НАЦИОНАЛЬНЫЙ

ИССЛЕДОВАТЕЛЬСКИЙ

УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 - Программная инженерия

Дисциплина – Вычислительная математика

Лабораторная работа №4

Вариант №11

Выполнил: Мухсинов С.П

Группа: P3217

Преподаватель: Малышева Т.А

Санкт-Петербург

2024

**Цель работы**:

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

**Задание**:

**Вычислительная реализация задачи:**

1. Сформировать таблицу табулирования заданной функции на указанном интервале.
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.
6. Привести в отчете подробные вычисления.

**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 и, следовательно, наилучшее приближение.
5. Построить графики полученных эмпирических функций.

**Задание:**

1. Предусмотреть ввод исходных данных из файла/консоли.
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. Вычислить коэффициент детерминации, программа должна выводить соответствующее сообщение в зависимости от полученного значения
6. Программа должна отображать наилучшую аппроксимирующую функцию.
7. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом).
8. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных.

**Используемые формулы и методы**

Изображение выглядит как текст, снимок экрана, Шрифт, рукописный текст

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, рукописный текст

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, линия

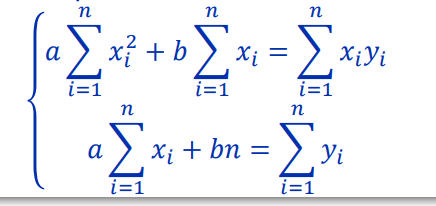
Автоматически созданное описание

**Вычислительная реализация**

**Сформировать таблицу табулирования заданной функции на указанном интервале:**

|  |  |
| --- | --- |
| **X** | **Y** |
| -2 | -0,3737 |
| -1,8 | **-**0,4187 |
| -1,6 | -0,4557 |
| -1,4 | -0,4716 |
| -1,2 | -0,4589 |
| -1 | -0,4167 |
| -0,8 | -0,3506 |
| -0,6 | -0,2696 |
| -0,4 | -0,1814 |
| -0,2 | -0,0909 |
| 0 | 0 |

**Линейное приближение:**

****

**Правило Крамера:**

**Полученное приближение:**

**Среднеквадратичное отклонение:**

**Квадратичное приближение:**

**Решив следующую систему:**

**Изображение выглядит как текст, Шрифт, рукописный текст, диаграмма

Автоматически созданное описание**

**Получим:**

**Полученное приближение:**

**Среднеквадратичное отклонение:**

**Так как , то наилучшим является квадратичное приближение.**

**Графики:**

***Изображение выглядит как линия, График, диаграмма, текст

Автоматически созданное описание***

**Программная реализация**

Экспоненциальная функция

public Exponential(double[] x, double[] y) {  
 super(x, y);  
 }  
  
 @Override  
 public Result solve() {  
 boolean isNan = false;  
 for(int i = 0; i < n; i++){  
 if(this.y[i] > 0){  
 this.y[i] = Math.*log*(this.y[i]);  
 }else{  
 isNan = true;  
 }  
 }  
 double[] r = new Linear(x, y).solve().getR();  
 r[1] = Math.*exp*(r[1]);  
 MathFunction f = new MathFunction(x -> r[1] \* Math.*exp*(x\*r[0]));  
 String function = r[1] + " \* " + "e^(" + r[0] + ")x";  
 return new Result(MethodType.*EXPONENTIAL*, function, f, deviation(f), determination(f), isNan);  
 }  
}

Линейная функция

public Linear(double[] x, double[] y){  
 super(x, y);  
 this.a = new double[2];  
 }  
 @Override  
 public Result solve() {  
 double a, b;  
 double sx = 0, sxx = 0, sy = 0, sxy = 0;  
 for (int i = 0; i < n; i++) {  
 sx += x[i];  
 sxx += x[i] \* x[i];  
 sy += y[i];  
 sxy += x[i] \* y[i];  
  
 }  
 double delta = sxx \* n - sx \* sx;  
 double delta1 = sxy \* n - sx \* sy;  
 double delta2 = sxx \* sy - sx \* sxy;  
 a = delta1 / delta;  
 b = delta2 / delta;  
 MathFunction f = new MathFunction(x -> a \* x + b);  
 String function = a + "x" + " + " + b;  
 return new Result(MethodType.*LINEAR*, function, f, deviation(f), correlation(), new double[]{a, b}, determination(f));  
 }  
  
 private double correlation() {  
 double x\_avg = Arrays.*stream*(x).sum() / n;  
 double y\_avg = Arrays.*stream*(y).sum() / n;  
 double numerator = 0;  
 double denominator1 = 0;  
 double denominator2 = 0;  
 for (int i = 0; i < n; i++) {  
 numerator += (x[i] - x\_avg) \* (y[i] - y\_avg);  
 denominator1 += Math.*pow*(x[i] - x\_avg, 2);  
 denominator2 += Math.*pow*(y[i] - y\_avg, 2);  
 }  
 double denominator = Math.*sqrt*(denominator1\*denominator2);  
 return numerator/denominator;  
 }  
}

Логарифмическая функция

public Logarithmic(double[] x, double[] y) {  
 super(x, y);  
 }  
  
 @Override  
 public Result solve() {  
 boolean isNan = false;  
 for(int i = 0; i < n; i++){  
 if(this.x[i] < 0){  
 isNan = true;  
 }  
 this.x[i] = Math.*log*(this.x[i]);  
 }  
 double[] r = new Linear(x,y).solve().getR();  
 MathFunction f = new MathFunction(x -> r[0] \* Math.*log*(x) + r[1]);  
 String function = r[0] + "ln(x)" + " + " + r[1];  
 double deviation = deviation(f);  
 return new Result(MethodType.*LOGARITHMIC*, function, f, deviation(f), determination(f), isNan);  
 }  
}

Кубическая функция

public Cubic(double[] x, double[] y) {  
 super(x, y);  
 }  
  
 @Override  
 public Result solve() {  
 double sx = 0, sx2 = 0, sx3 = 0, sx4 = 0, sx5 = 0, sx6 = 0;  
 double sy = 0, sxy = 0, sx2y = 0, sx3y = 0;  
 for(int i = 0; i < n; i++){  
 sx += x[i];  
 sx2 += Math.*pow*(x[i], 2);  
 sx3 += Math.*pow*(x[i], 3);  
 sx4 += Math.*pow*(x[i], 4);  
 sx5 += Math.*pow*(x[i], 5);  
 sx6 += Math.*pow*(x[i], 6);  
 sy += y[i];  
 sxy += x[i]\*y[i];  
 sx2y += Math.*pow*(x[i], 2) \* y[i];  
 sx3y += Math.*pow*(x[i], 3) \* y[i];  
 }  
 double[][] x\_i = {  
 {n, sx, sx2, sx3},  
 {sx, sx2, sx3, sx4},  
 {sx2, sx3, sx4, sx5},  
 {sx3, sx4, sx5, sx6}  
 };  
 double[] y\_i = {sy, sxy, sx2y, sx3y};  
 double[] result = solveLinearSystem(x\_i, y\_i);  
 double a0 = result[0], a1 = result[1], a2 = result[2], a3 = result[3];  
 MathFunction f= new MathFunction(x -> a3\*x\*x\*x + a2\*x\*x + a1\*x + a0);  
 String function = a3 + "x^3" + " + " + a2 +"x^2" + " + " + a3 + "x" + " + " + a0;  
 return new Result(MethodType.*CUBIC*, function,f, deviation(f), determination(f), false);  
 }  
}

Степенная функция

public Power(double[] x, double[] y) {  
 super(x, y);  
 }  
  
 @Override  
 public Result solve() {  
 boolean isNan = false;  
 for(int i = 0; i < n;i++){  
 if(x[i] < 0 || y[i] < 0){  
 isNan = true;  
 }  
 this.x[i] = Math.*log*(this.x[i]);  
 this.y[i] = Math.*log*(this.y[i]);  
 }  
 double[] r = new Linear(x , y).solve().getR();  
 r[1] = Math.*exp*(r[1]);  
 MathFunction f = new MathFunction(x -> r[1]\*Math.*pow*(x, r[0]));  
 String function = r[1] + " \* " + "x^(" + r[0] + ")";  
 return new Result(MethodType.*POWER*, function, f, deviation(f), determination(f), isNan);  
 }  
}

Квадратичная функция

public Quadratic(double[] x, double[] y) {  
 super(x, y);  
 }  
  
 @Override  
 public Result solve() {  
 double sx = 0, sx2 = 0, sx3 = 0, sx4 = 0;  
 double sy = 0, sxy = 0, sx2y = 0;  
 for(int i = 0; i < n; i++){  
 sx += x[i];  
 sx2 += Math.*pow*(x[i], 2);  
 sx3 += Math.*pow*(x[i], 3);  
 sx4 += Math.*pow*(x[i], 4);  
 sy += y[i];  
 sxy += x[i]\*y[i];  
 sx2y += Math.*pow*(x[i], 2) \* y[i];  
 }  
 double[][] x\_i = {{n, sx, sx2}, {sx, sx2, sx3}, {sx2, sx3, sx4}};  
 double[] y\_i = {sy, sxy, sx2y};  
 double[] result = solveLinearSystem(x\_i, y\_i);  
 double a0 = result[0], a1 = result[1], a2 = result[2];  
 MathFunction f= new MathFunction(x -> a2\*x\*x + a1\*x + a0);  
 String function = a2 + "x^2" + " + " + a1 +"x" + " + " + a0;  
 return new Result(MethodType.*QUADRATIC*, function, f, deviation(f), determination(f), false);  
 }  
}

**Результаты работы программы**

**Входные данные:**

**Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание**

**Результат:**

**Изображение выглядит как текст, снимок экрана, документ, меню

Автоматически созданное описание**

**Входные данные:**

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание**

**Результат:**

**Изображение выглядит как текст, снимок экрана, документ, Шрифт

Автоматически созданное описание**

**Входные данные:**

**Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание**

**Результат:**

**Изображение выглядит как текст, Шрифт, линия, алгебра

Автоматически созданное описание**

**Вывод**

В ходе выполнения лабораторной работы мы познакомились с различными способами аппроксимации функции и реализовали программу, находящую приближение функции через метод наименьших квадратов.