**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,   
МЕХАНИКИ И ОПТИКИ»**

**Факультет Программной Инженерии и Компьютерной Техники**

**Дисциплина:**

**«*Вычислительная математика*»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4  
*«Аппроксимация функции методом наименьших квадратов»***

***Вариант 3***

**Выполнил:**

Студент гр. P32151 *Горинов Даниил Андреевич*

**Проверил:**

*Машина Екатерина Алексеевна*

Санкт-Петербург

2023г.

Цель лабораторной работы:

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

Порядок выполнения лабораторной работы:

**Программная реализация задачи:**

Для исследования использовать:

* линейную функцию,
* полиномиальную функцию 2-й степени,
* полиномиальную функцию 3-й степени,
* экспоненциальную функцию,
* логарифмическую функцию,
* степенную функцию.

*Методика проведения исследования:*

1. Вычислить меру отклонения для всех исследуемых функций;
2. Уточнить значения коэффициентов эмпирических функций, минимизируя функцию S;
3. Сформировать массивы предполагаемых эмпирических зависимостей;
4. Определить среднеквадратичное отклонение для каждой аппроксимирующей функции. Выбрать наименьшее значение n, следовательно, наилучшее приближение;
5. Построить графики полученных эмпирических функций.

*Задание:*

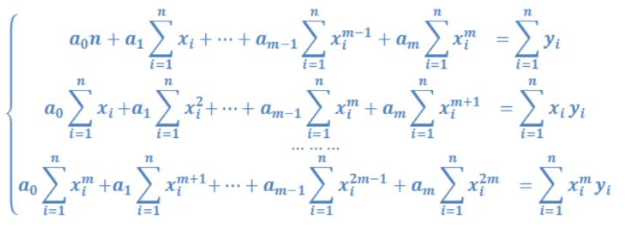
1. Предусмотреть ввод исходных данных из файла/консоли (таблица должна содержать от 8 до 12 точек);
2. Реализовать метод наименьших квадратов, исследуя все указанные функции;
3. Предусмотреть вывод результатов в файл/консоль: коэффициенты аппроксимирующих функций, среднеквадратичное отклонение, массивы значений ;
4. Для линейной зависимости вычислить коэффициент корреляции Пирсона;
5. Программа должна отображать наилучшую аппроксимирующую функцию;
6. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом);
7. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных;**Вычислительная реализация задачи:**
8. Сформировать таблицу табулирования заданной функции на указанном интервале;
9. Построить линейное и квадратичное приближения по 11 точкам заданного интервала;
10. Найти среднеквадратические отклонения для каждой аппроксимирующей функции. Ответы дать с тремя знаками после запятой;
11. Выбрать наилучшее приближение;
12. Построить графики заданной функции, а также полученные линейное и квадратичное приближения;
13. Привести в отчете подробные вычисления.Рабочие формулы методов:

Параметры эмпирической формулы находятся из условия минимума функции . Так как здесь параметры выступают в роли независимых переменных функции *S*, то ее минимум найдем, приравнивая к нулю частные производные по этим переменным.

Изображение выглядит как текст

Автоматически созданное описание

Преобразуем полученную линейную систему уравнений: раскроем скобки и перенесем свободные слагаемые в правую часть выражения:



В матричном виде:

Изображение выглядит как диаграмма, схематичный

Автоматически созданное описание

Вычислительная часть:

Функция: , исследуемый интервал:

*Составим таблицу с точками и значениями функции в этих точках на промежутке x∈[-2,0] с шагом 0.2:*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
|  | -0.421 | -0.533 | -0.670 | -0.819 | -0.946 | -1.000 | -0.939 | -0.767 | -0.529 | -0.267 | 0 |

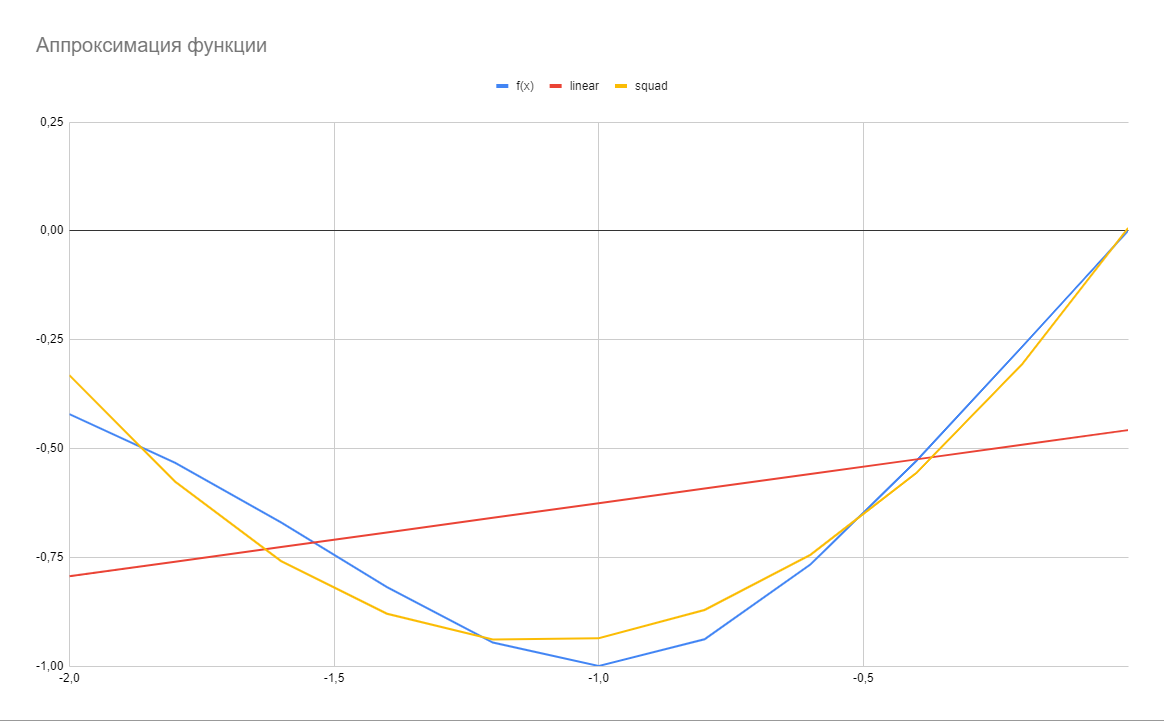
*Линейная аппроксимация:*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
|  | -0.421 | -0.533 | -0.670 | -0.819 | -0.946 | -1.000 | -0.939 | -0.767 | -0.529 | -0.267 | 0 |
|  | -0.794 | -0.760 | -0.727 | -0.693 | -0.660 | -0.626 | -0.592 | -0.559 | -0.525 | -0.492 | -0.458 |
|  | -0.373 | -0.227 | -0.057 | 0.125 | 0.286 | 0.374 | 0.346 | 0.208 | 0.004 | -0.225 | -0.458 |
|  | 0.139 | 0.052 | 0.003 | 0.016 | 0.082 | 0.140 | 0.120 | 0.043 | 0.000 | 0.051 | 0.210 |

*Квадратичная аппроксимация: 5.723 = 7.631, -5.167 = -6.89, -7.55=-10.066*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
|  | -0.421 | -0.533 | -0.670 | -0.819 | -0.946 | -1.000 | -0.939 | -0.767 | -0.529 | -0.267 | 0 |
|  | -0.332 | -0.576 | -0.759 | -0.880 | -0.939 | -0.936 | -0.871 | -0.745 | -0.556 | -0.306 | 0.006 |
|  | 0.089 | -0.043 | -0.089 | -0.061 | 0.007 | 0.064 | 0.067 | 0.022 | -0.027 | -0.040 | 0.006 |
|  | 0.008 | 0.002 | 0.008 | 0.004 | 0.000 | 0.004 | 0.005 | 0.000 | 0.001 | 0.002 | 0.000 |

*График полученных функций:*



Листинг программы:

1. public ApproximationResult cubicApproximation(double[][] functionTable) {
2. double x\_sum = 0, x2\_sum = 0, x3\_sum = 0, x4\_sum = 0, x5\_sum = 0, x6\_sum = 0,
3. y\_sum = 0, xy\_sum = 0, x2y\_sum = 0, x3y\_sum = 0;
4. for (double[] doubles : functionTable) {
5. x\_sum += doubles[0];
6. x2\_sum += Math.pow(doubles[0], 2);
7. x3\_sum += Math.pow(doubles[0], 3);
8. x4\_sum += Math.pow(doubles[0], 4);
9. x5\_sum += Math.pow(doubles[0], 5);
10. x6\_sum += Math.pow(doubles[0], 6);
11. y\_sum += doubles[1];
12. xy\_sum += doubles[0] \* doubles[1];
13. x2y\_sum += Math.pow(doubles[0], 2) \* doubles[1];
14. x3y\_sum += Math.pow(doubles[0], 3) \* doubles[1];
15. }
16. double[][] matrix = new double[][] {
17. {functionTable.length, x\_sum, x2\_sum, x3\_sum},
18. {x\_sum, x2\_sum, x3\_sum, x4\_sum},
19. {x2\_sum, x3\_sum, x4\_sum, x5\_sum},
20. {x3\_sum, x4\_sum, x5\_sum, x6\_sum}
21. };
22. double[] constants = new double[] {y\_sum, xy\_sum, x2y\_sum, x3y\_sum};
23. double[] solution = solveLinearSystem(matrix, constants);
24. reverseArray(solution);
25. Function<Double, Double> function = coefficientsToCubicFunction(solution);
26. double deviation = deviationMeasure(functionTable, function);
27. return new ApproximationResult(ApproximationType.CUBIC, solution, function, deviation);
28. }
29. public ApproximationResult squareApproximation(double[][] functionTable) {
30. double x\_sum = 0, x2\_sum = 0, x3\_sum = 0, x4\_sum = 0,
31. y\_sum = 0, xy\_sum = 0, x2y\_sum = 0;
32. for (double[] doubles : functionTable) {
33. x\_sum += doubles[0];
34. x2\_sum += Math.pow(doubles[0], 2);
35. x3\_sum += Math.pow(doubles[0], 3);
36. x4\_sum += Math.pow(doubles[0], 4);
37. y\_sum += doubles[1];
38. xy\_sum += doubles[0] \* doubles[1];
39. x2y\_sum += Math.pow(doubles[0], 2) \* doubles[1];
40. }
41. double[][] matrix = new double[][] {
42. {functionTable.length, x\_sum, x2\_sum},
43. {x\_sum, x2\_sum, x3\_sum},
44. {x2\_sum, x3\_sum, x4\_sum}
45. };
46. double[] constants = new double[] {y\_sum, xy\_sum, x2y\_sum};
47. double[] solution = solveLinearSystem(matrix, constants);
48. reverseArray(solution);
49. Function<Double, Double> function = coefficientsToSquareFunction(solution);
50. double deviation = deviationMeasure(functionTable, function);
51. return new ApproximationResult(ApproximationType.QUADRATIC, solution, function, deviation);
52. }
53. public ApproximationResult linearApproximation(double[][] functionTable) {
54. double x\_sum = 0, x2\_sum = 0, y\_sum = 0, xy\_sum = 0;
55. for (double[] doubles : functionTable) {
56. x\_sum += doubles[0];
57. x2\_sum += Math.pow(doubles[0], 2);
58. y\_sum += doubles[1];
59. xy\_sum += doubles[0] \* doubles[1];
60. }
61. double[][] matrix = {
62. {x2\_sum, x\_sum},
63. {x\_sum, functionTable.length}
64. };
65. double[] constants = {
66. xy\_sum, y\_sum
67. };
68. double[] solution = solveLinearSystem(matrix, constants);
69. Function<Double, Double> function = coefficientsToLinearFunction(solution);
70. double deviation = deviationMeasure(functionTable, function);
71. return new ApproximationResult(ApproximationType.LINEAR, solution, function, deviation, linearCorrelation(functionTable));
72. }
73. public ApproximationResult exponentialApproximation(double[][] functionTable) {
74. double[][] modifiedFunctionTable = Arrays.stream(functionTable).map(double[]::clone).toArray(double[][]::new);
75. for (double[] xy: modifiedFunctionTable) {
76. if (xy[1] <= 0) continue;
77. xy[1] = Math.log(xy[1]);
78. }
79. ApproximationResult linear = linearApproximation(modifiedFunctionTable);
80. double[] coefficients = linear.getCoefficients();
81. coefficients[1] = Math.exp(coefficients[1]);
82. Function<Double, Double> f = coefficientsToExpFunction(coefficients);
83. return new ApproximationResult(ApproximationType.EXPONENTIAL, coefficients, f, deviationMeasure(functionTable, f));
84. }
85. public ApproximationResult logarithmicApproximation(double[][] functionTable) {
86. double[][] modifiedFunctionTable = Arrays.stream(functionTable).map(double[]::clone).toArray(double[][]::new);
87. for (double[] xy: modifiedFunctionTable) {
88. xy[0] = Math.log(xy[0]);
89. }
90. ApproximationResult linear = linearApproximation(modifiedFunctionTable);
91. double[] coefficients = linear.getCoefficients();
92. Function<Double, Double> f = coefficientsToLogFunction(coefficients);
93. return new ApproximationResult(ApproximationType.LOGARITHMIC, coefficients, f, deviationMeasure(functionTable, f));
94. }
95. public ApproximationResult powerApproximation(double[][] functionTable) {
96. double[][] modifiedFunctionTable = Arrays.stream(functionTable).map(double[]::clone).toArray(double[][]::new);
97. for (double[] xy: modifiedFunctionTable) {
98. xy[0] = Math.log(xy[0]);
99. xy[1] = Math.log(xy[1]);
100. }
101. ApproximationResult linear = linearApproximation(modifiedFunctionTable);
102. double[] coefficients = linear.getCoefficients();
103. coefficients[1] = Math.exp(coefficients[1]);
104. Function<Double, Double> f = coefficientsToPowerFunction(coefficients);
105. return new ApproximationResult(ApproximationType.POWER, coefficients, f, deviationMeasure(functionTable, f));
106. }

Результаты выполнения программы:

Введите название файла или 0 для ввода с клавиатуры:

0

Введите название файла или 0 для вывода в консоль:

0

Введите количество пар (x, y) (не менее 8)

8

1.2 7.4

2.9 9.5

4.1 11.1

5.5 12.9

6.7 14.6

7.8 17.3

9.2 18.2

10.3 20.7

Approximation result.

Type: LINEAR

Function: 1,454330x + 5,291060

Deviation: 1,345854

Correlation: 0.9954179478701582

Approximation result.

Type: QUADRATIC

Function: 0,025974x^2 + 1,152563x + 5,943053

Deviation: 1,015890

Approximation result.

Type: EXPONENTIAL

Function: 6,839635e^(0,111077x)

Deviation: 2,718870

Approximation result.

Type: LOGARITHMIC

Function: 6,008624lnx + 4,295866

Deviation: 37,832367

Approximation result.

Type: POWER

Function: 6,128671x^(0,479894)

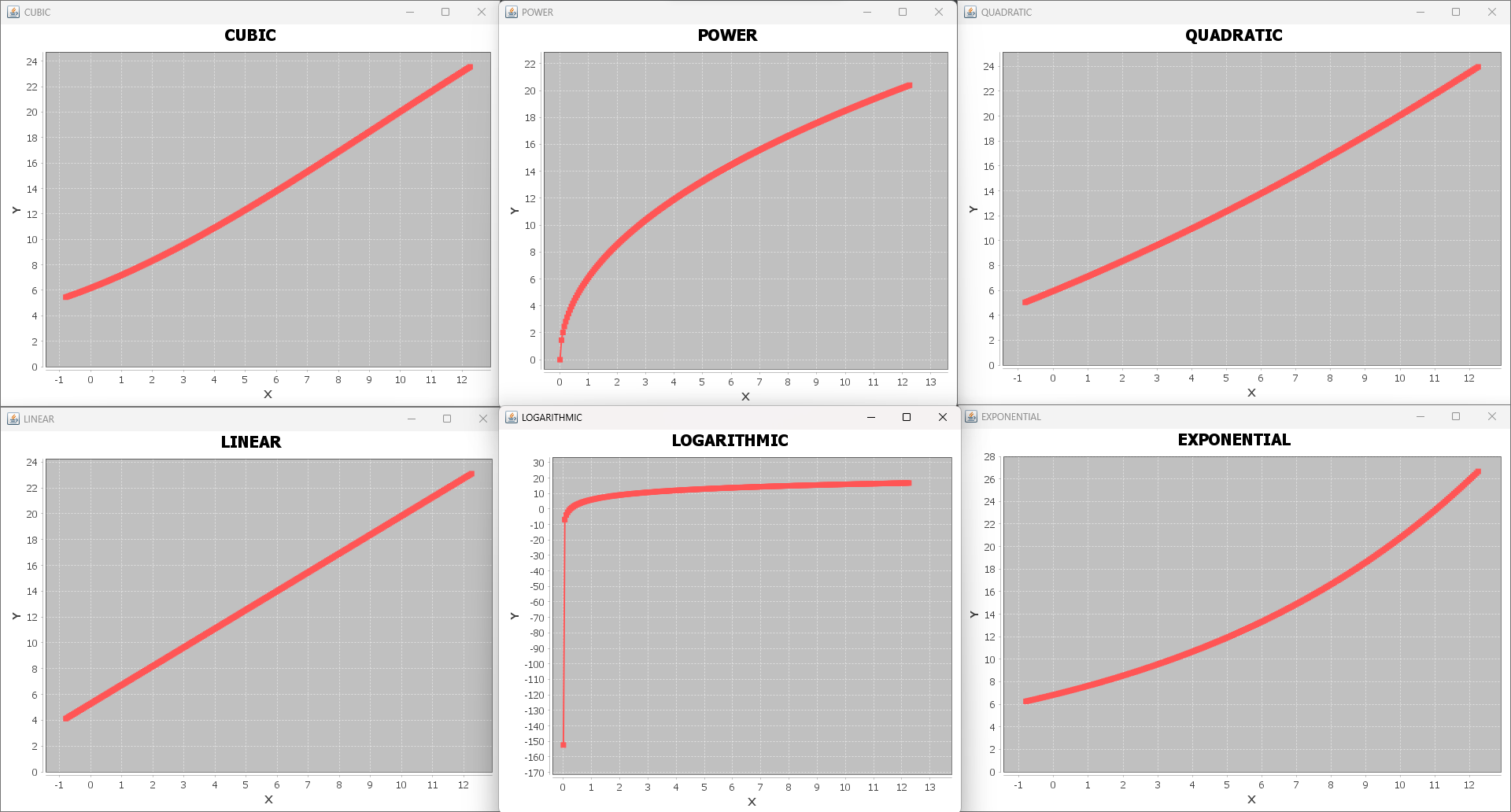
Deviation: 8,046184

Approximation result.

Type: CUBIC

Function: -0,002371x^3 + 0,066875x^2 + 0,954754x + 6,177879

Deviation: 0,999587



Вывод:

В результате проведения данной лабораторной работы, я ознакомился с методом наименьших квадратов и успешно реализовал его на языке Python. Метод наименьших квадратов обладает несколькими преимуществами, такими как: простота расчетов - требуется только найти коэффициенты, простота функции и широкий выбор возможных аппроксимирующих функций. Однако, основным недостатком метода наименьших квадратов является его чувствительность к резким выбросам, которые могут присутствовать в исходных данных.