Университет ИТМО, факультет ПИиКТ

Лабораторная работа №5

Дисциплина: Вычислительная математика

Вариант 19

Выполнил: Щелыкалов Виктор

Группа: Р3214

Преподаватель:

Малышева Татьяна Алексеевна

Санкт-Петербург, 2020 год

**Цель работы:** решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек.

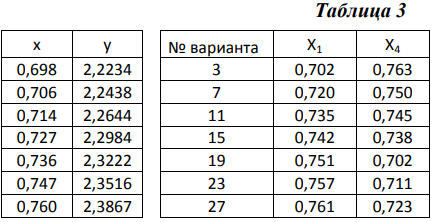
**Исходные данные:**

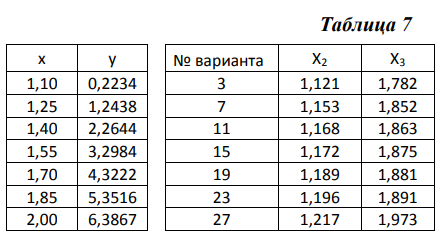
Для исследования использовать:

• линейную и квадратичную интерполяцию;

• многочлен Лагранжа;

• многочлен Ньютона.





**Методика проведения исследования:**

* С помощью линейной и квадратичной интерполяции найти приближенное значение функции при х= Х1 (см. табл. 1).
* Найти приближенное значение функции при х= Х1 (см. табл. 1) с помощью многочлена Лагранжа.
* Используя первую или вторую интерполяционную формулу Ньютона, вычислить значения функции при данных значениях аргумента (для значения Х2 и Х3, см. табл. 5).
* Вычислить значения функции, используя интерполяционную формулу Ньютона для не равноотстоящих узлов (для х=Х4, см. табл. 1). При вычислениях учитывать только разделенные разности первого и второго порядков. Вычисления произвести дважды, используя различные узлы.

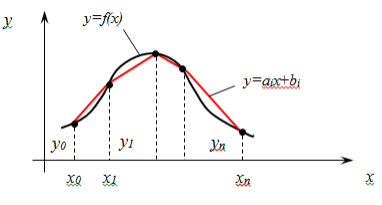
**Программная реализация задачи:**

* Предусмотреть ввод исходных данных (исходные таблицы) из файла.
* Предусмотреть ввод значения аргумента, для которого вычисляется приближенное значение функции, с клавиатуры.
* Реализовать численные методы интерполирования, каждый метод в отдельной функции/классе.
* Предусмотреть вывод результатов на экран.

**Ход работы:**

**Линейная интерполяция:**

*Идея метода:* простейшим и часто используемым видом локальной интерполяции является линейная интерполяция. Она состоит в том, что заданные точки , соединяются прямолинейными отрезками, и функция приближается к ломаной с вершинами в данных точках.



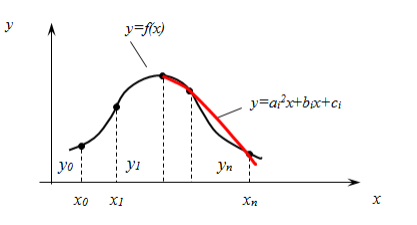
Уравнения каждого отрезка ломаной линии в общем случае разные. Поскольку имеется n интервалов , то для каждого из них в качестве уравнения интерполяционного полинома используется уравнение прямой, проходящей через две точки. В частности, для — го интервала можно написать уравнение прямой, проходящей через точки и в виде:

Отсюда:

Следовательно, при использовании линейной интерполяции сначала нужно определить интервал, в который попадает значение аргумента x, а затем подставить его в формулу (2) и найти приближенное значение функций в этой точке.

**Квадратичная интерполяция:**

*Идея метода:* в случае квадратичной интерполяции в качестве интерполяционной функции на отрезке принимается квадратный трехчлен. Уравнения квадратного трехчлена имеет вид:



Для определения неизвестных коэффициента необходимы три уравнения. Ими служат условия прохождения параболы (3) через три точкиЭти условия можно записать в виде:

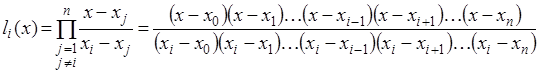
Интерполяция для любой точки проводится по трем ближайшим ней узлам.

**Многочлен Лагранжа:**

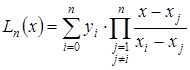
*Идея метода:* при глобальной интерполяции на всем интервале  строится единый многочлен. Одной из форм записи интерполяционного многочлена для глобальной интерполяции является многочлен Лагранжа:



где  – базисные многочлены степени *n*:



То есть многочлен Лагранжа можно записать в виде:



Многочлен удовлетворяет условию

Это условие означает, что многочлен равен нулю при каждом  кроме , то есть

 – корни этого многочлена.

Таким образом, степень многочлена  равна *n* и при  обращаются в ноль все слагаемые суммы, кроме слагаемого с номером , равного .

Выражение применимо как для равноотстоящих, так и для не равноотстоящих узлов. Погрешность интерполяции методом Лагранжа зависит от свойств функции *,* от расположения узлов интерполяции и точки *x*. Полином Лагранжа имеет малую погрешность при небольших значениях *n* (*n*<20). При больших *n* погрешность начинает расти, что свидетельствует о том, что метод Лагранжа не сходится (то есть его погрешность не убывает с ростом *n*).

**Многочлен Ньютона:**

*Идея метода:* другая форма записи интерполяционного многочлена – интерполяционный многочлен Ньютона с разделенными разностями. Пусть функция  задана с произвольным шагом, и точки таблицы значений пронумерованы в произвольном порядке.

**Разделенные разности** нулевого порядка совпадают со значениями функции в узлах. Разделенные разности первого порядка определяются через разделенные разности нулевого порядка:



Разделенные разности второго порядка определяются через разделенные разности первого порядка:



Разделенные разности *k*-го порядка определяются через разделенные разности порядка :



Используя понятие разделенной разности интерполяционный многочлен Ньютона можно записать в следующем виде:



**Интерполяционный полином Ньютона для равноотстоящих узлов:**

Узлы интерполирования *x*0, *x*1, ..., *xn* называются равноотстоящими, если:

 , где*h* - шаг интерполирования, *.*

Конечные разности являются рабочим аппаратом при изучении функций, заданных таблицей значений в равноотстоящих узлах.

**Конечными разностями первого порядка** называют величины:

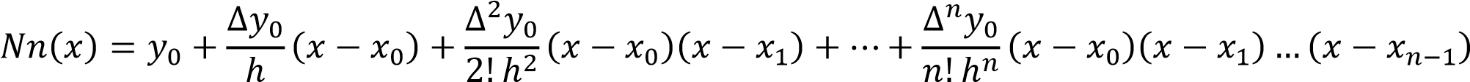
, 

**Конечными разностями второго порядка** называют величины:

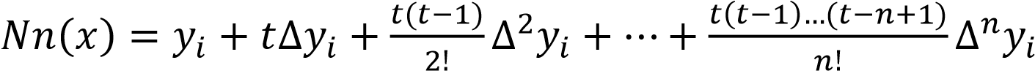


**Конечными разностями k-го порядка** называют величины:





Введем обозначение: ( . Тогда получим формулу Ньютона, которая называется *первой интерполяционной формулой Ньютона для интерполирования вперед*:



Полученное выражение может аппроксимировать функцию на левом половине отрезка.

Для правой половины отрезка разности вычисляют справа налево: (. Тогда получим формулу Ньютона, которая называется **второй интерполяционной формулой Ньютона для интерполирования назад:**



При **экстраполировании** для отыскания значений функции для используется первый интерполяционный многочлен Ньютона. В этом случае *t =< 0* и говорят, что первая интерполяционная формула Ньютона применяется для **экстраполирования назад**.

При отыскании значений функции для используется второй интерполяционный многочлен Ньютона.

В этом случае *t => 0* и говорят, что вторая интерполяционная формула Ньютона применяется для **экстраполирования вперед**.

**Вычисления:**

**1.** Найти приближенное значение функции y=f(x) при х=0,751 для таблицы 3.

**Решение:** используем линейную интерполяцию. Значение x=0,751 находится между узлами . Тогда:

**Решение:** используем квадратичную интерполяцию. Составим систему уравнений для ближайших узлов к точке x=0,751: , соответственно .

В результате решения системы получим:

**2.** Найти приближенное значение функции y=f(x) при х=0,751 для таблицы 3 с помощью многочлена Лагранжа.

**Решение:**

**3.** Используя первую или вторую интерполяционную формулу Ньютона найти приближенное значение функции для , для таблицы 7.

**Решение:** для вычисления значение функции при воспользуемся формулой Ньютона для интерполирования вперед, т.к. лежит в левой половине отрезка.

Поскольку , то при использовании первой интерполяционной формулой Ньютона для интерполирования вперед необходимо вычислить:

Примем . Тогда

Конечные разности первого порядка:

Тогда:

Конечные разности второго порядка:

Тогда:

Конечные разности третьего порядка:

Тогда :

Конечные разности четвертого порядка:

Тогда:

Конечные разности пятого порядка:

Тогда:

Конечные разности шестого порядка:

Тогда:

Получаем:

**Решение:** для вычисления значение функции при воспользуемся формулой Ньютона для интерполирования назад, т.к. лежит в правой половине отрезка.

Тогда

Конечные разности уже посчитаны.

Получаем:

**4.** Используя интерполяционную формулу Ньютона для неравноотстоящих узлов найти приближенное значение функции для х=0,702 для таблицы 3. При вычислениях учитывать только разделенные разности первого и второго порядков. Вычисления провести дважды, используя различные узлы.

**Решение:** вычисления произведем по формуле:

Для вычисления значения функции при за возьмем сначала 0.698, затем 0.706

Для

Принимаем

**Листинг:**

Main.java

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Scanner;

public class Main {

static double[] x = new double[7];

static double[] y = new double[7];

public static void main(String[] args) throws IOException {

String answer;

double x\_curr;

Scanner scanner = new Scanner(System.in);

readFile("data\_1.txt");

System.out.println("Введите значение аргумента для м.Лагранжа: ");

x\_curr = scanner.nextDouble();

answer = Lagrange.solve(x, y, x\_curr);

System.out.println("Ответ: " + answer);

readFile("data\_2.txt");

System.out.println("Введите значение аргумента для м.Ньютона для равностоящих узлов: ");

x\_curr = scanner.nextDouble();

NewtonEqualDiff newtonEqualDiff = new NewtonEqualDiff(x, y, x\_curr);

answer = newtonEqualDiff.solve();

System.out.println("Ответ: " +answer);

readFile("data\_1.txt");

System.out.println("Введите значение аргумента для м.Ньютона для НЕравностоящих узлов: ");

x\_curr = scanner.nextDouble();

NewtonNotEqualDiff newtonNotEqualDiff = new NewtonNotEqualDiff(x, y, x\_curr);

answer = newtonNotEqualDiff.solve();

System.out.println("Ответ: " +answer);

}

public static void readFile(String filename) throws IOException {

String row;

String[] data;

BufferedReader reader = new BufferedReader(new FileReader(filename));

int i = 0;

while ((row = reader.readLine()) != null) {

data = row.split(" ");

x[i] = Double.parseDouble(data[0]);

y[i] = Double.parseDouble(data[1]);

i++;

}

}

}

NewtonNotEqualDiff.java

public class NewtonNotEqualDiff {

private double x\_curr;

private double[] x;

private double[] y;

private double[] node\_x;

private double[] node\_y;

private double res;

private int size;

public NewtonNotEqualDiff(double[] x, double[] y, double x\_curr) {

this.x = x;

this.y = y;

this.size = x.length;

this.x\_curr = x\_curr;

}

public String solve() {

for (int i = 0; i < size; i++) {

if (x\_curr < x[size - 1] && x\_curr > x[size - 1 - 1] || x\_curr > x[i] && (size - 1 - i) < 3) {

for (int j = 1; j >= 0; j--) {

node\_x = new double[]{x[size - 1 - 2 - j], x[size - 1 - 1 - j], x[size - 1 - j]};

node\_y = new double[]{y[size - 1 - 2 - j], y[size - 1 - 1 - j], y[size - 1 - j]};

res += calculate();

}

break;

} else if (x\_curr > x[0] && x\_curr < x[1] || x\_curr > x[i] && (size - 1 - i) >= 3) {

for (int j = 0; j <= 1; j++) {

node\_x = new double[]{x[i + j], x[i + j + 1], x[i + j + 2]};

node\_y = new double[]{y[i + j], y[i + j + 1], y[i + j + 2]};

res += calculate();

}

break;

}

}

return String.valueOf(res / 2);

}

private double calculate() {

return node\_y[0] + f2(0, 1) \* (x\_curr - node\_x[0]) + f3(0, 1, 2) \* (x\_curr - node\_x[0]) \* (x\_curr - node\_x[1]);

}

private double f2(int a, int b) {

return (node\_y[b] - node\_y[a]) / (node\_x[b] - node\_x[a]);

}

private double f3(int a, int b, int c) {

return (f2(b, c) - f2(a, b)) / (node\_x[c] - node\_x[a]);

}

}

NewtonEqualDiff.java

public class NewtonEqualDiff {

private double[] x;

private double[] y;

private double x\_curr;

private int n;

static int begin;

public NewtonEqualDiff(double[] x, double[] y, double x\_curr) { // равностоящие узлы

this.x = x;

this.y = y;

this.n = x.length;

this.x\_curr = x\_curr;

}

public String solve() {

double res;

double t;

if (x\_curr < x[0]) {

t = (x\_curr - x[0]) / (x[1] - x[0]);

res = backward(t);

} else if (x\_curr > x[n - 1]) {

t = (x\_curr - x[n - 1]) / (x[1] - x[0]);

res = forward(t);

} else if (x\_curr <= x[n / 2]) {

begin = 0;

while (x\_curr > x[begin])

begin++;

begin--;

t = (x\_curr - x[begin]) / (x[1] - x[0]);

res = forward(t);

} else {

begin = 0;

while (x\_curr > x[begin])

begin++;

begin--;

t = (x\_curr - x[begin]) / (x[1] - x[0]);

res = backward(t);

}

return String.valueOf(res);

}

private double forward(double t) {

double res = y[begin] + t \* delta(1);

double ti = t;

int iterator = 0;

for (int i = begin + 1; i < n - 1; i++) {

iterator++;

ti \*= (t - iterator);

res += ti / fact(iterator + 1) \* delta(iterator + 1);

}

return res;

private double backward(double t) {

double res = y[begin] + t \* delta(1, begin - 1);

double ti = t;

for (int i = 1; i < begin; i++) {

ti \*= (t + i);

res += ti / fact(i + 1) \* delta(i + 1, begin - i - 1);

}

return res;

}

private double delta(int q) {

return calcCurrentDelta(q)[begin];

}

private double delta(int q, int p) {

return calcCurrentDelta(q)[p];

}

private double[] calcCurrentDelta(int q) {

double[] res = new double[n];

if (q == 1) {

res = new double[n];

for (int i = 0; i < n - 1; i++) {

res[i] = y[1 + i] - y[i];

}

} else {

for (int i = 0; i < n - 1; i++) {

double a = calcCurrentDelta(q - 1)[i + 1];

double b = calcCurrentDelta(q - 1)[i];

res[i] = a - b;

}

}

return res;

}

int fact(int n) {

if (n <= 1)

return 1;

else

return n \* fact(n - 1);

}

}

**Примеры работы программы:**

Введите значение аргумента для м.Лагранжа:

0,51

Ответ: 2.362389151302407

Введите значение аргумента для м.Ньютона для равностоящих узлов:

1,189

Ответ: 0.8348156988823346

Введите значение аргумента для м.Ньютона для НЕравностоящих узлов:

0,751

Ответ: 2.362367424242424

Введите значение аргумента для м.Лагранжа:

0,705

Ответ: 2.24124354387142

Введите значение аргумента для м.Ньютона для равностоящих узлов:

1,26

Ответ: 1.3097333509179696

Введите значение аргумента для м.Ньютона для НЕравностоящих узлов:

0,705

Ответ: 2.2412406850961535

Ссылка на github - [https://github.com/numvc/Comp\_Mathematics\_lab](https://github.com/numvc/Comp_Mathematics_lab3)4

**Вывод:** в результате выполнения лабораторной работы я научился искать приближение для какой-либо функции методом наименьших квадратов. За исходные данный может браться как сама функция, так и точки, соответствующие её координатам на плоскости. Также вспомнил статистические характеристики(отклонения), исходя из которых выбирается наилучшая аппроксимирующая функция.