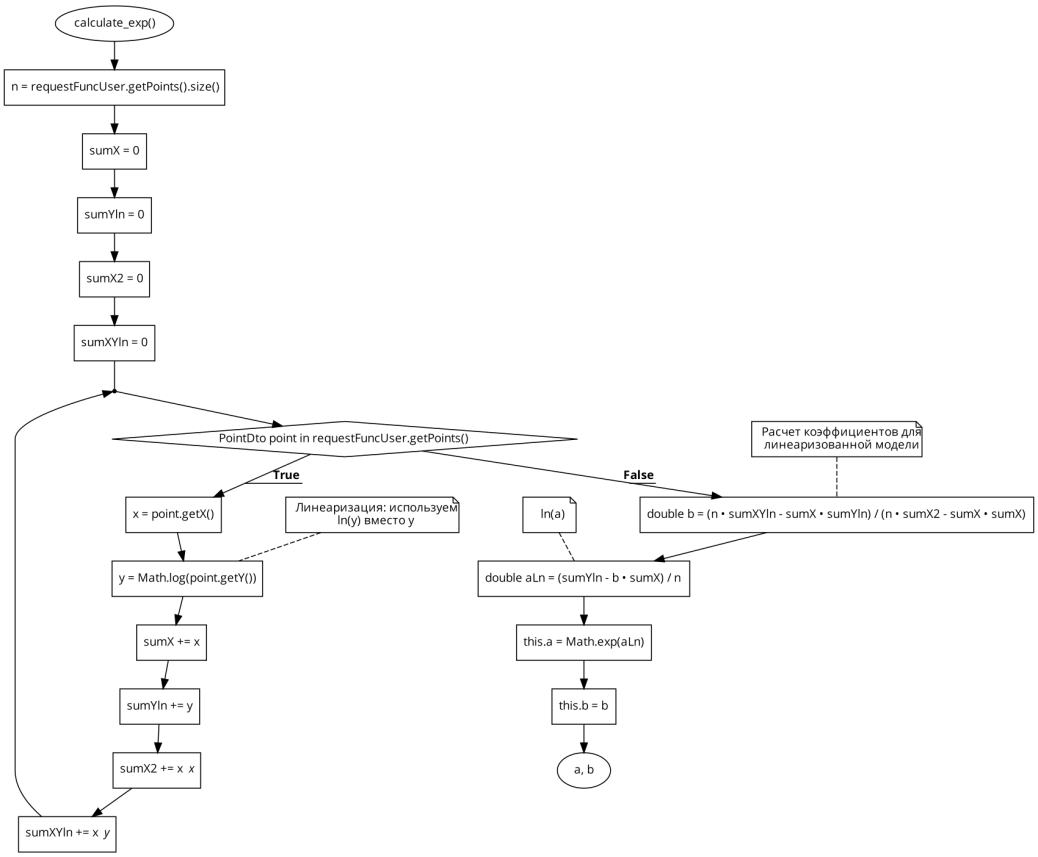
Функция вида :

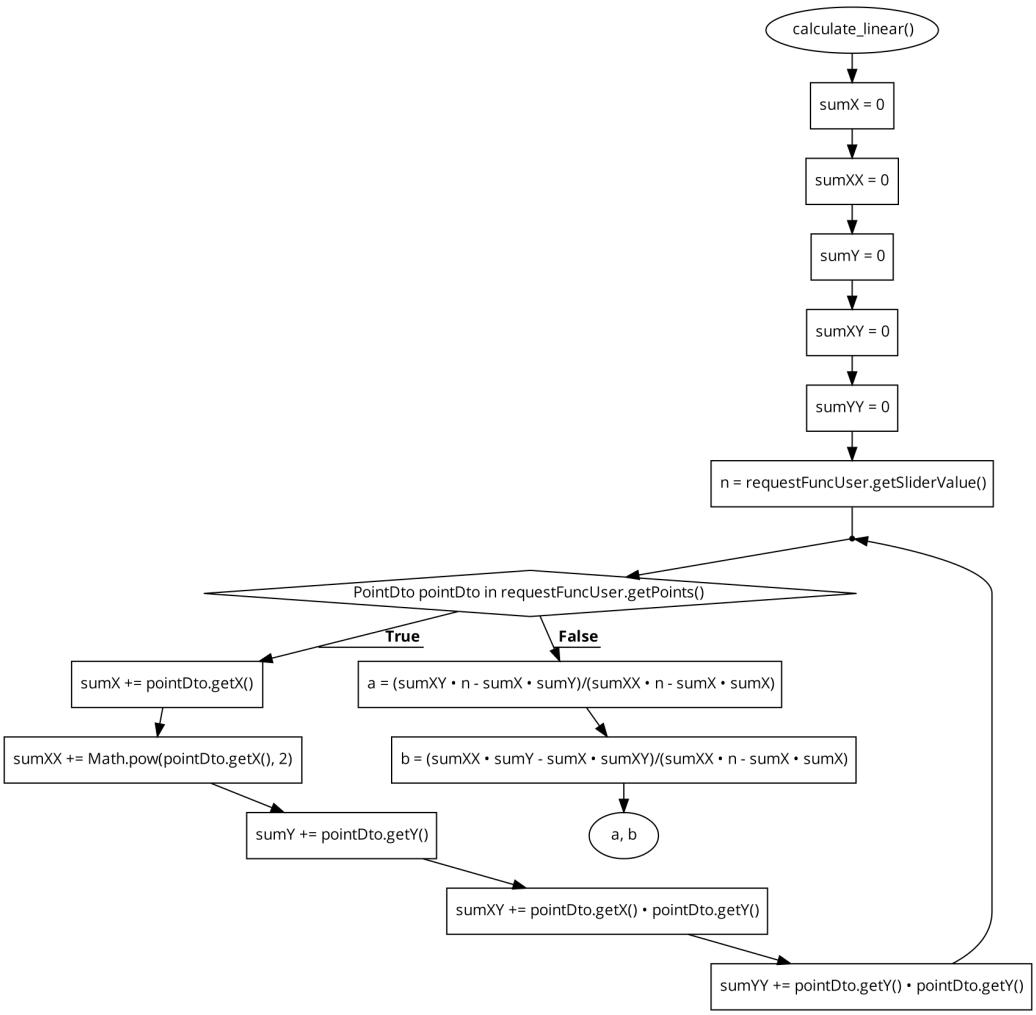
$$\phi(x) = 3.3393366572013674 \cdot 10^{-16} \cdot x^2 + 0.9999999999999943 \cdot x + 1.0000000000000175$$

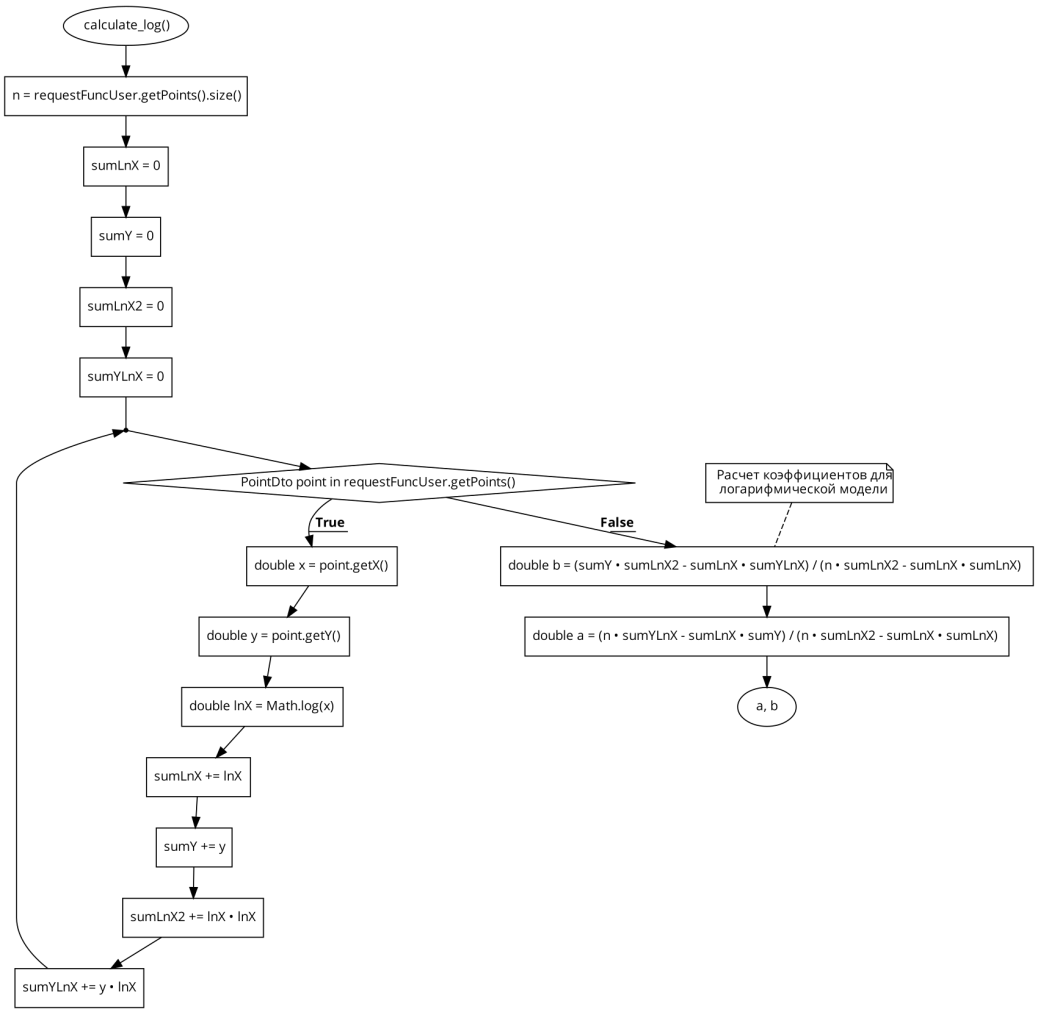


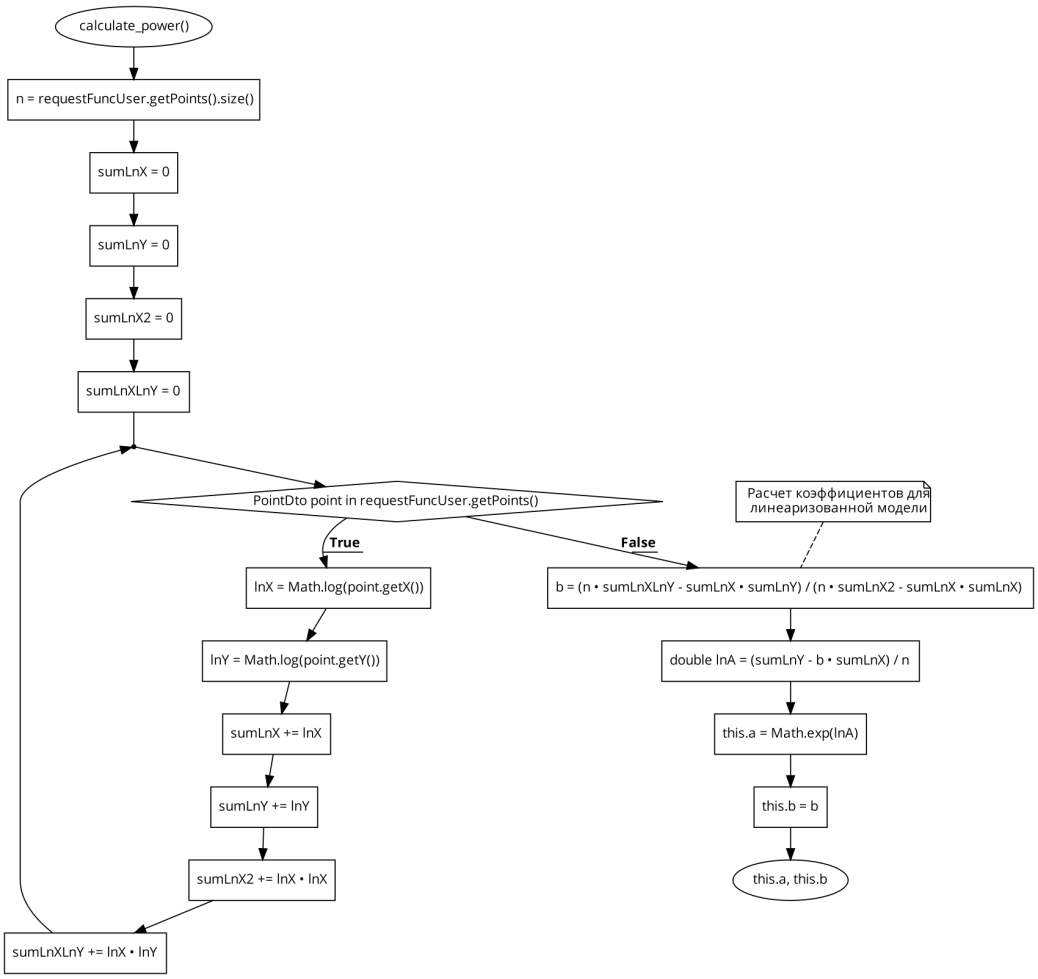
8 Схемы

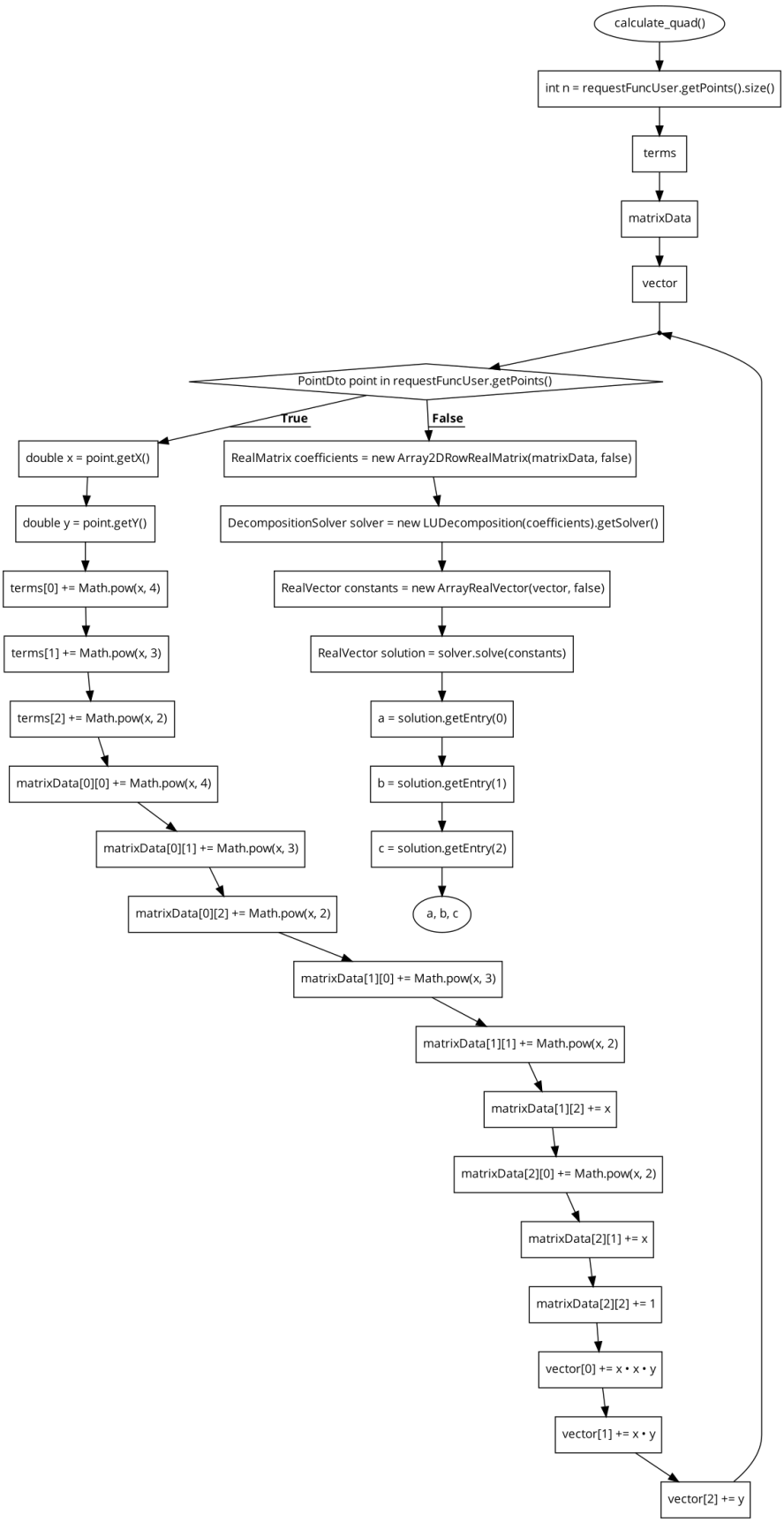

calculate_cubic()
private Double a
private Double b
private Double c
private Double d
int n
matrixData = new double[4]
4
vector = new double
4
PointDto point in requestFuncUser.getPoints()
Заполнение вектора неизвестных
False
True
RealVector solution = new LUDecomposition(new Array2DRowRealMatrix(matrixData, false)).getSolver().solve(new ArrayRealVector(vector, false)
)
a = solution.getEntry(0)
b = solution.getEntry(1)
c = solution.getEntry(2)
d = solution.getEntry(3)
a, b, c, d
matrixData[0][0] += Math.pow(x, 6)
matrixData[0][1] += Math.pow(x, 5)
matrixData[0][2] += Math.pow(x, 4)
matrixData[0][3] += Math.pow(x, 3)
matrixData[1][0] += Math.pow(x, 5)
matrixData[1][1] += Math.pow(x, 4)
matrixData[1][2] += Math.pow(x, 3)
matrixData[1][3] += Math.pow(x, 2)
matrixData[2][0] += Math.pow(x, 4)
matrixData[2][1] += Math.pow(x, 3)
matrixData[2][2] += Math.pow(x, 2)
matrixData[2][3] += point.getX()
matrixData[3][0] += Math.pow(x, 3)
matrixData[3][1] += Math.pow(x, 2)
matrixData[3][2] += point.getX()
matrixData[3][3] += 1
Заполнение вектора констант
vector[0] += point.getY() • Math.pow(x, 3)
vector[1] += point.getY() • Math.pow(x, 2)
vector[2] += point.getY() • point.getX()
vector[3] += point.getY()


calculate_exp()
n = requestFuncUser.getPoints().size()
sumX = 0
sumYln = 0
sumX2 = 0
sumXYln = 0
PointDto point in requestFuncUser.getPoints()
True
False
x = point.getX()
Линеаризация: используем ln(y) вместо y
ln(a)
Расчет коэффициентов для линеаризованной модели
double b = (n • sumXYln - sumX • sumYln) / (n • sumX2 - sumX • sumX)
y = Math.log(point.getY())
double aLn = (sumYln - b • sumX) / n
sumX += x
this.a = Math.exp(aLn)
sumYln += y
this.b = b
sumX2 += x x
a, b
sumXYln += x y


calculate_linear()
sumX = 0
sumXX = 0
sumY = 0
sumXY = 0
sumYY = 0
n = requestFuncUser.getSliderValue()
PointDto pointDto in requestFuncUser.getPoints()
True
False
sumX += pointDto.getX()
sumXX += Math.pow(pointDto.getX(), 2)
sumY += pointDto.getY()
sumXY += pointDto.getX() • pointDto.getY()
sumYY += pointDto.getY() • pointDto.getY()
a = (sumXY • n - sumX • sumY)/(sumXX • n - sumX • sumX)
b = (sumXX • sumY - sumX • sumXY)/(sumXX • n - sumX • sumX)
a, b


calculate_log()
n = requestFuncUser.getPoints().size()
sumLnX = 0
sumY = 0
sumLnX2 = 0
sumYLnX = 0
PointDto point in requestFuncUser.getPoints()
True
False
double x = point.getX()
double y = point.getY()
double lnX = Math.log(x)
sumLnX += lnX
sumY += y
sumLnX2 += lnX • lnX
sumYLnX += y • lnX
Расчет коэффициентов для логарифмической модели
double b = (sumY • sumLnX2 - sumLnX • sumYLnX) / (n • sumLnX2 - sumLnX • sumLnX)
double a = (n • sumYLnX - sumLnX • sumY) / (n • sumLnX2 - sumLnX • sumLnX)
a, b


calculate_power()
n = requestFuncUser.getPoints().size()
sumLnX = 0
sumLnY = 0
sumLnX2 = 0
sumLnXLnY = 0
PointDto point in requestFuncUser.getPoints()
True
False
lnX = Math.log(point.getX())
lnY = Math.log(point.getY())
sumLnX += lnX
sumLnY += lnY
sumLnX2 += lnX • lnX
sumLnXLnY += lnX • lnY
Расчет коэффициентов для линеаризованной модели
b = (n • sumLnXLnY - sumLnX • sumLnY) / (n • sumLnX2 - sumLnX • sumLnX)
double lnA = (sumLnY - b • sumLnX) / n
this.a = Math.exp(lnA)
this.b = b
this.a, this.b


calculate_quad()
int n = requestFuncUser.getPoints().size()
terms
matrixData
vector
PointDto point in requestFuncUser.getPoints()
True
False
double x = point.getX()
double y = point.getY()
terms[0] += Math.pow(x, 4)
terms[1] += Math.pow(x, 3)
terms[2] += Math.pow(x, 2)
matrixData[0][0] += Math.pow(x, 4)
matrixData[0][1] += Math.pow(x, 3)
matrixData[0][2] += Math.pow(x, 2)
RealMatrix coefficients = new Array2DRowRealMatrix(matrixData, false)
DecompositionSolver solver = new LUDecomposition(coefficients).getSolver()
RealVector constants = new ArrayRealVector(vector, false)
RealVector solution = solver.solve(constants)
a = solution.getEntry(0)
b = solution.getEntry(1)
c = solution.getEntry(2)
a, b, c
matrixData[1][0] += Math.pow(x, 3)
matrixData[1][1] += Math.pow(x, 2)
matrixData[1][2] += x
matrixData[2][0] += Math.pow(x, 2)
matrixData[2][1] += x
matrixData[2][2] += 1
vector[0] += x • x • y
vector[1] += x • y
vector[2] += y

9 GitHub

Ссылка на мой репозиторий на GitHub: `https://github.com/Alex-de-bug/cm_math/tree/main/lab4`.

10 Вывод

При работе были изучены метод аппроксации различными функциями, написано приложение для автоматизации подсчётов. Изучено поведение аппроксимации различных функций.