**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по практической работе №5**

**по дисциплине «Вычислительная математика»**

**Тема: Интерполяционные и аппроксимационные формулы для равноотстоящих узлов.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Гайфутдинов А.Р. |
| Преподаватель |  | Попова Е.В. |

Санкт-Петербург

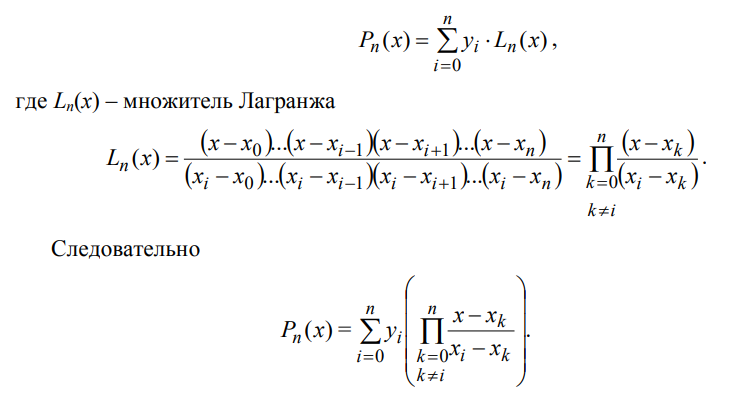
2025

**Цель работы:**

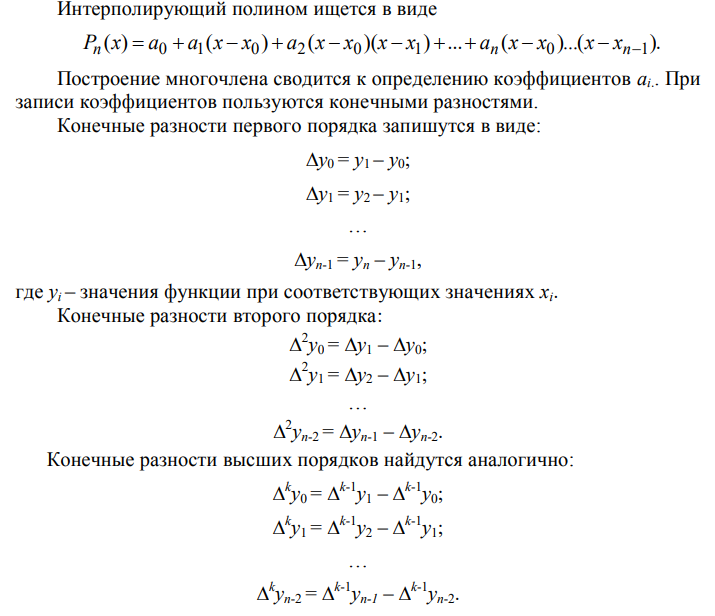
Исследование методов интерполяции и аппроксимации для равноотстоящих узлов с последующей реализацией на языке программирования.

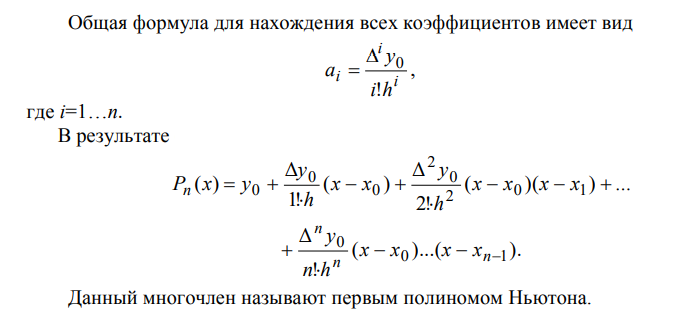
Значения функции заданы в точках необходимо найти промежуточные значения.

Интерполяционный многочлен Лагранжа.

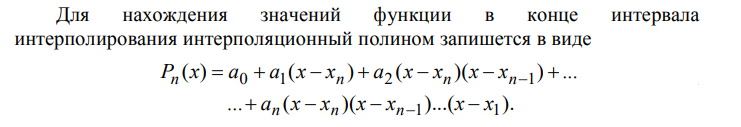


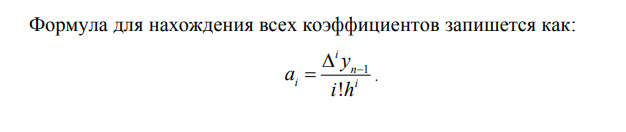
Первый интерполяционный многочлен Ньютона. Точка интерполирования находится в начале таблицы.

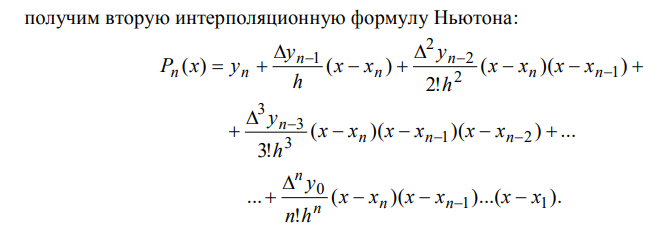




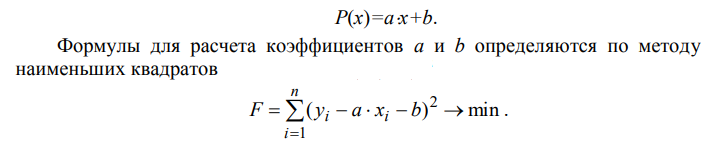
Второй интерполяционный многочлен Ньютона. Точка интерполирования находится в конце таблицы.

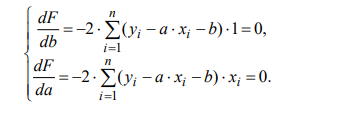


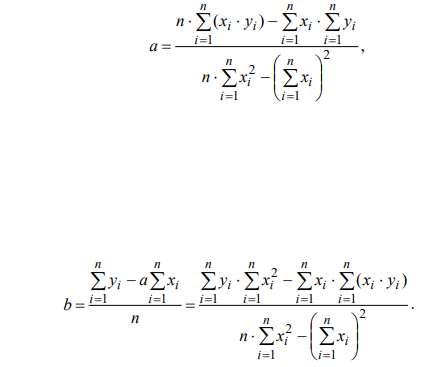




Аппроксимация функции. Необходимо найти эмпирическую формулу, значения которой при мало бы отличались от входных данных. В случае **линейной аппроксимации**





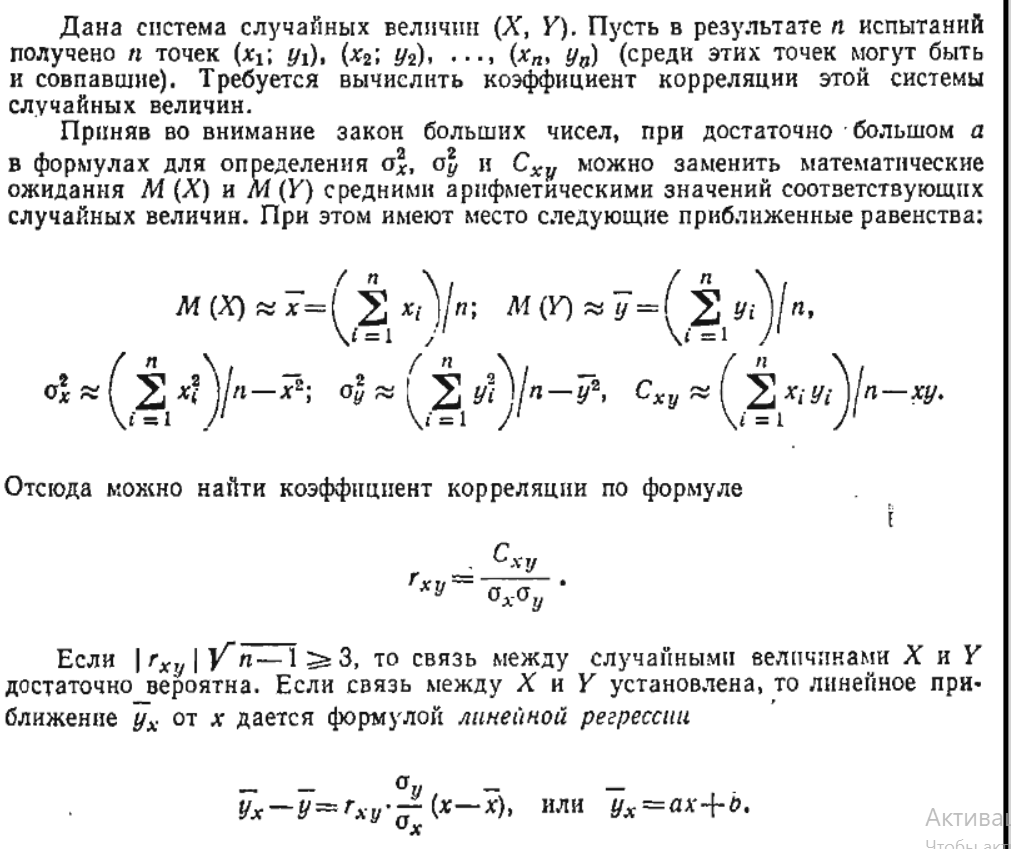


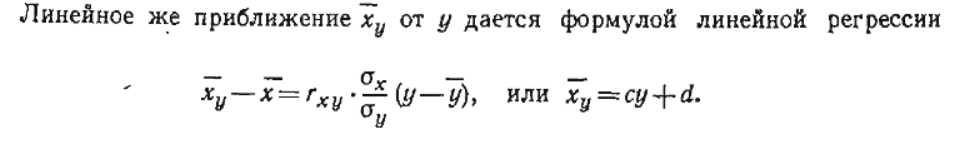
В случае **показательной аппроксимации**

В случае **экспоненциальной аппроксимации** .

**Оценка аппроксимации**:

Линии регрессии. Корреляция.





**Порядок выполнения работы.**

1. Выбрать 2 точки, не входящие в таблицу данных, ближе к началу и к концу таблицы.

2. Составить необходимые подпрограммы-функции на языке Java.

3. Составить головную программу, содержащую обращение к

соответствующим подпрограммам и осуществляющую печать результатов.

4. Используя многочлен Лагранжа, найти приближенные значения функции.

5. Используя многочлены Ньютона найти приближенные значения этих же выбранных точек. Построить графики интерполяционных кривых вместе с данными точками.

6. Произвести линейную, показательную, экспоненциальную аппроксимации, опираясь на метод наименьших квадратов. Вычислить значения функции в тех же точках. Построить графики аппроксимационных кривых вместе с данными точками. Рассчитать оценку аппроксимации.

7. По входным данным вычислить значения статистических параметров, проверить гипотезу о существовании связи между *Х* и *Y*, составить уравнения линий регрессий, графики этих линий и входных точек.

8. Составить сводную таблицу по результатам исследования. Сделать выводы.

**Вариант согласно номеру в списке - 6.**

|  |  |
| --- | --- |
| 6 | Columns 1 through 10  0.0363 -0.7447 -0.7196 -0.0220 1.0135 1.9967 2.5964 2.6232 2.0760 1.1515  Columns 11 through 17  0.2115 -0.2916 0.0628 1.4931 3.7932 6.0557 6.3377 |

**ВЫПОЛНЕНИЕ РАБОТЫ**

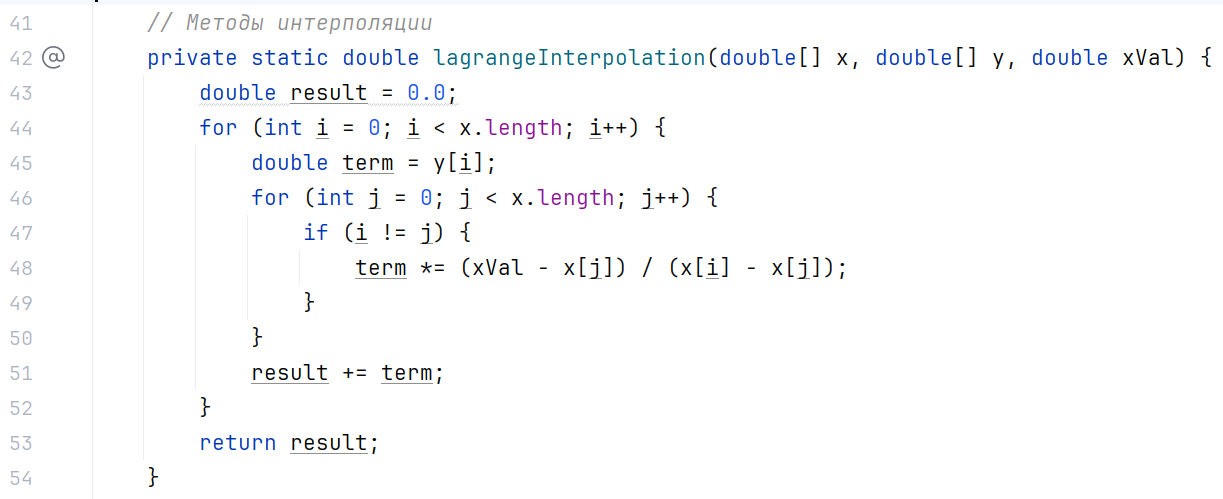
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *0.5 (1)* | *0.6* | *0.7* | *0.8* | *0.9 (5)* |
|  | **0.0363** | **-0.7447** | **-0.7196** | **-0.0220** | **1.0135** |
|  | *1.0 (6)* | *1.1* | *1.2* | *1.3* | *1.4 (10)* |
|  | **1.9967** | **2.5964** | **2.6232** | **2.0760** | **1.1515** |
|  | *1.5 (11)* | *1.6* | *1.7* | *1.8* | *1.9 (15)* |
|  | **0.2115** | **-0.2916** | **0.0628** | **1.4931** | **3.7932** |
|  | *2.0 (16)* | *2.1* |  |  |  |
|  | **6.0557** | **6.3377** |  |  |  |

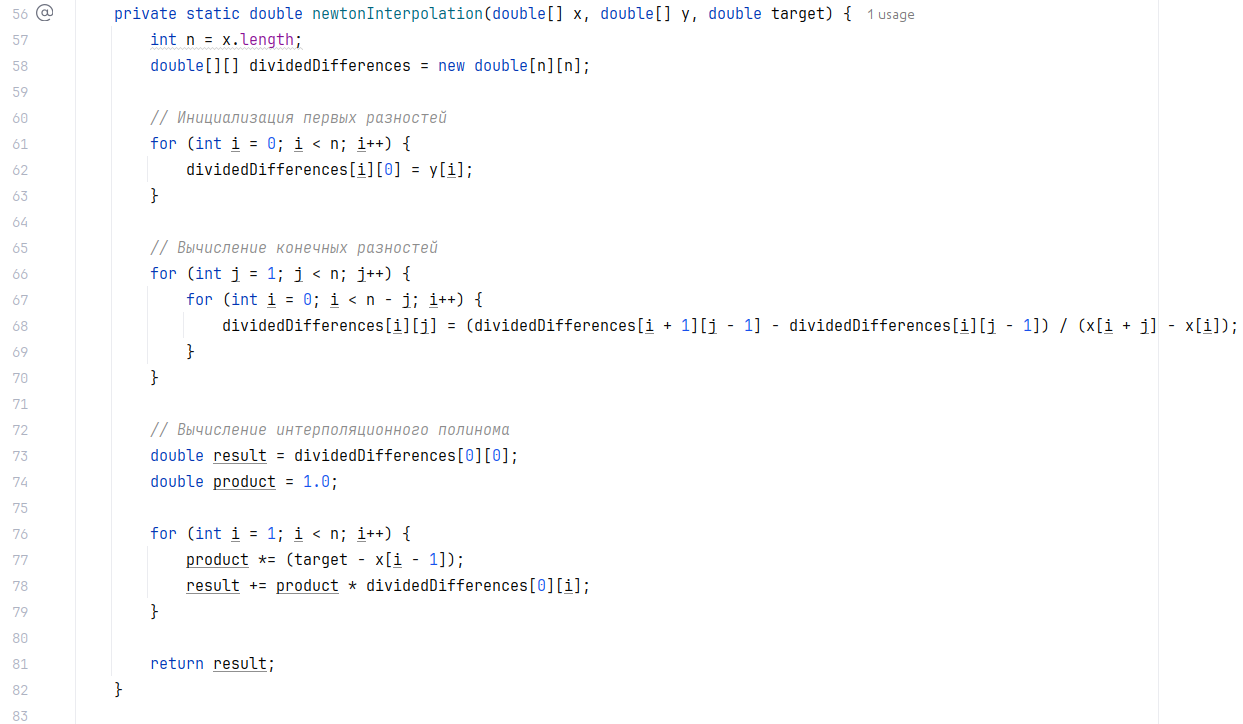
**1. Выбрать 2 точки, не входящие в таблицу данных, ближе к началу и к концу таблицы.**

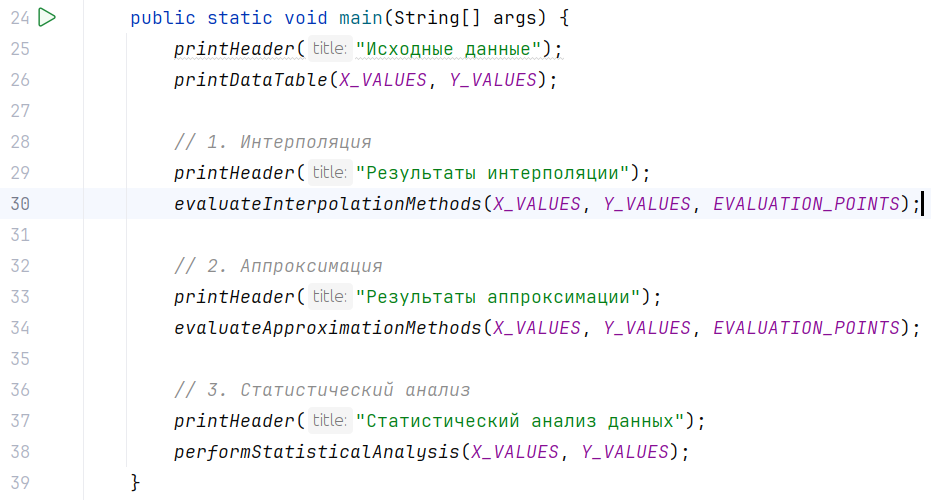
В качестве точек выберем = 0.55 и = 1.55.

**2.** **Составить необходимые подпрограммы-функции на языке Java.**

Составлены функции по вычислению значения интерполяционного многочлена Лагранжа и Ньютона:



3. Составить головную программу, содержащую обращение к соответствующим подпрограммам и осуществляющую печать результатов.



**4. Используя многочлен Лагранжа, найти приближенные значения функции.**

Точка интерполяции: x = 0,55

Метод Лагранжа: -0,437167

Точка интерполяции: x = 2,05

Метод Лагранжа: 6,629457

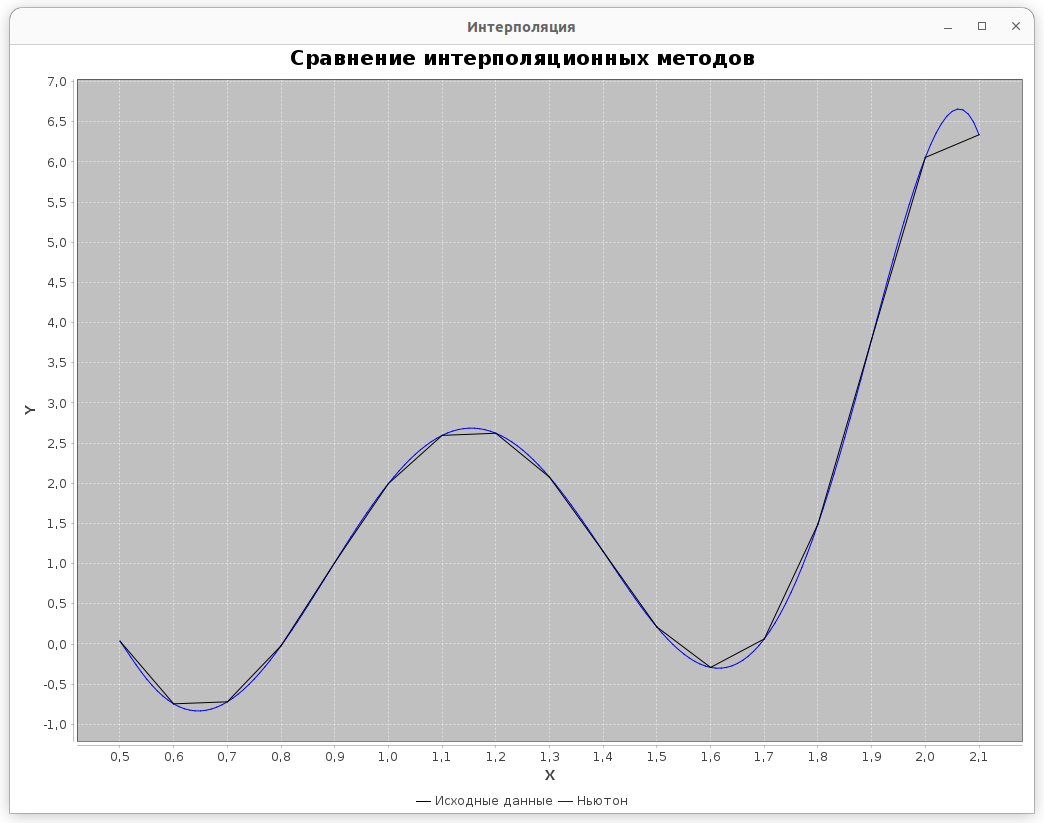
**5. Используя многочлены Ньютона найти приближенные значения этих же выбранных точек. Построить графики интерполяционных кривых вместе с данными точками.**

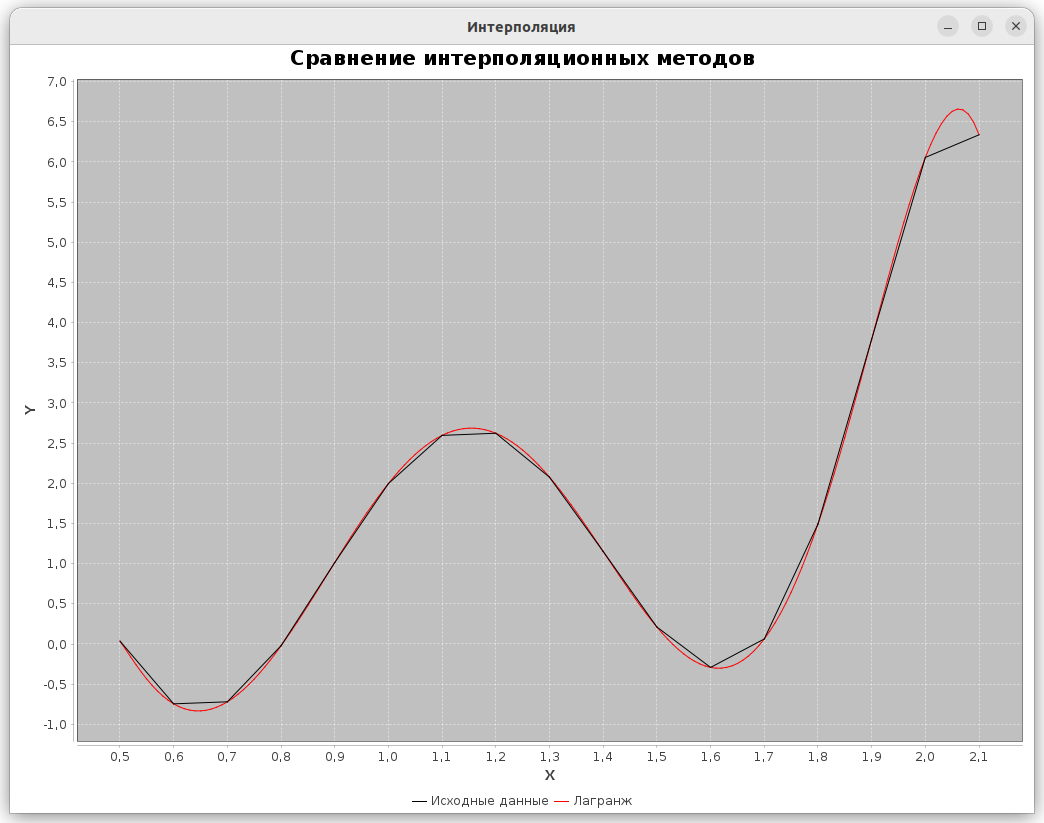
Точка интерполяции: x = 0,55

Метод Ньютона: -0,437167

Точка интерполяции: x = 2,05

Метод Ньютона: 6,629457





**6. Произвести линейную, показательную, экспоненциальную аппроксимации, опираясь на метод наименьших квадратов. Вычислить значения функции в тех же точках. Построить графики аппроксимационных кривых вместе с данными точками. Рассчитать оценку аппроксимации.**

x = 0,55

Линейная: -0,500258

Экспоненциальная: NaN (принимает отрицательные значения)

Степенная: NaN (принимает отрицательные значения)

x = 2,05

Линейная: 3,755517

Экспоненциальная: NaN (принимает отрицательные значения)

Степенная: NaN (принимает отрицательные значения)

Коэффициенты детерминации **аппроксимаций** (R²):

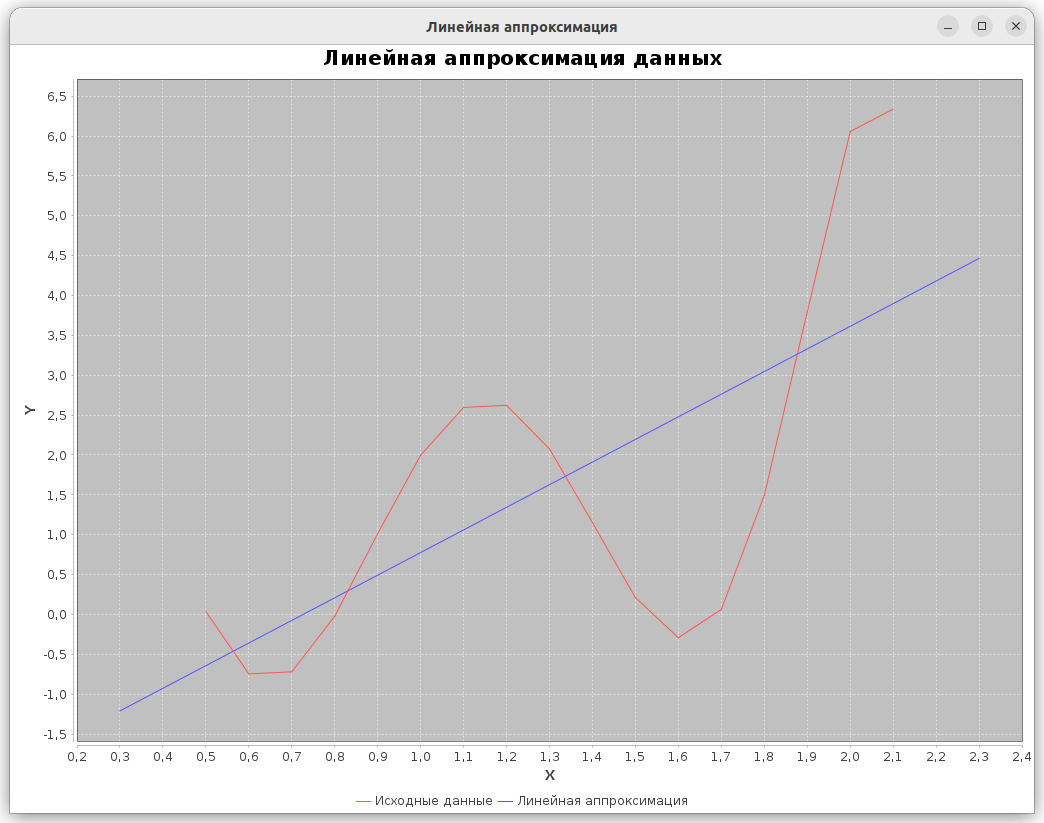
Линейная: 0,4444

Экспоненциальная: NaN (принимает отрицательные значения)

Степенная: NaN (принимает отрицательные значения)

Уравнение аппроксимации:

Линейная: y = 2,8372x + -2,0607

**7. По входным данным вычислить значения статистических параметров, проверить гипотезу о существовании связи между *Х* и *Y*, составить уравнения линий регрессий, графики этих линий и входных точек.**

Среднее X: 1,3000, Y: 1,6276

Дисперсия X: 0,2400, Y: 4,3470

Стандартное отклонение X: 0,4899, Y: 2,0850

Ковариация: 0,6809

Коэффициент корреляции: 0,6666

Гипотеза о наличии корреляции: ОТВЕРГНУТА

**8. Составить сводную таблицу по результатам исследования. Сделать выводы.**

|  |  |
| --- | --- |
| Параметр | Значение / Результат |
| Интерполяция  x = 0.55 | Лагранж: -0,437167  Ньютон: -0,437167 |
| Интерполяция  x = 2.05 | Лагранж: 6,629457  Ньютон: 6,629457 |
| Аппроксимация  x = 0.55 | Линейная: -0,500258  Степенная: NaN  Эксп: NaN |
| Аппроксимация  x = 2.05 | Линейная: 3,755517  Степенная: NaN  Эксп: NaN |
| R² (Линейная аппроксимация) | 0,4444 |
| R² (Экспоненциальная) | Не применима |
| R² (Степенная) | Не применима |
| Коэффициент корреляции (линейная) | 0,6666 |
| Гипотеза о связи (линейная) | ОТВЕРГНУТА |
| Регрессия Y на X (линейная) | Не применима |
| Регрессия X на Y (линейная) | Не применима |

На основе предоставленных данных можно сделать следующие выводы:

**Интерполяция (Лагранж и Ньютон):**

Оба метода дают идентичные результаты (значения совпадают до 6 знака после запятой), что подтверждает корректность реализации

Интерполяционные кривые точно проходят через все исходные точки

**Аппроксимация:**

Линейная:

Уравнение: y = 2.8372x - 2.0607

Качество аппроксимации:

R² = 0.4444 (умеренное качество, объясняет ~44% дисперсии данных)

Экспоненциальная и степенная модели не применимы из-за отрицательных значения в исходных данных (y=-0.7447 и др.).Экспоненциальная и степенная модели требуют строго положительных значений Y.

**Статистический анализ:**

Корреляция: коэффициент 0.6666 (умеренная положительная связь)

Гипотеза о значимой корреляции отвергнута связь статистически незначима на данном объеме данных)

**Итог:**

Данные демонстрируют нелинейный характер (подтверждается низким R² линейной модели и отвергнутой гипотезой о корреляции).

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл InterpolationAndApproximation.java:

package com.github.chelovekkrokant;import org.jfree.chart.ChartFactory;import org.jfree.chart.ChartFrame;import org.jfree.chart.JFreeChart;import org.jfree.chart.plot.PlotOrientation;import org.jfree.chart.plot.XYPlot;import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;import org.jfree.data.xy.XYSeries;import org.jfree.data.xy.XYSeriesCollection;import java.awt.\*;import java.util.Arrays;class Visualization { *// Исходные данные*private static final double[] *X\_VALUES* = InterpolationAndApproximation.*X\_VALUES*; private static final double[] *Y\_VALUES* = InterpolationAndApproximation.*Y\_VALUES*; static void launchLagrange() { *// Создаем наборы данных*XYSeries originalData = new XYSeries("Исходные данные"); XYSeries lagrangeSeries = new XYSeries("Лагранж"); XYSeries newtonSeries = new XYSeries("Ньютон"); *// Заполняем исходные данные*for (int i = 0; i < *X\_VALUES*.length; i++) { originalData.add(*X\_VALUES*[i], *Y\_VALUES*[i]); } *// Генерируем точки для интерполяционных кривых*double step = 0.01; *// Шаг для плавного графика*double start = *X\_VALUES*[0]; double end = *X\_VALUES*[*X\_VALUES*.length - 1]; for (double x = start; x <= end; x += step) { lagrangeSeries.add(x, InterpolationAndApproximation.*lagrangeInterpolation*(*X\_VALUES*, *Y\_VALUES*, x)); newtonSeries.add(x, InterpolationAndApproximation.*newtonInterpolation*(*X\_VALUES*, *Y\_VALUES*, x)); } *// Создаем коллекции данных*XYSeriesCollection dataset = new XYSeriesCollection(); dataset.addSeries(originalData); dataset.addSeries(lagrangeSeries); *//dataset.addSeries(newtonSeries);* *// Создаем график*JFreeChart chart = ChartFactory.*createXYLineChart*( "Сравнение интерполяционных методов", *// Заголовок*"X", *// Ось X*"Y", *// Ось Y*dataset, *// Данные*PlotOrientation.*VERTICAL*, true, *// Легенда*true, *// Подсказки*false *// URLs*); XYPlot plot = chart.getXYPlot(); XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.getRenderer(); renderer.setSeriesPaint(0, Color.*BLACK*); *// Исходные точки*renderer.setSeriesPaint(1, Color.*RED*); *// Лагранж* *// Отображаем график*ChartFrame frame = new ChartFrame("Интерполяция", chart); frame.pack(); frame.setVisible(true); } static void launchNewton() { *// Создаем наборы данных*XYSeries originalData = new XYSeries("Исходные данные"); XYSeries lagrangeSeries = new XYSeries("Лагранж"); XYSeries newtonSeries = new XYSeries("Ньютон"); *// Заполняем исходные данные*for (int i = 0; i < *X\_VALUES*.length; i++) { originalData.add(*X\_VALUES*[i], *Y\_VALUES*[i]); } *// Генерируем точки для интерполяционных кривых*double step = 0.01; *// Шаг для плавного графика*double start = *X\_VALUES*[0]; double end = *X\_VALUES*[*X\_VALUES*.length - 1]; for (double x = start; x <= end; x += step) { lagrangeSeries.add(x, InterpolationAndApproximation.*lagrangeInterpolation*(*X\_VALUES*, *Y\_VALUES*, x)); newtonSeries.add(x, InterpolationAndApproximation.*newtonInterpolation*(*X\_VALUES*, *Y\_VALUES*, x)); } *// Создаем коллекции данных*XYSeriesCollection dataset = new XYSeriesCollection(); dataset.addSeries(originalData); *//dataset.addSeries(lagrangeSeries);*dataset.addSeries(newtonSeries); *// Создаем график*JFreeChart chart = ChartFactory.*createXYLineChart*( "Сравнение интерполяционных методов", *// Заголовок*"X", *// Ось X*"Y", *// Ось Y*dataset, *// Данные*PlotOrientation.*VERTICAL*, true, *// Легенда*true, *// Подсказки*false *// URLs*); XYPlot plot = chart.getXYPlot(); XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.getRenderer(); renderer.setSeriesPaint(0, Color.*BLACK*); *// Исходные точки*renderer.setSeriesPaint(1, Color.*BLUE*); *// Ньютон* *// Отображаем график*ChartFrame frame = new ChartFrame("Интерполяция", chart); frame.pack(); frame.setVisible(true); } public static void launchApproximation() { *// 1. Вычисляем коэффициенты линейной аппроксимации*double[] coefficients = InterpolationAndApproximation.*linearLeastSquares*(*X\_VALUES*, *Y\_VALUES*); double slope = coefficients[0]; double intercept = coefficients[1]; System.*out*.printf("Уравнение прямой: y = %.4fx + %.4f%n", slope, intercept); *// 2. Создаем наборы данных*XYSeries originalData = new XYSeries("Исходные данные"); XYSeries approximationLine = new XYSeries("Линейная аппроксимация"); *// 3. Заполняем исходные данные*for (int i = 0; i < *X\_VALUES*.length; i++) { originalData.add(*X\_VALUES*[i], *Y\_VALUES*[i]); } *// 4. Строим линию аппроксимации (две точки для прямой)*double xStart = *X\_VALUES*[0] - 0.2; *// Немного выходим за границы*double xEnd = *X\_VALUES*[*X\_VALUES*.length - 1] + 0.2; approximationLine.add(xStart, slope \* xStart + intercept); approximationLine.add(xEnd, slope \* xEnd + intercept); *// 5. Создаем коллекцию данных*XYSeriesCollection dataset = new XYSeriesCollection(); dataset.addSeries(originalData); dataset.addSeries(approximationLine); *// 6. Создаем график*JFreeChart chart = ChartFactory.*createXYLineChart*( "Линейная аппроксимация данных", *// Заголовок*"X", *// Ось X*"Y", *// Ось Y*dataset, *// Данные*PlotOrientation.*VERTICAL*, true, *// Легенда*true, *// Подсказки*false *// URLs*);*//**// // 7. Настраиваем отображение точек**// chart.getXYPlot().getRenderer().setSeriesLinesVisible(0, false); // Только точки для исходных данных**// chart.getXYPlot().getRenderer().setSeriesShapesVisible(1, false); // Только линия для аппроксимации* *// 8. Отображаем график*ChartFrame frame = new ChartFrame("Линейная аппроксимация", chart); frame.pack(); frame.setVisible(true); }}public class InterpolationAndApproximation { private static final boolean *visualizationFlag* = false; *// Исходные данные*public static final double[] *X\_VALUES* = { 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1 }; public static final double[] *Y\_VALUES* = { 0.0363, -0.7447,-0.7196, -0.0220, 1.0135, 1.9967, 2.5964, 2.6232, 2.0760, 1.1515, 0.2115, -0.2916, 0.0628, 1.4931, 3.7932, 6.0557, 6.3377 }; *// Точки для интерполяции*private static final double[] *EVALUATION\_POINTS* = {0.55, 2.05}; public static void main(String[] args) { if (*visualizationFlag*) { Visualization.*launchLagrange*(); Visualization.*launchNewton*(); Visualization.*launchApproximation*(); } else { *printHeader*("Исходные данные"); *printDataTable*(*X\_VALUES*, *Y\_VALUES*); *// 1. Интерполяция* *printHeader*("Результаты интерполяции"); *evaluateInterpolationMethods*(*X\_VALUES*, *Y\_VALUES*, *EVALUATION\_POINTS*); *// 2. Аппроксимация* *printHeader*("Результаты аппроксимации"); *evaluateApproximationMethods*(*X\_VALUES*, *Y\_VALUES*, *EVALUATION\_POINTS*); *// 3. Статистический анализ* *printHeader*("Статистический анализ данных"); *performStatisticalAnalysis*(*X\_VALUES*, *Y\_VALUES*); } } *// Методы интерполяции*public static double lagrangeInterpolation(double[] x, double[] y, double xVal) { double result = 0.0; for (int i = 0; i < x.length; i++) { double term = y[i]; for (int j = 0; j < x.length; j++) { if (i != j) { term \*= (xVal - x[j]) / (x[i] - x[j]); } } result += term; } return result; } public static double newtonInterpolation(double[] x, double[] y, double target) { int n = x.length; double[][] dividedDifferences = new double[n][n]; *// Инициализация первых разностей*for (int i = 0; i < n; i++) { dividedDifferences[i][0] = y[i]; } *// Вычисление конечных разностей*for (int j = 1; j < n; j++) { for (int i = 0; i < n - j; i++) { dividedDifferences[i][j] = (dividedDifferences[i + 1][j - 1] - dividedDifferences[i][j - 1]) / (x[i + j] - x[i]); } } *// Вычисление интерполяционного полинома*double result = dividedDifferences[0][0]; double product = 1.0; for (int i = 1; i < n; i++) { product \*= (target - x[i - 1]); result += product \* dividedDifferences[0][i]; } return result; } *// Методы аппроксимации*public static double[] linearLeastSquares(double[] x, double[] y) { int n = x.length; double sumX = 0, sumY = 0, sumXY = 0, sumXX = 0; for (int i = 0; i < n; i++) { sumX += x[i]; sumY += y[i]; sumXY += x[i] \* y[i]; sumXX += x[i] \* x[i]; } double slope = (n \* sumXY - sumX \* sumY) / (n \* sumXX - sumX \* sumX); double intercept = (sumY - slope \* sumX) / n; return new double[]{slope, intercept}; } private static double linearApproximation(double[] x, double[] y, double xVal, double[] coeffs) { double[] params = *linearLeastSquares*(x, y); coeffs[0] = params[0]; coeffs[1] = params[1]; return params[0] \* xVal + params[1]; } private static double exponentialApproximation(double[] x, double[] y, double xVal, double[] coeffs) { double[] logY = new double[y.length]; for (int i = 0; i < y.length; i++) { logY[i] = Math.*log*(y[i]); } double[] params = *linearLeastSquares*(x, logY); coeffs[0] = Math.*exp*(params[1]); *// A = e^b*coeffs[1] = params[0]; *// k = a*return coeffs[0] \* Math.*exp*(coeffs[1] \* xVal); } private static double powerApproximation(double[] x, double[] y, double xVal, double[] coeffs) { double[] logX = new double[x.length]; double[] logY = new double[y.length]; for (int i = 0; i < x.length; i++) { logX[i] = Math.*log*(x[i]); logY[i] = Math.*log*(y[i]); } double[] params = *linearLeastSquares*(logX, logY); coeffs[0] = Math.*exp*(params[1]); *// A = e^b*coeffs[1] = params[0]; *// n = a*return coeffs[0] \* Math.*pow*(xVal, coeffs[1]); } *// Методы статистического анализа*private static double calculateMean(double[] data) { return Arrays.*stream*(data).average().orElse(0.0); } private static double calculateVariance(double[] data, double mean) { return Arrays.*stream*(data).map(v -> Math.*pow*(v - mean, 2)).sum() / data.length; } private static double calculateCovariance(double[] x, double[] y, double meanX, double meanY) { double sum = 0.0; for (int i = 0; i < x.length; i++) { sum += (x[i] - meanX) \* (y[i] - meanY); } return sum / x.length; } private static double calculateCorrelationCoefficient(double covariance, double stdDevX, double stdDevY) { return covariance / (stdDevX \* stdDevY); } private static double calculateRSquared(double[] yTrue, double[] yPred) { double ssRes = 0.0; double ssTot = 0.0; double yMean = *calculateMean*(yTrue); for (int i = 0; i < yTrue.length; i++) { ssRes += Math.*pow*(yTrue[i] - yPred[i], 2); ssTot += Math.*pow*(yTrue[i] - yMean, 2); } return 1 - (ssRes / ssTot); } *// Вспомогательные методы для вывода*private static void printHeader(String title) { System.*out*.println("\n" + "=".repeat(50)); System.*out*.println(title.toUpperCase()); System.*out*.println("=".repeat(50)); } private static void printDataTable(double[] x, double[] y) { System.*out*.printf("%-10s %-10s%n", "X", "Y"); System.*out*.println("-".repeat(20)); for (int i = 0; i < x.length; i++) { System.*out*.printf("%-10.4f %-10.4f%n", x[i], y[i]); } } private static void evaluateInterpolationMethods(double[] x, double[] y, double[] points) { for (double point : points) { System.*out*.printf("\nТочка интерполяции: x = %.2f%n", point); System.*out*.printf("Метод Лагранжа: %.6f%n", *lagrangeInterpolation*(x, y, point)); System.*out*.printf("Метод Ньютона: %.6f%n", *newtonInterpolation*(x, y, point)); } } private static void evaluateApproximationMethods(double[] x, double[] y, double[] points) { double[] linearCoeffs = new double[2]; double[] expCoeffs = new double[2]; double[] powerCoeffs = new double[2]; *// Вычисление коэффициентов* *linearApproximation*(x, y, 0, linearCoeffs); *exponentialApproximation*(x, y, 0, expCoeffs); *powerApproximation*(x, y, 0, powerCoeffs); *// Вывод уравнений*System.*out*.println("\nУравнения аппроксимации:"); System.*out*.printf("Линейная: y = %.4fx + %.4f%n", linearCoeffs[0], linearCoeffs[1]); System.*out*.printf("Экспоненциальная: y = %.4f \* e^(%.4fx)%n", expCoeffs[0], expCoeffs[1]); System.*out*.printf("Степенная: y = %.4f \* x^%.4f%n", powerCoeffs[0], powerCoeffs[1]); *// Оценка качества аппроксимации*System.*out*.println("\nКоэффициенты детерминации (R²):"); System.*out*.printf("Линейная: %.4f%n", *calculateRSquared*(y, *getLinearPredictions*(x, linearCoeffs))); System.*out*.printf("Экспоненциальная: %.4f%n", *calculateRSquared*(y, *getExponentialPredictions*(x, expCoeffs))); System.*out*.printf("Степенная: %.4f%n", *calculateRSquared*(y, *getPowerPredictions*(x, powerCoeffs))); *// Вычисление значений в заданных точках*System.*out*.println("\nЗначения в точках:"); for (double point : points) { System.*out*.printf("\nx = %.2f%n", point); System.*out*.printf("Линейная: %.6f%n", *linearApproximation*(x, y, point, linearCoeffs)); System.*out*.printf("Экспоненциальная: %.6f%n", *exponentialApproximation*(x, y, point, expCoeffs)); System.*out*.printf("Степенная: %.6f%n", *powerApproximation*(x, y, point, powerCoeffs)); } } private static void performStatisticalAnalysis(double[] x, double[] y) { double meanX = *calculateMean*(x); double meanY = *calculateMean*(y); double varianceX = *calculateVariance*(x, meanX); double varianceY = *calculateVariance*(y, meanY); double stdDevX = Math.*sqrt*(varianceX); double stdDevY = Math.*sqrt*(varianceY); double covariance = *calculateCovariance*(x, y, meanX, meanY); double correlation = *calculateCorrelationCoefficient*(covariance, stdDevX, stdDevY); System.*out*.printf("Среднее X: %.4f, Y: %.4f%n", meanX, meanY); System.*out*.printf("Дисперсия X: %.4f, Y: %.4f%n", varianceX, varianceY); System.*out*.printf("Стандартное отклонение X: %.4f, Y: %.4f%n", stdDevX, stdDevY); System.*out*.printf("Ковариация: %.4f%n", covariance); System.*out*.printf("Коэффициент корреляции: %.4f%n", correlation); *// Проверка гипотезы о корреляции*boolean isCorrelated = Math.*abs*(correlation) \* Math.*sqrt*(x.length - 1) >= 3; System.*out*.printf("\nГипотеза о наличии корреляции || abs(corellation) \* sqrt(n - 1) >= 3|| : %s%n\n", isCorrelated ? String.*format*("ПРИНЯТА : %.4f >= 3",Math.*abs*(correlation) \* Math.*sqrt*(x.length - 1)) : String.*format*("ОТВЕРГНУТА : %.4f < 3",Math.*abs*(correlation) \* Math.*sqrt*(x.length - 1))); *// Регрессионные уравнения*if (isCorrelated) { double[] yOnX = *calculateRegressionYX*(correlation, stdDevX, stdDevY, meanX, meanY); double[] xOnY = *calculateRegressionXY*(correlation, stdDevX, stdDevY, meanX, meanY); System.*out*.println("\nРегрессионные уравнения:"); System.*out*.printf("Y на X: y = %.4fx + %.4f%n", yOnX[0], yOnX[1]); System.*out*.printf("X на Y: x = %.4fy + %.4f%n", xOnY[0], xOnY[1]); } } private static double[] calculateRegressionYX(double r, double sigmaX, double sigmaY, double meanX, double meanY) { double slope = r \* sigmaY / sigmaX; double intercept = meanY - slope \* meanX; return new double[]{slope, intercept}; } private static double[] calculateRegressionXY(double r, double sigmaX, double sigmaY, double meanX, double meanY) { double slope = r \* sigmaX / sigmaY; double intercept = meanX - slope \* meanY; return new double[]{slope, intercept}; } private static double[] getLinearPredictions(double[] x, double[] coeffs) { double[] pred = new double[x.length]; for (int i = 0; i < x.length; i++) { pred[i] = coeffs[0] \* x[i] + coeffs[1]; } return pred; } private static double[] getExponentialPredictions(double[] x, double[] coeffs) { double[] pred = new double[x.length]; for (int i = 0; i < x.length; i++) { pred[i] = coeffs[0] \* Math.*exp*(coeffs[1] \* x[i]); } return pred; } private static double[] getPowerPredictions(double[] x, double[] coeffs) { double[] pred = new double[x.length]; for (int i = 0; i < x.length; i++) { pred[i] = coeffs[0] \* Math.*pow*(x[i], coeffs[1]); } return pred; }}