

Университет ИТМО

Вычислительная математика  
Лабораторная работа №4  
«Аппроксимация функции методом наименьших квадратов»

Работу выполнил:  
Бавыкин Роман  
Группа: Р3210  
Вариант 2

Санкт-Петербург  
2022 г.

### Цель работы:

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

### Порядок выполнения работы:

$$\frac{3x}{x^4 + 1}; x \in [0; 2]; h = 0,2$$

0	0,2	0,4	0,6	0,8	1	1,2	1,4	1,6	1,8	2
0	0,60	1,17	1,59	1,70	1,50	1,17	0,87	0,64	0,47	0,35

$$SX = 11; SXX = 15,4; SY = 10,062; SXY = 9,594$$

$$\begin{cases} aSXX + bSX = SXY \\ aSX + bn = SY \end{cases}$$

$$\Delta = SXX \cdot n - SX \cdot SX = 15,4 \cdot 11 - 11 \cdot 11 = 48,4$$

$$\Delta_1 = SXY \cdot n - SX \cdot SY = 9,594 \cdot 11 - 11 \cdot 10,062 = -5,148$$

$$\Delta_2 = SXX \cdot SY - SX \cdot SXY = 15,4 \cdot 10,062 - 11 \cdot 9,594 = 49,421$$

$$a = \frac{\Delta_1}{\Delta} = \frac{-5,148}{48,4} = -0,106; b = \frac{\Delta_2}{\Delta} = \frac{49,421}{48,4} = 1,021$$

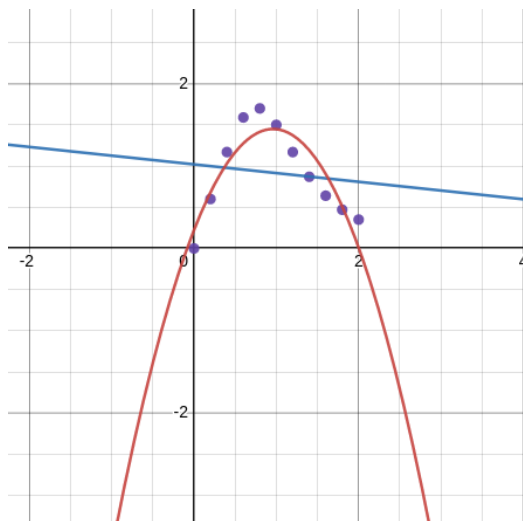
$$SXXX = 24,2; SXXXX = 40,533; SXXY = 11,322$$

$$\begin{cases} na_0 + SXa_1 + SXXa_2 = SY \\ SXa_0 + SXXa_2 + SXXXXa_3 = SXY \\ SXXa_0 + SXXXXa_1 + SXXXXa_2 = SXXY \end{cases}$$

$$\begin{cases} 11a_0 + 11a_1 + 15,4a_2 = 10,062 \\ 11a_0 + 15,4a_2 + 24,2a_3 = 9,594 \\ 15,4a_0 + 24,2a_1 + 40,533a_2 = 11,322 \end{cases}$$

$$\begin{cases} a_0 = 0,222 \\ a_1 = 2,558 \\ a_2 = -1,332 \end{cases}$$

$$\delta_{\text{лин}} = 0,525; \delta_{\text{кв}} = 0,233$$



**Рабочие формулы:**

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

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

Степенная функция:  $\phi(x, a, b) = ax^b$

Экспоненциальная функция:  $\phi(x, a, b) = ae^{bx}$

Логарифмическая функция:  $\phi(x, a, b) = a \ln(x) + b$

$$r = \sum_{i=1}^n \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}$$

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

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \phi_i)^2}{\sum_{i=1}^n \phi_i^2 - \frac{1}{n} \left( \sum_{i=1}^n \phi_i \right)^2}$$

## Листинг программы:

```
1  @Component
2  public class LinearApproximation implements Approximation{
3      private ApproximationAnalyzer approximationAnalyzer;
4
5      @Autowired
6      public LinearApproximation(ApproximationAnalyzer approximationAnalyzer) {
7          this.approximationAnalyzer = approximationAnalyzer;
8      }
9
10
11     @Override
12     public ApproximationResults approximate(double[] xValues, double[] yValues) {
13         double x = 0;
14         double x2 = 0;
15         double y = 0;
16         double xy = 0;
17         int n = Math.min(xValues.length, yValues.length);
18         for (int i = 0; i < n; i++) {
19             x += xValues[i];
20             x2 += Math.pow(xValues[i], 2);
21             y += yValues[i];
22             xy += xValues[i] * yValues[i];
23         }
24         double[][] left = new double[][] {
25             {x2, x},
26             {x, n}};
27         double[] right = new double[]{
28             xy, y
29         };
30         DecompositionSolver solver = new LUDecomposition(new Array2DRowRealMatrix(left)).getSolver();
31         double[] solution = solver.solve(new ArrayRealVector(right)).toArray();
32         LinearFunction function = new LinearFunction();
33         function.setA(solution[0]);
34         function.setB(solution[1]);
35         return ApproximationResults
36             .builder()
37             .function(function)
38             .deviation(approximationAnalyzer.calculateDeviation(xValues, yValues, function))
39             .r2(approximationAnalyzer.calculateR2(xValues, yValues, function))
40             .correlation(approximationAnalyzer.calculateCorrelation(xValues, yValues))
41             .build();
42     }
43 }
```

```

1  @Component
2  public class Polynomial2Approximation implements Approximation{
3      private ApproximationAnalyzer approximationAnalyzer;
4
5      @Autowired
6      public Polynomial2Approximation(ApproximationAnalyzer approximationAnalyzer) {
7          this.approximationAnalyzer = approximationAnalyzer;
8      }
9
10     @Override
11     public ApproximationResults approximate(double[] xValues, double[] yValues) {
12         double x = 0;
13         double x2 = 0;
14         double x3 = 0;
15         double x4 = 0;
16         double y = 0;
17         double xy = 0;
18         double x2y = 0;
19         int n = Math.min(xValues.length, yValues.length);
20         for (int i = 0; i < n; i++) {
21             x += xValues[i];
22             x2 += Math.pow(xValues[i], 2);
23             x3 += Math.pow(xValues[i], 3);
24             x4 += Math.pow(xValues[i], 4);
25             y += yValues[i];
26             xy += xValues[i] * yValues[i];
27             x2y += Math.pow(xValues[i], 2) * yValues[i];
28         }
29         double[][] left = new double[][]{
30             {n, x, x2},
31             {x, x2, x3},
32             {x2, x3, x4}
33         };
34         double[] right = new double[]{
35             y, xy, x2y
36         };
37         DecompositionSolver solver = new LUDecomposition(new Array2DRowRealMatrix(left)).getSolver();
38         double[] solution = solver.solve(new ArrayRealVector(right)).toArray();
39         Polynomial2Function function = new Polynomial2Function();
40         function.setA0(solution[0]);
41         function.setA1(solution[1]);
42         function.setA2(solution[2]);
43         return ApproximationResults
44             .builder()
45             .function(function)
46             .deviation(approximationAnalyzer.calculateDeviation(xValues, yValues, function))
47             .r2(approximationAnalyzer.calculateR2(xValues, yValues, function))
48             .correlation(null)
49             .build();
50     }
51 }
52 }

```

```

1  @Component
2  public class Polynomial3Approximation implements Approximation{
3      private ApproximationAnalyzer approximationAnalyzer;
4
5      @Autowired
6      public Polynomial3Approximation(ApproximationAnalyzer approximationAnalyzer) {
7          this.approximationAnalyzer = approximationAnalyzer;
8      }
9
10     @Override
11     public ApproximationResults approximate(double[] xValues, double[] yValues) {
12         double x = 0;
13         double x2 = 0;
14         double x3 = 0;
15         double x4 = 0;
16         double x5 = 0;
17         double x6 = 0;
18         double y = 0;
19         double xy = 0;
20         double x2y = 0;
21         double x3y = 0;
22         int n = Math.min(xValues.length, yValues.length);
23         for (int i = 0; i < n; i++) {
24             x += xValues[i];
25             x2 += Math.pow(xValues[i], 2);
26             x3 += Math.pow(xValues[i], 3);
27             x4 += Math.pow(xValues[i], 4);
28             x5 += Math.pow(xValues[i], 5);
29             x6 += Math.pow(xValues[i], 6);
30             y += yValues[i];
31             xy += xValues[i] * yValues[i];
32             x2y += Math.pow(xValues[i], 2) * yValues[i];
33             x3y += Math.pow(xValues[i], 3) * yValues[i];
34         }
35         double[][] left = new double[][]{
36             {n, x, x2, x3},
37             {x, x2, x3, x4},
38             {x2, x3, x4, x5},
39             {x3, x4, x5, x6}
40         };
41         double[] right = new double[]{
42             y, xy, x2y, x3y
43         };
44         DecompositionSolver solver = new LUDecomposition(new Array2DRowRealMatrix(left)).getSolver();
45         double[] solution = solver.solve(new ArrayRealVector(right)).toArray();
46         Polynomial3Function function = new Polynomial3Function();
47         function.setA0(solution[0]);
48         function.setA1(solution[1]);
49         function.setA2(solution[2]);
50         function.setA3(solution[3]);
51         return ApproximationResults
52             .builder()
53             .function(function)
54             .deviation(approximationAnalyzer.calculateDeviation(xValues, yValues, function))
55             .r2(approximationAnalyzer.calculateR2(xValues, yValues, function))
56             .correlation(null)
57             .build();
58     }
59 }

```

```

1  @Component
2  public class ExponentialApproximation implements Approximation{
3      private LinearApproximation linearApproximation;
4      private ApproximationAnalyzer approximationAnalyzer;
5
6      @Autowired
7      public ExponentialApproximation(LinearApproximation linearApproximation, ApproximationAnalyzer approximationAnalyzer) {
8          this.linearApproximation = linearApproximation;
9          this.approximationAnalyzer = approximationAnalyzer;
10     }
11     @Override
12     public ApproximationResults approximate(double[] xValues, double[] yValues) {
13         int n = Math.min(xValues.length, yValues.length);
14         double[] xValuesNew = new double[n];
15         double[] yValuesNew = new double[n];
16         for (int i = 0; i < n; i++) {
17             yValuesNew[i] = yValues[i] > 0 ? Math.log(yValues[i]) : yValues[i];
18             xValuesNew[i] = xValues[i];
19         }
20         ApproximationResults results = linearApproximation.approximate(xValuesNew, yValuesNew);
21         LinearFunction linearFunction = (LinearFunction) results.getFunction();
22         ExponentialFunction exponentialFunction = new ExponentialFunction();
23         exponentialFunction.setA(Math.exp(linearFunction.getB()));
24         exponentialFunction.setB(linearFunction.getA());
25         results.setFunction(exponentialFunction);
26         results.setDeviation(approximationAnalyzer.calculateDeviation(xValues, yValues, exponentialFunction));
27         results.setR2(approximationAnalyzer.calculateR2(xValues, yValues, exponentialFunction));
28         results.setCorrelation(null);
29         return results;
30     }
31 }

```

```

1  @Component
2  public class LogarithmicApproximation implements Approximation {
3      private LinearApproximation linearApproximation;
4      private ApproximationAnalyzer approximationAnalyzer;
5
6      @Autowired
7      public LogarithmicApproximation(LinearApproximation linearApproximation, ApproximationAnalyzer approximationAnalyzer) {
8          this.linearApproximation = linearApproximation;
9          this.approximationAnalyzer = approximationAnalyzer;
10     }
11
12     @Override
13     public ApproximationResults approximate(double[] xValues, double[] yValues) {
14         int n = Math.min(xValues.length, yValues.length);
15         double[] xValuesNew = new double[n];
16         double[] yValuesNew = new double[n];
17         for (int i = 0; i < n; i++) {
18             xValuesNew[i] = xValues[i] > 0 ? Math.log(xValues[i]) : xValues[i];
19             yValuesNew[i] = yValues[i];
20         }
21         ApproximationResults results = linearApproximation.approximate(xValuesNew, yValuesNew);
22         LinearFunction linearFunction = (LinearFunction) results.getFunction();
23         LogarithmicFunction logarithmicFunction = new LogarithmicFunction();
24         logarithmicFunction.setA(linearFunction.getA());
25         logarithmicFunction.setB(linearFunction.getB());
26         results.setFunction(logarithmicFunction);
27         results.setDeviation(approximationAnalyzer.calculateDeviation(xValues, yValues, logarithmicFunction));
28         results.setR2(approximationAnalyzer.calculateR2(xValues, yValues, logarithmicFunction));
29         results.setCorrelation(null);
30         return results;
31     }
32 }

```

```

1  @Component
2  public class PowerApproximation implements Approximation{
3      private LinearApproximation linearApproximation;
4      private ApproximationAnalyzer approximationAnalyzer;
5
6      @Autowired
7      public PowerApproximation(LinearApproximation linearApproximation, ApproximationAnalyzer approximationAnalyzer) {
8          this.linearApproximation = linearApproximation;
9          this.approximationAnalyzer = approximationAnalyzer;
10     }
11     @Override
12     public ApproximationResults approximate(double[] xValues, double[] yValues) {
13         int n = Math.min(xValues.length, yValues.length);
14         double[] xValuesNew = new double[n];
15         double[] yValuesNew = new double[n];
16         for (int i = 0; i < n; i++) {
17             if (yValues[i] > 0 && xValues[i] > 0) {
18                 xValuesNew[i] = Math.log(xValues[i]);
19                 yValuesNew[i] = Math.log(yValues[i]);
20             } else {
21                 xValuesNew[i] = xValues[i];
22                 yValuesNew[i] = yValues[i];
23             }
24         }
25         ApproximationResults results = linearApproximation.approximate(xValuesNew, yValuesNew);
26         LinearFunction linearFunction = (LinearFunction) results.getFunction();
27         PowerFunction powerFunction = new PowerFunction();
28         powerFunction.setA(Math.exp(linearFunction.getB()));
29         powerFunction.setB(linearFunction.getA());
30         results.setFunction(powerFunction);
31         results.setDeviation(approximationAnalyzer.calculateDeviation(xValues, yValues, powerFunction));
32         results.setR2(approximationAnalyzer.calculateR2(xValues, yValues, powerFunction));
33         results.setCorrelation(null);
34         return results;
35     }
36 }

```

```

1  @Component
2  public class ApproximationAnalyzer {
3      public double calculateR2 (double[] xValues, double[] yValues, Function function) {
4          int n = Math.min(xValues.length, yValues.length);
5          double upper = 0;
6          double lowerLeft = 0;
7          double lowerRight = 0;
8          for (int i = 0; i < n; i++) {
9              upper += Math.pow(yValues[i] - function.apply(xValues[i]), 2);
10             lowerLeft += Math.pow(function.apply(xValues[i]), 2);
11             lowerRight += function.apply(xValues[i]);
12         }
13         double result = 1 - upper / (lowerLeft - Math.pow(lowerRight, 2) / n);
14         if (Double.isNaN(result)) {
15             result = 0;
16         }
17         return result;
18     }
19
20     public double calculateCorrelation(double[] xValues, double[] yValues) {
21         int n = Math.min(xValues.length, yValues.length);
22         double averageX = 0;
23         double averageY = 0;
24         for (int i = 0; i < n; i++) {
25             averageX += xValues[i];

```

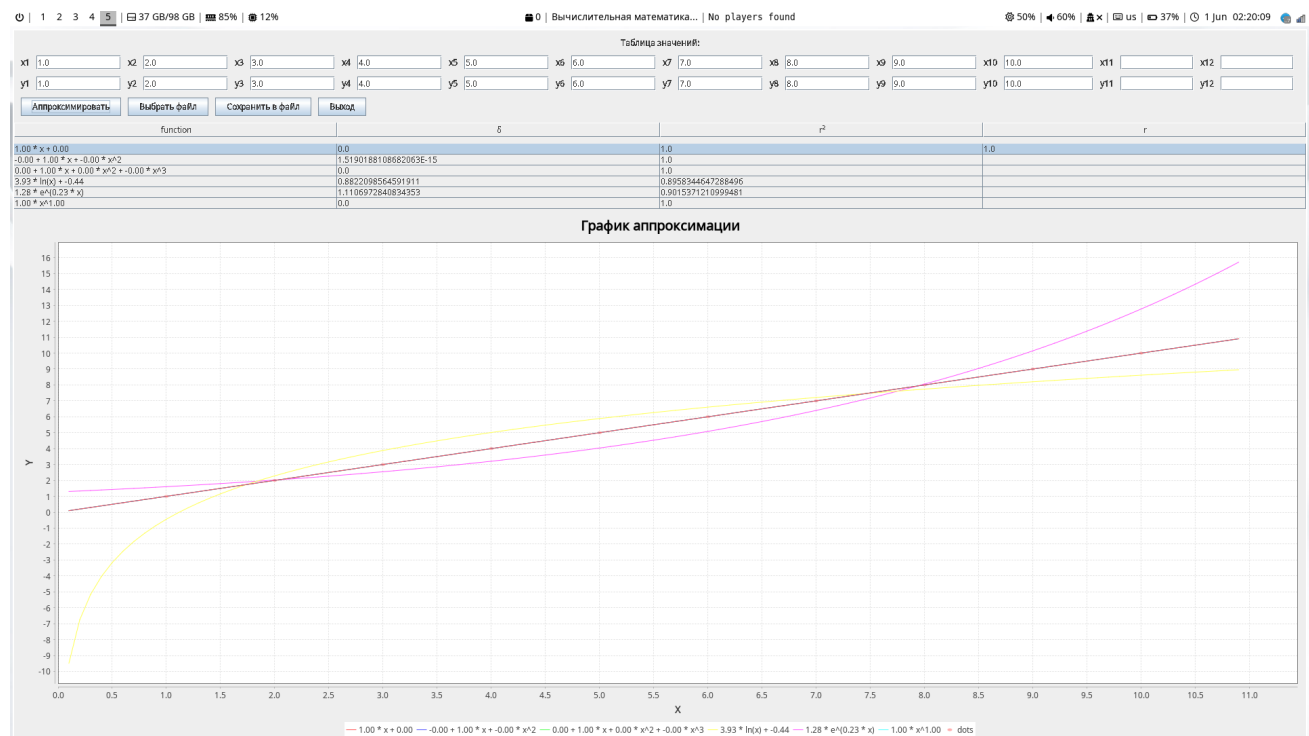


```

26         averageY += yValues[i];
27     }
28     averageX /= n;
29     averageY /= n;
30     double upper = 0;
31     double lowerLeft = 0;
32     double lowerRight = 0;
33     for (int i = 0; i < n; i++) {
34         upper += (xValues[i] - averageX) * (yValues[i] - averageY);
35         lowerLeft += Math.pow(xValues[i] - averageX, 2);
36         lowerRight += Math.pow(yValues[i] - averageY, 2);
37     }
38     return upper / Math.sqrt(lowerLeft * lowerRight);
39 }
40
41 public double calculateDeviation(double[] xValues, double[] yValues, Function function) {
42     int n = Math.min(xValues.length, yValues.length);
43     double sum = 0;
44     for (int i = 0; i < n; i++) {
45         sum += Math.pow(function.apply(xValues[i]) - yValues[i], 2);
46     }
47     return Math.sqrt(sum / n);
48 }
49 }

```

## Результаты выполнения программы:



**Выводы:** во время выполнения лабораторной работы ознакомился с видами аппроксимации функций (линейная, квадратичная, кубическая, логарифмическая, экспоненциальная, степенная), а также методами оценки аппроксимации (метод наименьших квадратов, коэффициент корреляции, коэффициент достоверности), все методы аппроксимации и оценки реализовал программно, а также вручную произвел линейную и квадратичную аппроксимации для заданной функции, а также оценил полученные аппроксимации с помощью метода наименьших квадратов.