БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа № 5

Интерполяционный многочлен Лагранжа и Ньютона

Кратное интерполирование. Интерполирование при равноотстоящих узлах

Вариант 7

**Выполнил**

Дунаев Виктор

2 курс 5 группа

**Преподаватель**

Радкевич Е.В.

2017

**1.Постановка задачи**

Дана функция ,то

Даны значения в функции в узлах интерполирования.

1. Построить многочлен Лагранжа и Ньютона по значениям функции в заданных узлах интерполирования.
2. Вычислить значение  Лагранжа и Ньютона в точках ,,,.
3. Провести анализ результатов.

**2.1. Теория : многочлен Лагранжа**

Рассмотрим задачу интерполирования  по значениям в ,  с помощью многочлена

.

Коэффициенты  определим из условия , то есть

, .

Определитель этой системы

.

Если все  различны, то , значит, задача алгебраического интерполирования имеет единственное решение.

Будем строить  в виде линейной комбинации  с коэффициентами , являющимися многочленами.

,  - многочлены степени .

При этом должно выполняться

, .

Для этого нужно потребовать

, , .

Таким образом, все ,  являются корнями многочлена -ной степени . Тогда

.

 выберем из условия

. .

.

 - интерполяционный многочлен Лагранжа.

 - погрешность многочлена Лагранжа.

Данная оценка имеет место, если   раз непрерывно дифференцируема на .

, .

**2.2. Теория: многочлен Ньютона**

Рассмотрим многочлен Лагранжа , построенный по узлам , , .





.

Таким образом,

. (1)

Обозначим  многочлен Лагранжа степени , построенный по точкам .

. (2)

, , , .

Рассмотрим .

.

.

.

(1)    .

.

(2)  . (3)

(3) – интерполяционный многочлен Ньютона.

Так как интерполяционный многочлен единственный, то справедлива оценка, полученная для многочлена Лагранжа.

.

1. : .

.

**3. Листинг программы**

class LagrangePolynom {

private double[] x;

private double[] y;

LagrangePolynom(double[] x, double[] y) {

this.x = x;

this.y = y;

}

public double[] theoretical(double[] p){

double[] result = new double[p.length];

for(int j = 0;j<p.length;j++){

double w = p[j] - x[0];

for (int i = 1; i <= x.length-1; ++i)

{

w \*= (p[j] - x[i]);

w /= i;

}

w = Math.abs(w);

result[j] = -1088664.07 \* w;

}

return result;

}

private double calculate(double value) {

double result = 0.0;

for (int i = 0; i < y.length; i++) {

double P = 1.0;

for (int j = 0; j < x.length; j++) {

if (j != i) {

P \*= (value - x[j]) / (x[i] - x[j]);

}

}

result += P \* y[i];

}

return result;

}

public double[] findSolution(double[] points) {

double[] result = new double[points.length];

for (int i = 0; i < points.length; i++) {

result[i] = calculate(points[i]);

}

return result;

}

}

class NewtonPolynom {

private double[] x;

private double[] y;

private double[][] rr;

private int size;

NewtonPolynom(double[] x, double[] y) {

this.size = x.length;

this.x = x;

this.y = y;

rr = new double[size][size];

}

public void createMatrix() {

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

rr[i][0] = y[i];

for (int j = 1; j < size; ++j)

for (int i = 0; i < size - j; ++i)

rr[i][j] = (rr[i + 1][j - 1] - rr[i][j - 1]) / (x[i + j] - x[i]);

}

private double calculate(double value) {

double res = 0, add;

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

add = rr[0][i];

for (int j = 0; j < i; ++j)

add \*= (value - x[j]);

res += add;

}

return res;

}

public double[] findSolution(double[] points) {

createMatrix();

double[] result = new double[points.length];

for (int i = 0; i < points.length; i++) {

result[i] = calculate(points[i]);

}

return result;

}

public double[][] getRr() {

return rr;

}

}

**4.Выходные данные**

**Лагранж**

Значение многочлена Лагранжа:

x\*: 1.435627384542298 r\*: -2.6427486865543415E-13  -1.2374103311078314E-6

x\*\*: 1.3928114421844482 r\*\*: 1.5164092204145163E-13  -1.438988114376071E-8

x\*\*\*: 1.0867447155673067 656398 r\*\*\*: 1.5284287446792888E-13 -1.2374103311078314E-6

Теоретическая погрешность в среднем на порядок отличается от истинной в силу того, что при её вычислении брали не точное значение .

**Ньютон**

1.4305 0.177999999999998 -0.4649 -0.283 0.250 -0.500 1.388 -2.7777777777901322 3.720238095268618 -2.20458553 -4.6847442

1.4483 0.085 -0.55 -0.183333333 -3.3827107781547743E-13 0.333 -0.555 0.1984126984247622 1.73611 -6.88932

1.4568 -0.025 -0.6049999999 -0.1833 0.16666 -1.1672884880908896E-12 -0.416666 1.587301587 -4.4642857

1.4543 -0.1459999 -0.66 -0.1166 0.1666 -0.24999 0.6944444 -1.98412698411

1.4397 -0.278 -0.6949 -0.0500 0.04166 0.1666 -0.6944444444396621

1.4119 -0.4169 -0.7100 -0.0333 0.1249 -0.24999999999871805

1.3702 -0.5590 -0.7199 0.0166 2.91407523111964E-13

1.3143 -0.7029 -0.7150 0.0166

1.244 -0.8460 -0.7099

1.1594 -0.9879

1.0606

Значение многочлена Ньютона:

x\*: 1.4356273845422995 r\*: -7.633937926243561E-13 -1.2374103311078314E-6

x\*\*: 1.3928114421844482 r\*\*: 2.9984014443252818E-13 -1.438988114376071E-8

x\*\*\*: 1.0867447155673042 r\*\*\*: 5.551115123125783E-13 -1.2374103311078314E-6

Реальная погрешность многочлена Ньютона больше, чем многочлена Лагранжа, однако теоретическая оценка даёт лучшие результаты, чем аналогичная построенная для метода Лагранжа. Это происходит из-за того, что верхняя граница оценки вычисляется более точно, так как нет необходимости дополнительно оценивать . Многочлен Лагранжа в явном виде содержит значения функций в узлах интерполяции, поэтому он удобен, когда значения функций меняются, а узлы интерполяции неизменны. Число арифметических операции, необходимых для построения многочлена Лагранжа, пропорционально http://aco.ifmo.ru/el_books/numerical_methods/lectures/images/image153.png и является наименьшим для всех форм записи. К недостаткам этой формы записи можно отнести то, что с изменением числа узлов приходится все вычисления проводить заново.

**5.Постановка задачи**

Дана функция ,то

Даны значения в функции в узлах интерполирования.

1. Построить многочлен Ньютона для интерполирования в начале таблицы и в конце таблицы по значениям функции в заданных равноотстоящих узлах.

2.Вычислить значение  Лагранжа и Ньютона в точках ,,,.

3. Провести анализ результатов.

**6.Теория: интерполирование при равноотстоящих узлах**



Пусть  близка к . Положим . Узлы интерполирования расположим в следующем порядке: .

.



Многочлен примет вид

 (1)

(1) – многочлен Ньютона для интерполирования в начале таблицы.

.

Пусть  близка к . Положим . Узлы интерполирования расположим в следующем порядке: .

.



 (2)

(2) – многочлен Ньютона для интерполирования в конце таблицы.

.

**7.Листинг программы**

class Equidistant {

private double[] x;

private double[] y;

private double[][] kr;

private double h;

private int size;

public Equidistant(double[] x, double[] y) {

this.size = x.length;

this.h = x[1] - x[0];

this.x = x;

this.y = y;

this.kr = new double[size][size];

this.createMatrix();

}

private void createMatrix() {

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

kr[i][0] = y[i];

for (int j = 1; j < size; ++j)

for (int i = 0; i < size - j; ++i)

kr[i][j] = (kr[i + 1][j - 1] - kr[i][j - 1]);

}

private double newtonPolynom (double value)

{

double t = (value - x[0]) / h;

double res = kr[0][0];

double mul = t;

for (int i = 1; i < size; ++i)

{

res += (kr[0][i] \* mul);

mul \*= ((t - i) / (i + 1));

}

return res;

}

public double[] findSolution(double[] points) {

double[] result = new double[points.length];

for (int i = 0; i < points.length; i++) {

result[i] = newtonPolynom(points[i]);

}

return result;

}

public double[][] getKr(){

return kr;

}

}

**8.Выходные данные**

Таблица конечных разностей:

1.4305 0.01779 -0.0092 -0.0017 6.000000012662E-4 -6.000000000021544E-4 0.0010 -0.0014 0.00150000 -8.000000000225604E-4 -0.00169999

1.4483 0.0085 -0.011 -0.00109 -8.88178419752E-16 4.0000051026E-4 -4.000000000028425E-4 1.000000000055401E-4 6.9999868E-4 -0.00249

1.4568 -0.00250 -0.01209 -0.0011 4.000000000006221E-4 -1.3322676201878E-15 -2.999999999973024E-4 7.999999999950269E-4 -0.0017999

1.4543 -0.014599 -0.0132 -6.999999999997009E-4 3.999999999992898E-4 -2.999986347E-4 4.99999977245E-4 -9.9999965592E-4

1.4397 -0.02780 -0.013899 -3.0000000000041105E-4 1.0000000000065512E-4 1.9999999990898E-4 -4.999999988347E-4

1.4119 -0.04169999 -0.0142 -1.9999999999975593E-4 2.999999999997449E-4 -2.99999997449E-4

1.3702 -0.0559 -0.014399999999999968 9.999999999998899E-5

1.3143 -0.0703 -0.0142 9.999999999998899E-5

1.244 -0.0846000 -0.0141999999

1.1594 -0.0988

1.0606

P(x\*): 1.4356273845423 r\*: -2.642747793759218E-14 -1.2374103311078314E-6

P(x\*\*):1.3928114421844486 r\*\*:1.516398188427E-14-1.438988114376071E-8

P(x\*\*\*): 1.0867447155673056 r\*\*\*:1.528428078545474E-14-1.2374103311078314E-6

Теоретическую погрешность вычислял по аналогии Ньютона.

Интерполяция с применением данных формул даёт более точный результат, чем с использованием многочлена Ньютона без учёта расположения узлов и точек, в которых необходимо узнать значение функции.

**9.Постановка задачи**

Дана функция ,то

Даны значения в функции в узлах интерполирования.



1. Построить многочлен Эрмита пятой степени по значениям функции и её производной в заданных узлах интерполирования.
2. Вычислить значение  в точках , , .
3. Провести анализ результатов.

**10.Теория: кратное интерполирование**

Рассмотрим функцию  на . Пусть , , , . Предположим, что известны , , ,  - кратность узла . Пусть .

.

, , .

Последнее – система линейных уравнений для определения коэффициентов . Она имеет единственное решение, так как определитель этой системы отличен от нуля.

Пусть известна .

.

Известно, что , . Тогда

.

Рассмотрим многочлен Ньютона

.

Если учесть кратность узлов, т.е. , , , , то получим многочлен Эрмита.

**11.Листинг программы**

class Hermit {

private double[] x;

private double[] y;

private double[] ders;

private double[][] rr;

private int size;

public Hermit(double[] x, double[] y, double[] ders) {

this.size = x.length;

this.x = x;

this.y = y;

this.ders = ders;

this.rr = new double[size][size];

this.createMatrix();

}

private void createMatrix() {

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

rr[i][0] = y[i];

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

rr[i][1] = (x[i] == x[i + 1] ? ders[i] : (rr[i + 1][0] - rr[i][0]) / (x[i + 1] - x[i]));

for (int j = 2; j < size; ++j)

for (int i = 0; i < size - j; ++i)

rr[i][j] = (rr[i + 1][j - 1] - rr[i][j - 1]) / (x[i + j] - x[i]);

}

private double hermitPolynom(double value) {

double res = 0, add;

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

add = rr[0][i];

for (int j = 0; j < i; ++j)

add \*= (value - x[j]);

res += add;

}

return res;

}

public double[] findSolution(double[] points) {

double[] result = new double[points.length];

for (int i = 0; i < points.length; i++) {

result[i] = hermitPolynom(points[i]);

}

return result;

}

public double[][] getRr() {

return rr;

}

}

**12.Выходные данные**

Таблица разделённых разностей.

1.4305 0.2185 -0.5114000000000007 -0.21399999999999708 0.11999999999999655 -0.02039999999999642

1.4305 -0.037200000000000344 -0.6183999999999993 -0.09400000000000053 0.09960000000000013

1.4119 -0.3464 -0.7123999999999998 0.005599999999999605

1.4119 -0.7025999999999999 -0.7096

1.0606 -1.0574

1.0606

P(x\*): 1.435726132925781 r\*: 3.624387916E-8 -1.2374103311078314E-6

P(x\*\*): 1.3928374423749998 r\*\*: 9.2343981293E-8 -1.438988114376071E-8

P(x\*\*\*): 1.0866044028554687 r\*\*\*: 1.123863389217E-9 -1.2374103311078314E-6

Теоретическую погрешность вычислял по аналогии Ньютона.

Для данных точек погрешность кратного интерполирования при помощи первой производной ниже (должна быть, но так как у нас всего 6 узлов, погрешность будет больше) , чем интерполирования многочленом Ньютона, так как они находятся в достаточно небольших окрестностях узлов интерполирования, а значение первой производной в этих узлах даёт информацию о поведении функции в этих окрестностях, в то время как при интерполировании многочленом Ньютона известна информация о поведении функции только в самих узлах.