**ЛАБОРАТОРНАЯ РАБОТА № 6-7**

**ИНТЕРПОЛЯЦИЯ И АППРОКСИМАЦИЯ ФУНКЦИЙ**

Цель работы: научиться строить интерполяционные и аппроксимационные многочлены по заданной системе точек с помощью ЭВМ.

Содержание работы:

1. Изучить принципы построения интерполяционной формулы Лагранжа, I и II интерполяционных формул Ньютона и аппроксимационного полинома.

2. На конкретном примере усвоить порядок построения указанных полиномов с помощью ЭВМ.

3. Составить программу на любом языке программирования, реализующую процесс построения указанных полиномов второго порядка для системы из трех равноотстоящих узловых точек.

4. Сделать вывод о точности построения полиномов.

5. Составить отчет о проделанной работе.

Задание. (Вариант 3)

1. Составить таблицу значений экспериментальной функции с точностью для равностоящей системы из трех узловых точек на отрезке из области допустимых значений функции, где
2. По сформированной системе точек построить интерполяционную формулу Лагранжа, I и II интерполяционные формулы Ньютона и аппроксимационный полином второго порядка.
3. Составить программу на любом языке программирования, реализующую процесс построения указанных полиномов для заданной системы точек.

Решение.

1. Таблица значений функции с точностью для равностоящей системы из трех узловых точек на отрезке , где имеет вид:

|  | 0 | 0.5 | 1 |
| --- | --- | --- | --- |
|  | -2.423 | 3.658 | 19.683 |

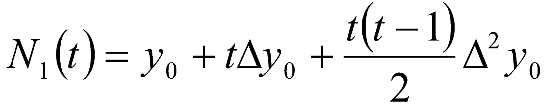
1. Интерполяционный полином Лагранжа. Так как данный полином строится для произвольной системы узловых точек, то запишем этот полином для равноотстоящих узловых точек:

где коэффициенты вычисляются так:

Тогда искомый многочлен Лагранжа второго порядка будут иметь вид:

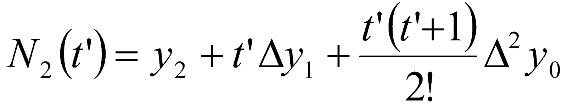
где

I интерполяционная формула Ньютона второго порядка по заданной системе точек строится в виде:

**

Здесь величины называются табличными разностями первого и второго порядка соответственно,

II интерполяционная формула Ньютона второго порядка по заданной системе точек строится в виде:

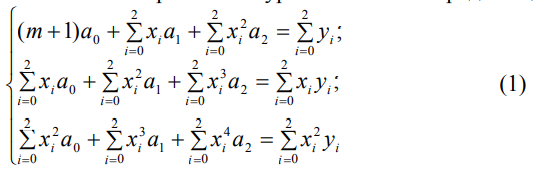
**

Здесь величины и вводятся аналогично случаю, рассмотренному выше,

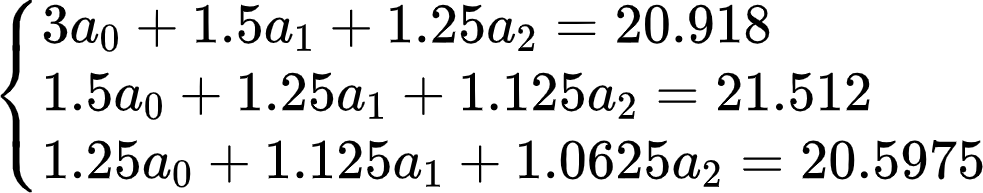
При построении аппроксимационного многочлена методом наименьших квадратов необходимо, чтобы сумма квадратов отклонений построенной функции от экспериментальной в узловых точках была минимальна. Будем строить функцию в виде многочлена второго порядка

Согласно алгоритму метода наименьших квадратов, для построения многочлена второй степени необходимо вычислить следующие суммы:

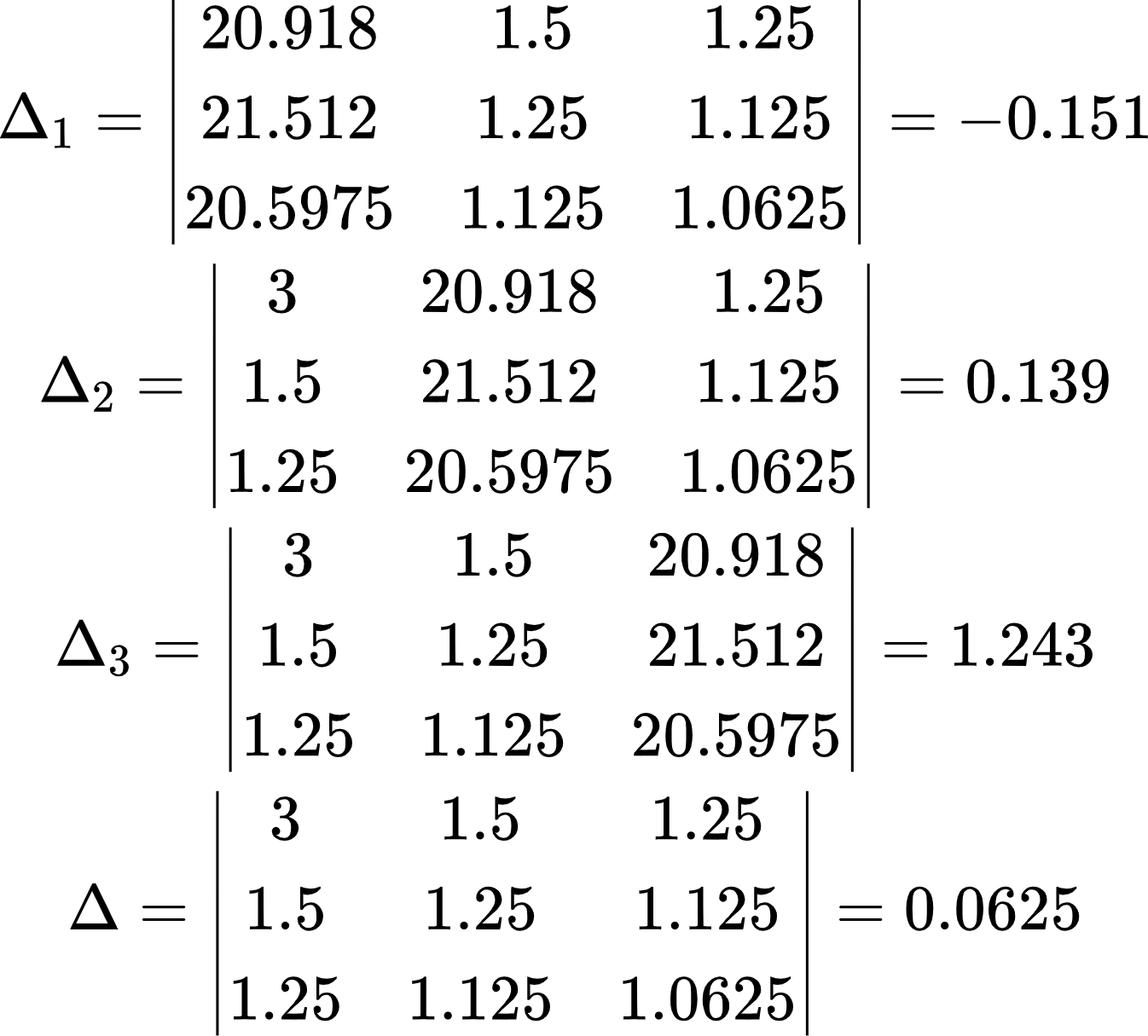
и решить СЛАУ 3-го порядка вида



относительно неизвестных коэффициентов . В данном случае система (1) будет выглядеть так:



Для ее решения можно воспользоваться любым известным методом, например, методом Крамера. Для этого необходимо вычислить четыре определителя системы (2) вида:



Значения искомых коэффициентов вычисляются по формулам:

Искомый многочлен второго порядка будет иметь вид:

Для проверки правильности построения полиномов необходимо провести программно процесс табулирования четырех построенных полиномов и экспериментальной функции при с одинаковым шагом табулирования.

Графики этих функций представлены на рисунке 1. Из графика видно, что искомые полиномы на отрезке практически совпадают с экспериментальной функцией и проходят через узловые точки.

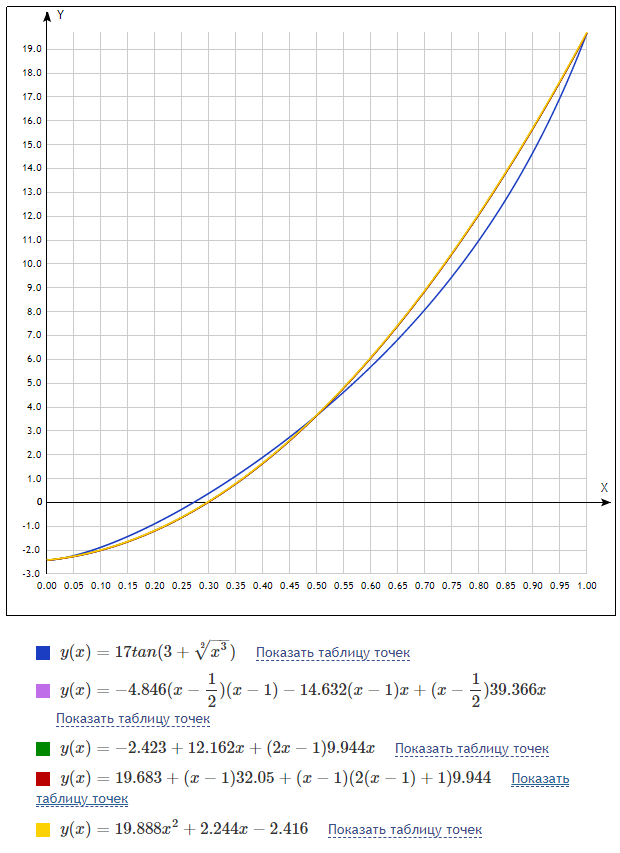


Рис.1 (сверху-вниз: исходная функция, интерполяционный полином Лагранжа, I интерполяционная формула Ньютона, II интерполяционная формула Ньютона, аппроксимационный многочлен)

1. Листинг программы

Program.cs

namespace Program

{

public class Program

{

static float \_x0 = 0f ;

static float \_x2 = 1f ;

static int n = 10 ;

static float \_x1 = (\_x0 + \_x2) / 2 ;

static float \_h = \_x1 - \_x0;

static float step = (\_x2 - \_x0) / n ;

static float \_y0 = CompInitFunction(\_x0);

static float \_y1 = CompInitFunction(\_x1);

static float \_y2 = CompInitFunction(\_x2);

*// Коэффициенты интерполяционного полинома Лагранжа*

static float a0 = \_y0 / ((\_x0 - \_x1) \* (\_x0 - \_x2));

static float a1 = \_y1 / ((\_x1 - \_x0) \* (\_x1 - \_x2));

static float a2 = \_y2 / ((\_x2 - \_x0) \* (\_x2 - \_x1));

*// Табличные разности для интерполяционных формул Ньютона*

static float[] dy = { \_y1 - \_y0, \_y2 - \_y1 };

*// Вычисление коэффициентов аппроксимационного многочлена*

static float[] x\_arr = {\_x0, \_x1, \_x2 };

static float[] y\_arr = {\_y0, \_y1, \_y2 };

static IEnumerable<(float X, float Y)> x\_y\_arrs = x\_arr.Zip(y\_arr);

static float[][] columns =

{

new [] { 3, x\_arr.Sum(), x\_arr.Sum(x => x \* x) },

new [] { x\_arr.Sum(), x\_arr.Sum(x => x \* x), x\_arr.Sum(x => x \* x \* x) },

new [] { x\_arr.Sum(x => x \* x), x\_arr.Sum(x => x \* x \* x), x\_arr.Sum(x => x \* x \* x \* x) }

};

static float[] free\_column = {y\_arr.Sum(), x\_y\_arrs.Sum(z => z.X \* z.Y), x\_y\_arrs.Sum(z => z.X \* z.X \* z.Y)};

private static float det = FindDeterminator(columns);

private static float det\_1 = FindDeterminator(new[] { free\_column, columns[1], columns[2] });

private static float det\_2 = FindDeterminator(new[] { columns[0], free\_column, columns[2] });

private static float det\_3 = FindDeterminator(new[] { columns[0], columns[1], free\_column });

private static float \_a0 = det\_1 / det;

private static float \_a1 = det\_2 / det;

private static float \_a2 = det\_3 / det;

private static float FindDeterminator(float[][] columns)

{

return columns[0][0] \* (columns[1][1] \* columns[2][2] - columns[2][1] \* columns[1][2]) -

columns[0][1] \* (columns[1][0] \* columns[2][2] - columns[2][0] \* columns[1][2]) +

columns[0][2] \* (columns[1][0] \* columns[2][1] - columns[2][0] \* columns[1][1]);

}

private static float FindAbsError(float accurateValue, float approximateValue)

{

return Math.Abs(accurateValue - approximateValue);

}

private static float CompInitFunction(float x)

{

return 17 \* (float)Math.Tan(3 + Math.Sqrt(Math.Pow(x, 3)));

}

private static float CompLagrangePolynomial(float x)

{

return a0 \* (x - \_x1) \* (x - \_x2) + a1 \* (x - \_x0) \* (x - \_x2)

+ a2 \* (x - \_x0) \* (x - \_x1);

}

private static float CompFirstPolynomialNewton(float x)

{

var t = (x - \_x0) / \_h;

return \_y0 + t \* dy[0] + (t \* (t - 1) / 2) \* (dy[1] - dy[0]) ;

}

private static float CompSecondPolynomialNewton(float x)

{

var t = (x - \_x2) / \_h;

return \_y2 + t \* dy[1] + (t \* (t + 1) / 2) \* (dy[1] - dy[0]) ;

}

private static float CompApproximationPolynomial(float x)

{

return \_a2\*x\*x + \_a1\*x + \_a0;

}

private static void Main(string[] args)

{

var separator = string.Join(null, Enumerable.Repeat( '-', 55));

Console.WriteLine("y(x)=17tg(3 + sqrt(x^3))\n");

Console.WriteLine(separator);

Console.WriteLine("|{0, -5}|{1,-15}|{2,-15}|{3, -15}|","Xi",\_x0,\_x1,\_x2);

Console.WriteLine(separator);

Console.WriteLine("|{0, -5}|{1,-15}|{2,-15}|{3, -15}|","Yi",\_y0,\_y1,\_y2);

Console.WriteLine(separator + "\n");

separator = string.Join(null, Enumerable.Repeat("-", 156));

Console.WriteLine("|{0,-10}|{1,-15}|{2,-15}|{3,-15}|{4,-15}|{5,-15}|{6,-15}|{7,-15}|{8,-15}|{9,-15}|",

"Xi","f(Xi)","L2(Xi)","|f(Xi)-L2(Xi)|","N1(Xi)","|f(Xi)-N1(Xi)|","N2(Xi)","|f(Xi)-N2(Xi)|","P2(Xi)","|f(Xi)-P2(Xi)|");

Console.WriteLine(separator);

for (float i = \_x0; i < \_x2 + step / 2; i+= step)

{

float x = (float)Math.Round(i, 3);

var Y = CompInitFunction(x);

var L2 = CompLagrangePolynomial(x);

var N1 = CompFirstPolynomialNewton(x);

var N2 = CompSecondPolynomialNewton(x);

var P2 = CompApproximationPolynomial(x);

Console.WriteLine("|{0,-10}|{1,-15}|{2,-15}|{3,-15}|{4,-15}|{5,-15}|{6,-15}|{7,-15}|{8,-15}|{9,-15}|",

x, Y, L2, FindAbsError(Y, L2), N1, FindAbsError(Y, N1), N2, FindAbsError(Y, N2), P2, FindAbsError(Y, P2));

Console.WriteLine(separator);

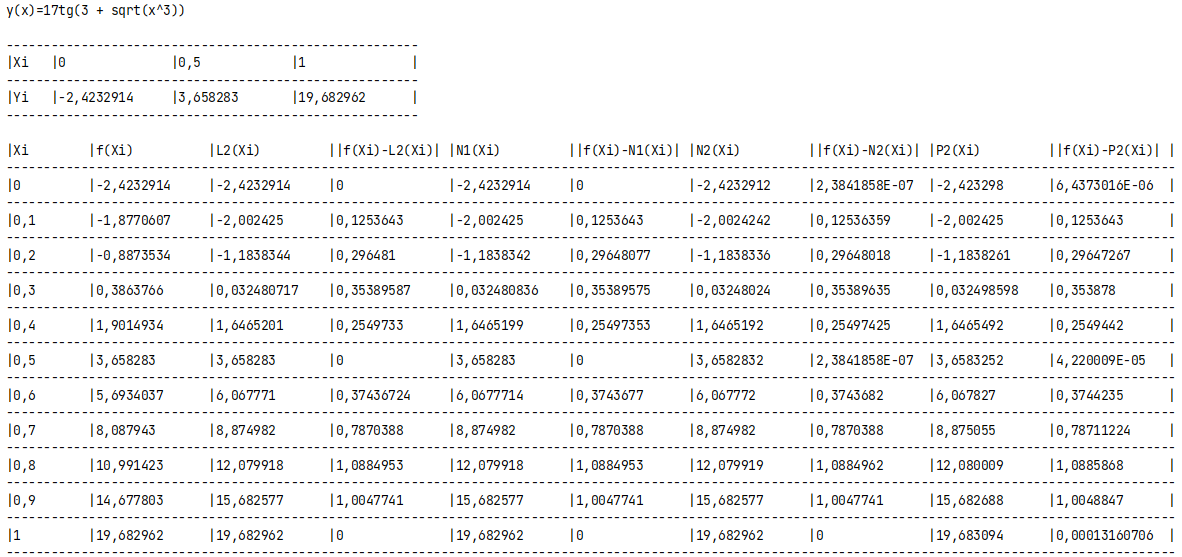
}

}

}

}

4. Работа программы



Вывод: Таким образом, исходя из работы программы мы можем говорить о близости значений построенных полиномов со значениями исходной функции. Значения отклонения минимальны в узловых точках (стремятся к нулю).