Университет ИТМО

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

Работу выполнил:
Бавыкин Роман
Группа: P3210
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

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

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

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

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

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

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

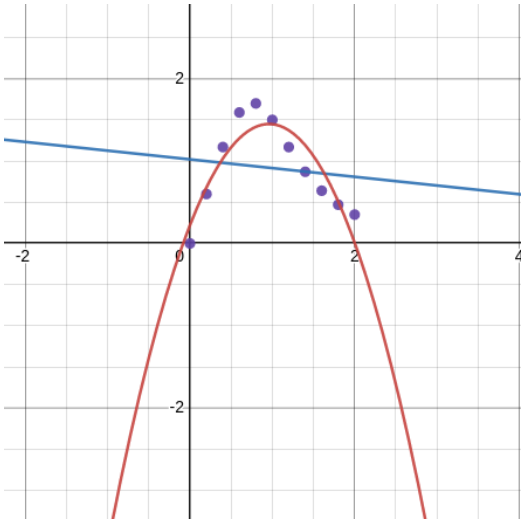
$$SXXX=24{,}2;\ SXXXX=40{,}533;\ SXXY=11{,}322$$

$$\begin{cases} na_0+SXa_1+SXXa_2=SY \\ SXa_0+SXXa_2+SXXXa_3=SXY \\ SXXa_0+SXXXa_1+SXXXXa_2=SXXY \end{cases}$$

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

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

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



Рабочие формулы:

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

$$Квадратичная\ функция: \phi(x,a_0,a_1,a_2)=a_0+a_1x+a_2x^2$$

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

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

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

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

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

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

Листинг программы:

```java
@Component
public class LinearApproximation implements Approximation{
    private ApproximationAnalyzer approximationAnalyzer;

    @Autowired
    public LinearApproximation(ApproximationAnalyzer approximationAnalyzer) {
        this.approximationAnalyzer = approximationAnalyzer;
    }


    @Override
    public ApproximationResults approximate(double[] xValues, double[] yValues) {
        double x = 0;
        double x2 = 0;
        double y = 0;
        double xy = 0;
        int n = Math.min(xValues.length, yValues.length);
        for (int i = 0; i < n; i++) {
            x += xValues[i];
            x2 += Math.pow(xValues[i], 2);
            y += yValues[i];
            xy += xValues[i] * yValues[i];
        }
        double[][] left = new double[][] {
                {x2, x},
                {x, n}};
        double[] right = new double[]{
                xy, y
        };
        DecompositionSolver solver = new LUDecomposition(new Array2DRowRealMatrix(left)).getSolver();
        double[] solution = solver.solve(new ArrayRealVector(right)).toArray();
        LinearFunction function = new LinearFunction();
        function.setA(solution[0]);
        function.setB(solution[1]);
        return ApproximationResults
                .builder()
                .function(function)
                .deviation(approximationAnalyzer.calculateDeviation(xValues, yValues, function))
                .r2(approximationAnalyzer.calculateR2(xValues, yValues, function))
                .correlation(approximationAnalyzer.calculateCorrelation(xValues, yValues))
                .build();
    }
}
```

```java
@Component
public class Polynomial2Approximation implements Approximation{
    private ApproximationAnalyzer approximationAnalyzer;

    @Autowired
    public Polynomial2Approximation(ApproximationAnalyzer approximationAnalyzer) {
        this.approximationAnalyzer = approximationAnalyzer;
    }

    @Override
    public ApproximationResults approximate(double[] xValues, double[] yValues) {
        double x = 0;
        double x2 = 0;
        double x3 = 0;
        double x4 = 0;
        double y = 0;
        double xy = 0;
        double x2y = 0;
        int n = Math.min(xValues.length, yValues.length);
        for (int i = 0; i < n; i++) {
            x += xValues[i];
            x2 += Math.pow(xValues[i], 2);
            x3 += Math.pow(xValues[i], 3);
            x4 += Math.pow(xValues[i], 4);
            y += yValues[i];
            xy += xValues[i] * yValues[i];
            x2y += Math.pow(xValues[i], 2) * yValues[i];
        }
        double[][] left = new double[][]{
                {n, x, x2},
                {x, x2, x3},
                {x2, x3, x4}
        };
        double[] right = new double[]{
                y, xy, x2y
        };
        DecompositionSolver solver = new LUDecomposition(new Array2DRowRealMatrix(left)).getSolver();
        double[] solution = solver.solve(new ArrayRealVector(right)).toArray();
        Polynomial2Function function = new Polynomial2Function();
        function.setA0(solution[0]);
        function.setA1(solution[1]);
        function.setA2(solution[2]);
        return ApproximationResults
                .builder()
                .function(function)
                .deviation(approximationAnalyzer.calculateDeviation(xValues, yValues, function))
                .r2(approximationAnalyzer.calculateR2(xValues, yValues, function))
                .correlation(null)
                .build();

    }
}
```

```java
@Component
public class Polynomial3Approximation implements Approximation{
    private ApproximationAnalyzer approximationAnalyzer;

    @Autowired
    public Polynomial3Approximation(ApproximationAnalyzer approximationAnalyzer) {
        this.approximationAnalyzer = approximationAnalyzer;
    }

    @Override
    public ApproximationResults approximate(double[] xValues, double[] yValues) {
        double x = 0;
        double x2 = 0;
        double x3 = 0;
        double x4 = 0;
        double x5 = 0;
        double x6 = 0;
        double y = 0;
        double xy = 0;
        double x2y = 0;
        double x3y = 0;
        int n = Math.min(xValues.length, yValues.length);
        for (int i = 0; i < n; i++) {
            x += xValues[i];
            x2 += Math.pow(xValues[i], 2);
            x3 += Math.pow(xValues[i], 3);
            x4 += Math.pow(xValues[i], 4);
            x5 += Math.pow(xValues[i], 5);
            x6 += Math.pow(xValues[i], 6);
            y += yValues[i];
            xy += xValues[i] * yValues[i];
            x2y += Math.pow(xValues[i], 2) * yValues[i];
            x3y += Math.pow(xValues[i], 3) * yValues[i];
        }
        double[][] left = new double[][]{
                {n, x, x2, x3},
                {x, x2, x3, x4},
                {x2, x3, x4, x5},
                {x3, x4, x5, x6}
        };
        double[] right = new double[]{
                y, xy, x2y, x3y
        };
        DecompositionSolver solver = new LUDecomposition(new Array2DRowRealMatrix(left)).getSolver();
        double[] solution = solver.solve(new ArrayRealVector(right)).toArray();
        Polynomial3Function function = new Polynomial3Function();
        function.setA0(solution[0]);
        function.setA1(solution[1]);
        function.setA2(solution[2]);
        function.setA3(solution[3]);
        return ApproximationResults
                .builder()
                .function(function)
                .deviation(approximationAnalyzer.calculateDeviation(xValues, yValues, function))
                .r2(approximationAnalyzer.calculateR2(xValues, yValues, function))
                .correlation(null)
                .build();
    }
}
```

```java
@Component
public class ExponentialApproximation implements Approximation{
    private LinearApproximation linearApproximation;
    private ApproximationAnalyzer approximationAnalyzer;

    @Autowired
    public ExponentialApproximation(LinearApproximation linearApproximation, ApproximationAnalyzer approximationAnalyzer) {
        this.linearApproximation = linearApproximation;
        this.approximationAnalyzer = approximationAnalyzer;
    }
    @Override
    public ApproximationResults approximate(double[] xValues, double[] yValues) {
        int n = Math.min(xValues.length, yValues.length);
        double[] xValuesNew = new double[n];
        double[] yValuesNew = new double[n];
        for (int i = 0; i < n; i++) {
            yValuesNew[i] = yValues[i] > 0 ? Math.log(yValues[i]) : yValues[i];
            xValuesNew[i] = xValues[i];
        }
        ApproximationResults results = linearApproximation.approximate(xValuesNew, yValuesNew);
        LinearFunction linearFunction = (LinearFunction) results.getFunction();
        ExponentialFunction exponentialFunction = new ExponentialFunction();
        exponentialFunction.setA(Math.exp(linearFunction.getB()));
        exponentialFunction.setB(linearFunction.getA());
        results.setFunction(exponentialFunction);
        results.setDeviation(approximationAnalyzer.calculateDeviation(xValues, yValues, exponentialFunction));
        results.setR2(approximationAnalyzer.calculateR2(xValues, yValues, exponentialFunction));
        results.setCorrelation(null);
        return results;
    }
}
```

```java
@Component
public class LogarithmicApproximation implements Approximation {
    private LinearApproximation linearApproximation;
    private ApproximationAnalyzer approximationAnalyzer;

    @Autowired
    public LogarithmicApproximation(LinearApproximation linearApproximation, ApproximationAnalyzer approximationAnalyzer) {
        this.linearApproximation = linearApproximation;
        this.approximationAnalyzer = approximationAnalyzer;
    }

    @Override
    public ApproximationResults approximate(double[] xValues, double[] yValues) {
        int n = Math.min(xValues.length, yValues.length);
        double[] xValuesNew = new double[n];
        double[] yValuesNew = new double[n];
        for (int i = 0; i < n; i++) {
            xValuesNew[i] = xValues[i] > 0 ? Math.log(xValues[i]) : xValues[i];
            yValuesNew[i] = yValues[i];
        }
        ApproximationResults results = linearApproximation.approximate(xValuesNew, yValuesNew);
        LinearFunction linearFunction = (LinearFunction) results.getFunction();
        LogarithmicFunction logarithmicFunction = new LogarithmicFunction();
        logarithmicFunction.setA(linearFunction.getA());
        logarithmicFunction.setB(linearFunction.getB());
        results.setFunction(logarithmicFunction);
        results.setDeviation(approximationAnalyzer.calculateDeviation(xValues, yValues, logarithmicFunction));
        results.setR2(approximationAnalyzer.calculateR2(xValues, yValues, logarithmicFunction));
        results.setCorrelation(null);
        return results;
    }
}
```

```java
@Component
public class PowerApproximation implements Approximation{
    private LinearApproximation linearApproximation;
    private ApproximationAnalyzer approximationAnalyzer;

    @Autowired
    public PowerApproximation(LinearApproximation linearApproximation, ApproximationAnalyzer approximationAnalyzer) {
        this.linearApproximation = linearApproximation;
        this.approximationAnalyzer = approximationAnalyzer;
    }
    @Override
    public ApproximationResults approximate(double[] xValues, double[] yValues) {
        int n = Math.min(xValues.length, yValues.length);
        double[] xValuesNew = new double[n];
        double[] yValuesNew = new double[n];
        for (int i = 0; i < n; i++) {
            if (yValues[i] > 0 && xValues[i] > 0) {
                xValuesNew[i] = Math.log(xValues[i]);
                yValuesNew[i] = Math.log(yValues[i]);
            } else {
                xValuesNew[i] = xValues[i];
                yValuesNew[i] = yValues[i];
            }
        }
        ApproximationResults results = linearApproximation.approximate(xValuesNew, yValuesNew);
        LinearFunction linearFunction = (LinearFunction) results.getFunction();
        PowerFunction powerFunction = new PowerFunction();
        powerFunction.setA(Math.exp(linearFunction.getB()));
        powerFunction.setB(linearFunction.getA());
        results.setFunction(powerFunction);
        results.setDeviation(approximationAnalyzer.calculateDeviation(xValues, yValues, powerFunction));
        results.setR2(approximationAnalyzer.calculateR2(xValues, yValues, powerFunction));
        results.setCorrelation(null);
        return results;
    }
}
```

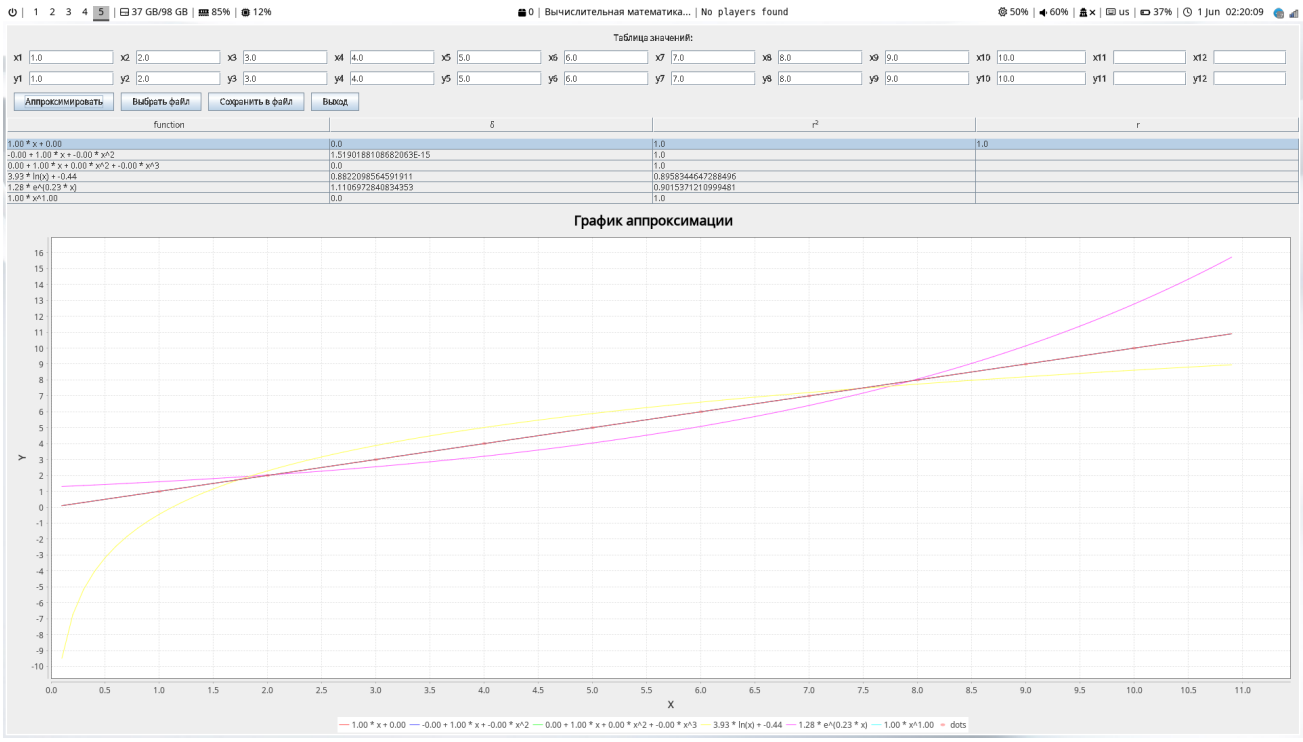
```java
@Component
public class ApproximationAnalyzer {
    public double calculateR2 (double[] xValues, double[] yValues, Function function) {
        int n = Math.min(xValues.length, yValues.length);
        double upper = 0;
        double lowerLeft = 0;
        double lowerRight = 0;
        for (int i = 0; i < n; i++) {
            upper += Math.pow(yValues[i] - function.apply(xValues[i]), 2);
            lowerLeft += Math.pow(function.apply(xValues[i]), 2);
            lowerRight += function.apply(xValues[i]);
        }
        double result = 1 - upper / (lowerLeft - Math.pow(lowerRight, 2) / n);
        if (Double.isNaN(result)) {
            result = 0;
        }
        return result;
    }

    public double calculateCorrelation(double[] xValues, double[] yValues) {
        int n = Math.min(xValues.length, yValues.length);
        double averageX = 0;
        double averageY = 0;
        for (int i = 0; i < n; i++) {
            averageX += xValues[i];
```

```
            averageY += yValues[i];
        }
        averageX /= n;
        averageY /= n;
        double upper = 0;
        double lowerLeft = 0;
        double lowerRight = 0;
        for (int i = 0; i < n; i++) {
            upper += (xValues[i] - averageX) * (yValues[i] - averageY);
            lowerLeft += Math.pow(xValues[i] - averageX, 2);
            lowerRight += Math.pow(yValues[i] - averageY, 2);
        }
        return upper / Math.sqrt(lowerLeft * lowerRight);
    }

    public double calculateDeviation(double[] xValues, double[] yValues, Function function) {
        int n = Math.min(xValues.length, yValues.length);
        double sum = 0;
        for (int i = 0; i < n; i++) {
            sum += Math.pow(function.apply(xValues[i]) - yValues[i], 2);
        }
        return Math.sqrt(sum / n);
    }
}
```

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.