Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра «»

Лабораторная работа №5

по дисциплине «Вычислительная математика»

«Интерполяция»

Вариант 15

Выполнил: студент гр. qwinmen.

Проверил:.

Тамбов, 20

**Постановка задачи:**

Составить блок-схему алгоритма и реализовать его в программе для вычислений на ЭВМ для следующих методов интерполяции:

1. Ньютона (1 и 2 формулы);
2. Гаусса (1 и 2 формулы);
3. Стирлинга;
4. Бесселя;
5. Лагранжа;
6. Кубических сплайнов.

При *х0*=1; *х1*=2; *х2*=3; *х3*=4; *х4*=5; y0=6; y1=5; y2=9; y3=2; y4=8;

1. **Первая интерполяционная формула Ньютона:**

(1)

Горизонтальная таблица разностей:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y | Δy | Δ2y | Δ3y | Δ4y |
| 1 | 6 | -1 | 5 | -16 | 40 |
| 2 | 5 | 4 | -11 | 24 |  |
| 3 | 9 | -7 | 13 |  |  |
| 4 | 2 | 6 |  |  |  |
| 5 | 8 |  |  |  |  |

Интерполирующий полином ищется в виде:

(2)

Примем n=4, x0=1, тогда интерполяционная формула Ньютона принимает вид:

(3)

Задача построения многочлена сводится к определению коэффициентов аi из условий:

Полагаем *x* = *x*0, тогда, т.к. второе, третье и другие слагаемые равны 0,

Найдем коэффициент *а*1.

При *x* = *x*1 получим:

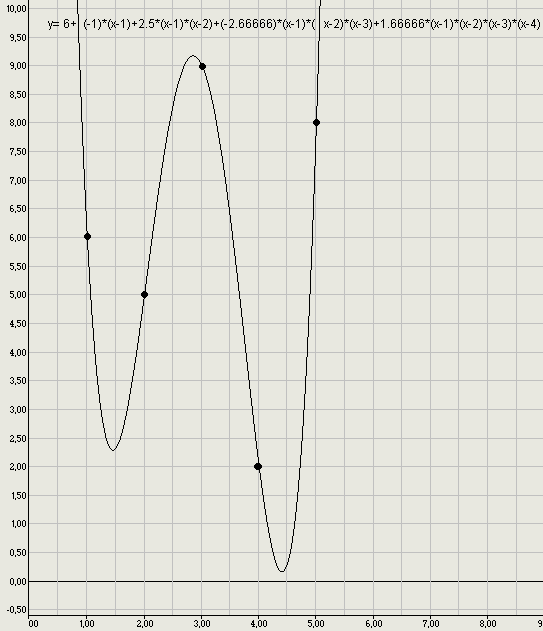
Для определения *а*2, составим конечную разность второго порядка.

При *x* = *x*2 получим:

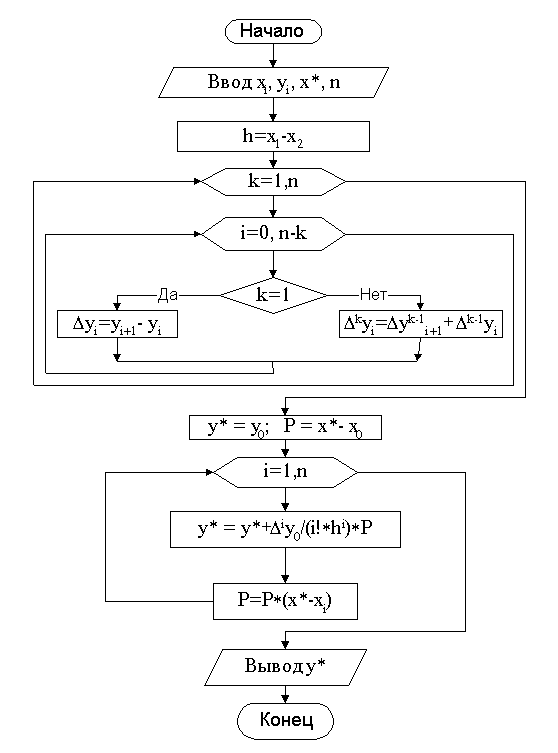
Аналогично можно найти другие коэффициенты. Общая формула имеет вид:

Подставляя эти выражения в формулу (2), получаем:

Этот многочлен называют *интерполяционным полиномом Ньютона* для интерполяции в начале таблицы (интерполирование «вперед») или *первым полиномом Ньютона.*

График функции:

Блок-схема:



Результаты работы программы:

При x= 1.5 y= 2,312

При x= 2.5 y= 8,312

При x= 4.4 y= 0,152

Листинг программы:

static void Main(string[] args)

{

double x = 4.4;//точка для которой строим полином

int[] Xi={1,2,3,4,5};

Console.WriteLine("По методу Ньютона 1: " + N(x, Xi));

}

#region 1Нютон

internal static int F(int x)

{//in X out Y

switch (x)

{

case 1:

return 6;

case 2:

return 5;

case 3:

return 9;

case 4:

return 2;

case 5:

return 8;

}

return 0;

}

static double Delta(int n, int k, int[] Xi) //n-порядок, k-коэффициент

{

if (n == 1)

return (F(Xi[k + 1]) - F(Xi[k]));

else

return (Delta(n - 1, k + 1, Xi) - Delta(n - 1, k, Xi));

}

static double Factorial(int i)

{

if (i <= 1)

return 1;

return i \* Factorial(i - 1);

}

static double N(double x, int[] Xi)

{

double P = 0, h = 1.0, q, c = 1.0;

int i;

q = (x - Xi[0])/h;

P = F(Xi[0]);

for (i = 1; i <= 4; i++)

{

c \*= (q - (i - 1));

P = P + (c\*(1/Factorial(i))\*Delta(i, 0, Xi));

}

return P;

}

#endregion

1. **Вторая интерполяционная формула Ньютона**

Для нахождения значений функций в точках, расположенных в конце интервала интерполирования, используют второй интерполяционный полином Ньютона.

Запишем интерполяционный многочлен в виде

(4)

Коэффициенты *а*0,*а*1,..., *аn* определяем из условия:

Полагаем в (4) n=4, xn=5, тогда

т.е. формула нахождений будет следующей: . (5)

Подставляя эти выражения в формулу (4), получим *вторую интерполяционную формулу Ньютона* или многочлен Ньютона для интерполирования «назад»:

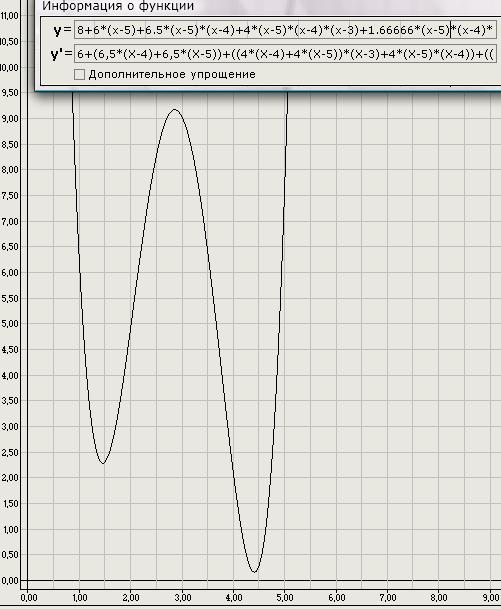
Введем обозначения (в знаменателе 1 – это используемый шаг h):

В общем виде:

Произведя замену в (5) , получим

Это вторая формула Ньютона для интерполирования «назад».

График функции:



Результаты работы программы:

При x= 1.8 y= 3,501

При x= 2.7 y= 8,995

При x= 4.1 y= 1,303

Блок-схема:

начало

a

x0=1; h=1; y[-2]=6; y[-1]=5; y[0]=9; y[1]=2; y[2]=8

a[0]= Δy[0]

Δy[0]=y[4]

3 i=1

1 j=1; l=3

a[i]= Δy[i]/i!

2 i=0

3

i++

Пока i<5

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

k0=a[0]-5a[1]+20a[2]-60a[3]+120a[4]

2

i++

Пока i<4

k1=a[1]-9a[2]+47a[3]-154a[4]

Δy[j]=y[l]

k2=a[2]-12a[3]+71a[4]

1

j++, l--

Пока i<5

k3=a[3]-14a[4]

k4=a[4]

a

b

Ввод *x, n*

конец

Вывод P(x)

b

P(x)=k4x4+k3x3+k2x2+k1x+k0

4 i=0

5

i++

Пока i<n

Листинг программы:

#region 2Ньютон

static void N2(double \_x)

{

double[] X = { 1, 2, 3, 4, 5 };

double[] Y = { 6, 5, 9, 2, 8 };

int n = 4;

double testX = 0;

testX = \_x;

if (testX > X[3] || testX < X[0])

{

Console.Write("Недопустимое значение\n");

}

double[][] razn = new double[n + n + 1][]; /\*создание динамического массива для таблицы конечных разностей

[n+n+1]столько у нас строк в таблице\*/

int k = 0;

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

{

razn[i] = new double[n + 1 + k];

--k;//выделяем память для каждой строки определенное кол столбцов [n+n+1+k]

//к- уменьшает число столбцов в строчке на 1, т.к.

//таблица разностей имеет диагональный вид

}

for (int i = 0; i <= n; ++i)// заполняем нулевой столбец массива значениями функции в узлах

{

razn[i][0] = Y[i];

}

int r = n;

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

{

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

{

int s = -i;

razn[i][j] = razn[-(s - 1)][j - 1] - razn[i][j - 1];//получаем таблицу разностей

}

r--;//это чтобы уменьшать число столбцов, в строке, приводить к диагональному виду таблицу разности

if (r == 0) break;

}

double sum, sum1, sum3, sum2;

double st = X[1] - X[0];

double[] q = new double[10];

double tmp = 1;

sum = Y[n];

int p = 1, u = n;

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

{

q[i] = (testX - X[u]);

tmp \*= q[i];

sum1 = razn[u - 1][i + 1] \* tmp;

sum2 = Math.Pow(st, p) \* factorial(p);//A в степени Б

sum3 = sum1 / sum2;

sum = sum + sum3;

p++; u--;

if (u - 1 < 0) break;//чтобы не выйти за границы таблицы конечн.разностей

}

Console.Write( sum + "\n");

}

static int factorial(int f)

{

int product = 1;

while (f > 0) { product = f \* product; f--; }

return product;

}

#endregion

1. **Первая интерполяционная формула Гаусса:**

Таблица разностей

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y | Δy | Δ2y | Δ3y | Δ4y |
| 1 | 6 |  |  |  |  |
|  |  | -1 |  |  |  |
| 2 | 5 |  | 5 |  |  |
|  |  | 4 |  | -16 |  |
| **3** | **9** |  | **-11** |  | **40** |
|  |  | -7 |  | 24 |  |
| 4 | 2 |  | 13 |  |  |
|  |  | 6 |  |  |  |
| 5 | 8 |  |  |  |  |

Значения разностей теже, что и в методе интерполяции Ньютона.

В общем виде формула имеет вид:

Примем n=4, x0=1, тогда интерполяционная формула Гаусса принимает вид:

Найдем коэффициенты:

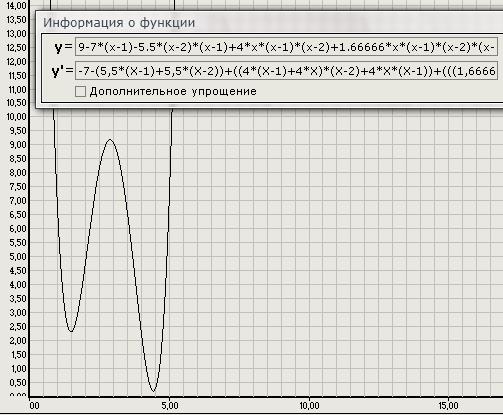
График функции:

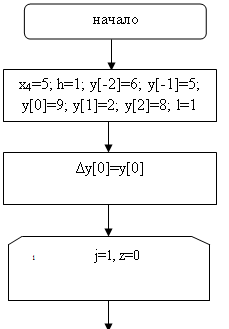
Результаты работы программы:

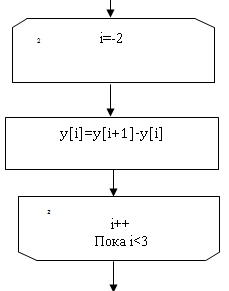
При x= 1.8 y= 3,501

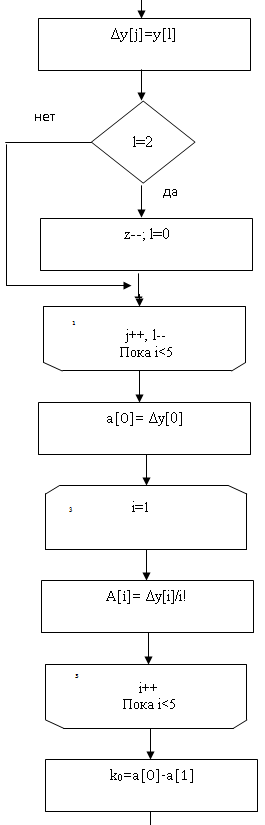
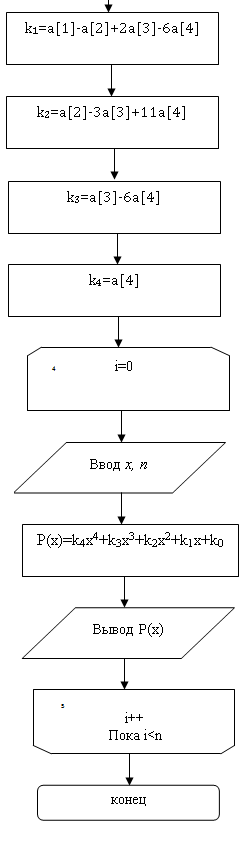
При x= 2.7 y= 8,995

При x= 4.1 y= 1,303

Блок-схема: 







Листинг программы:

static double fact(int n)

{

if (n <= 1)

return 1;

return n\*fact(n - 1);

}

static void main()

{

double[] y;double x = 5,a[],dy[],k4,k3,k2,k1,k0,P,l = 1;

y[-2] = 6;y[-1] = 5;y[0] = 9;y[1] = 2;y[2] = 8;dy[0] = y[0];

for (int j = 1, z = 0; j < 5; j++)

{

for (int i = -2; i < 3; i++)

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

dy[j] = y[z];

l++;

if (l == 2)

{

z--;

l = 0;

}

}

a[0] = dy[0];

for (int i = 1; i < 5; i++)

a[i] = dy[i]/fact(i);

k4 = a[4];

k3 = -6\*a[4] + a[3];

k2 = 11\*a[4] - 3\*a[3] + a[2];

k1 = -6\*a[4] + 2\*a[3] - 3\*a[2] + a[1];

k0 = 2\*a[2] - a[1] + a[0];

for (int x = 1; x < 6; x++)

{

P = k4\*Math.Pow(x, 4) + k3\*Math.Pow(x, 3) + k2\*Math.Pow(x, 2) + k1\*x + k0;

Console.Write("x = {0} P(x) = {1}\n", x, P);

}

}

1. **Вторая интерполяционная формула Гаусса**

Первая интерполяционная формула Гаусса содержит центральные разности

Аналогично можно получить *вторую интерполяционную формулу Гаусса,* содержащую центральные разности

Вторая интерполяционная формула Гаусса имеет вид

Примем n=4, x0=1, тогда интерполяционная формула Гаусса принимает вид:

(6)

Таблица разностей

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y | Δy | Δ2y | Δ3y | Δ4y |
| 1 | 6 |  |  |  |  |
|  |  | -1 |  |  |  |
| 2 | 5 |  | 5 |  |  |
|  |  | 4 |  | -16 |  |
| **3** | **9** |  | **-11** |  | **40** |
|  |  | -7 |  | 24 |  |
| 4 | 2 |  | 13 |  |  |
|  |  | 6 |  |  |  |
| 5 | 8 |  |  |  |  |

При подстановке значений (6) получим:

Результаты работы программы:

При x= 1.5 y= 2,312

При x= 1.05 y= 4,995

При x= 4.54 y= 0,513

Блок-схема:

начало

a

x4=5; h=1; y[-2]=6; y[-1]=5; y[0]=9; y[1]=2; y[2]=8; l=0

1

j++

Пока i<5

Δy[0]=y[0]

3

i++

Пока i<5

3 i=1

a[0]= Δy[0]

A[i]= Δy[i]/i!

k0=a[0]-a[1]

k1=a[1]-a[2]+2a[3]+2a[4]

k2=a[2]+2a[3]-a[4]

k3=a[3]-2a[4]

k4=a[4]

1 j=1, z=-1

2 i=-2

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

2

i++

Пока i<3

Δy[j]=y[z]; l++

нет

l=2

да

z--; l=0

a

b

Ввод *x, n*

конец

Вывод P(x)

b

P(x)=k4x4+k3x3+k2x2+k1x+k0

4 i=0

5

i++

Пока i<n

Листинг программы:

static double fact(int n)

{

if (n <= 1)

return 1;

return n\*fact(n - 1);

}

static void Main()

{

double y[5],x = 5,a[5],dy[5],k4,k3,k2,k1,k0,P,l = 0;

y[-2] = 6;y[-1] = 5;y[0] = 9;y[1] = 2;y[2] = 8;dy[0] = y[0];

for (int j = 1, z = -1; j < 5; j++)

{

for (int i = -2; i < 3; i++) y[i] = y[i + 1] - y[i];

dy[j] = y[z];

l++;

if (l == 2)

{

z--;

l = 0;

}

}

a[0] = dy[0];

for (int i = 1; i < 5; i++) a[i] = dy[i]/fact(i);

k4 = a[4];

k3 = -2\*a[4] + a[3];

k2 = -a[4] - 3\*a[3] + a[2];

k1 = 2\*a[4] + 2\*a[3] - a[2] + a[1];

k0 = -a[1] + a[0];

for (int x = 1; x < 6; x++)

{

P = k4\*pow(x, 4) + k3\*pow(x, 3) + k2\*pow(x, 2) + k1\*x + k0;

Console.Write("x = {0} P(x) = {1}\n", x, P);

}

}

1. **Интерполяционная формула Стирлинга**

Взяв среднее арифметическое первой и второй интерполяционных формул Гаусса, получим *формулу Стирлинга:*

Примем n=4, x0=1, тогда интерполяционная формула Стирлинга принимает вид:

При подстановке и упрощении значений получим:

Результаты работы программы:

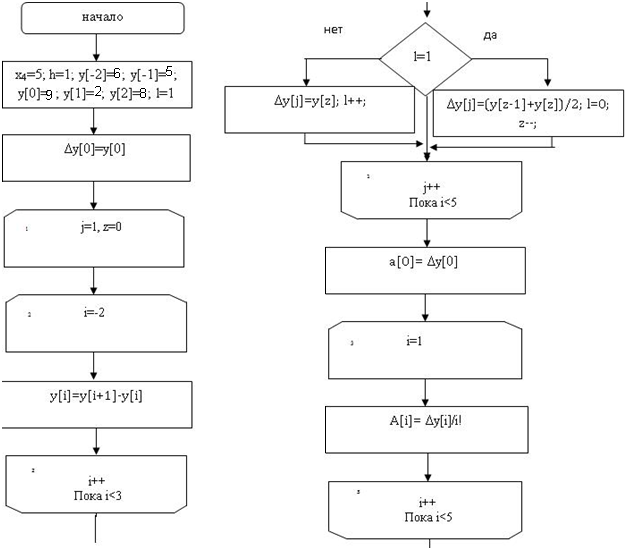
При x= 1.5 y= 2,312

При x= 2.05 y= 5,340

При x= 4.54 y= 0,460

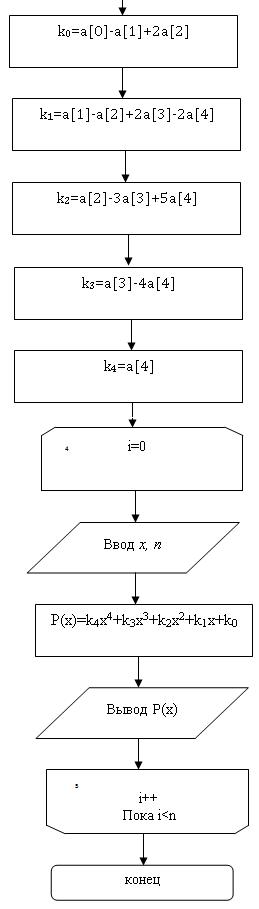
Блок-схема:

А



В

А



Листинг программы:

static double fact(int i)

{

if (i <= 1)

return 1;

return i\*fact(i - 1);

}

static void Main()

{

double y[5],x = 5, a[5], dy[5], k4, k3, k2, k1, k0, P, l = 1;

y[-2] = 6; y[-1] = 5; y[0] = 9; y[1] = 2; y[2] = 8; dy[0] = y[0];

for (int j = 1, z = 0; j < 5; j++)

{

for (int i = -2; i < 3; i++) y[i] = y[i + 1] - y[i];

if (l == 1)

{

dy[j] = (y[z - 1] + y[z])/2;

z--;

l = 0;

}

else

{

dy[j] = y[z];

l++;

}

}

a[0] = dy[0];

for (int i = 1; i < 5; i++) a[i] = dy[i]/fact(i);

k4 = a[4];

k3 = -4\*a[4] + a[3];

k2 = 5\*a[4] - 3\*a[3] + a[2];

k1 = -2\*a[4] + 2\*a[3] - 2\*a[2] + a[1];

k0 = a[2] - a[1] + a[0];

for (int x = 1; x < 6; x++)

{

P = k4\*Math.Pow(x, 4) + k3\*Math.Pow(x, 3) + k2\*Math.Pow(x, 2) + k1\*x + k0;

Console.Write("x = {0} P(x) = {1}\n", x, P);

}

}

1. **Интерполяционная формула Бесселя**

Примем n=4, x0=1, тогда интерполяционная формула Бесселя принимает вид:

При подстановке значений получим:

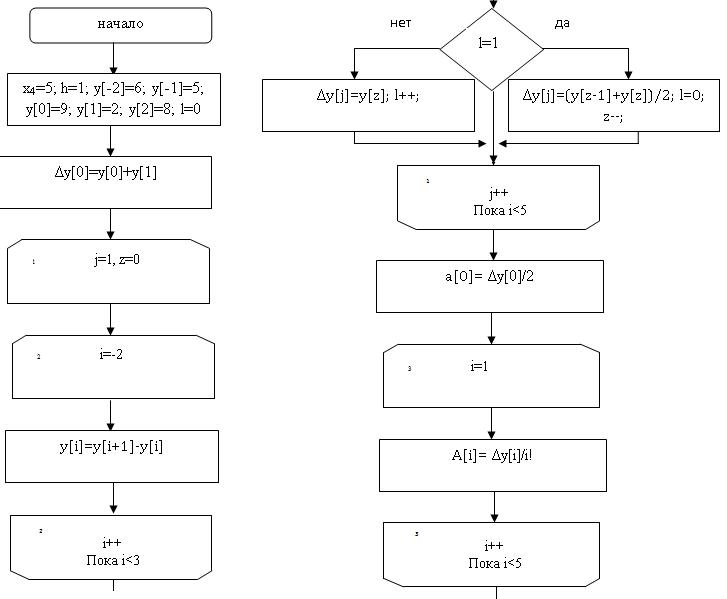
Результаты работы программы:

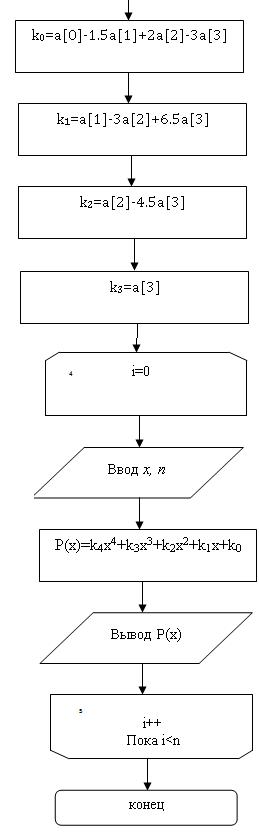
При x= 1.535 y= 2,312

При x= 2.130 y= 5,334

При x= 4.534 y= 0,463

Блок-схема:





Листинг программы:

static double fact(int i)

{

if (i <= 1)

return 1;

return i\*fact(i - 1);

}

static void Main()

{

double y[5], x = 5, a[5], dy[5], k4, k3, k2, k1, k0, P, l = 0;

y[-2] = 6; y[-1] = 5; y[0] = 9; y[1] = 2; y[2] = 8;

dy[0] = y[0] + y[1];

for (int j = 1, z = 0; j < 5; j++)

{

for (int i = -2; i < 3; i++)y[i] = y[i + 1] - y[i];

if (l == 1)

{

dy[j] = (y[z - 1] + y[z])/2;

z--;

l = 0;

}

else

{

dy[j] = y[z];

l++;

}

}

a[0] = dy[0]/2;

for (int i = 1; i < 4; i++)

a[i] = dy[i]/fact(i);

k3 = a[3];

k2 = -4.5\*a[3] + a[2];

k1 = 6.5\*a[3] - 3\*a[2] + a[1];

k0 = -3\*a[3] + 2\*a[2] - 1.5\*a[1] + a[0];

for (int x = 1; x < 6; x++)

{

P = k4\*Math.Pow(x, 4) + k3\*Math.Pow(x, 3) + k2\*Math.Pow(x, 2) + k1\*x + k0;

Console.Write("x = {0} P(x) = {1}\n", x, P);

}

}

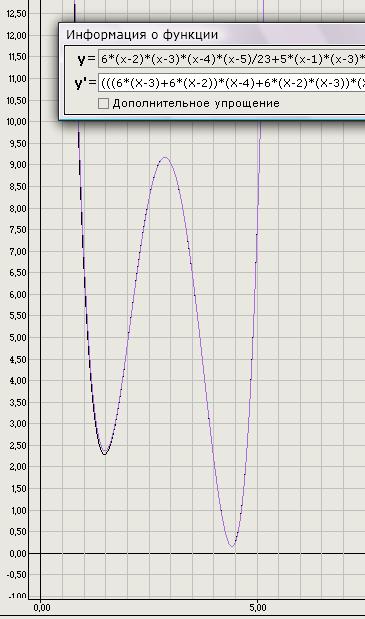
1. **Интерполяционная формулa Лагранжа**

Примем n=4, x0=1, x1=2, x2=3, x3=4, x4=5 тогда интерполяционная формула Лагранжа принимает вид:

При подстановке значений получим:

График функции:

y=6\*(x-2)\*(x-3)\*(x-4)\*(x-5)/23+5\*(x-1)\*(x-3)\*(x-4)\*(x-5)/(-6)+9\*(x-1)\*(x-2)\*(x-4)\*(x-5)/4+2\*(x-1)\*(x-2)\*(x-3)\*(x-5)/(-6)+8\*(x-1)\*(x-2)\*(x-3)\*(x-4)/24



Результаты программы:

При x= 1.535 y= 2,312

При x= 2.1305 y= 5,334

При x= 4.5344 y= 0,463

Блок схема:

y[0]=6; y[1]=5; y[2]=9; y[3]=2; y[4]=8;

начало

k4=y[4]/24+y[3]/(-6)+y[2]/4+y[1]/(-6)+y[0]/24;

k3=(-14)\*y[4]/24+(-13)\*y[3]/(-6)+(-12)\*y[2]/4+(-11)\*y[1]/(-6)+(-10)\*y[0]/24;

k2=71\*y[4]/24+59\*y[3]/(-6)+49\*y[2]/4+41\*y[1]/(-6)+35\*y[0]/24;

k1=(-154)\*y[4]/24+(-107)\*y[3]/(-6)+(-78)\*y[2]/4+(-61)\*y[1]/(-6)+(-50)\*y[0]/24;

k0=120\*y[4]/24+(60)\*y[3]/(-6)+40\*y[2]/4+(30)\*y[1]/(-6)+24\*y[0]/24;

1 i=0

P(x)=k4x4+k3x3+k2x2+k1x+k0

Вывод P(x)

Ввод *x, n*

1

i++

Пока i<n

конец

Листинг программы:

static void Main()

{

double x[5], y[5], k0, k1, k2, k3, k4, P;

y[0] = 6; y[1] = 5; y[2] = 9; y[3] = 2; y[4] = 8;

k4 = y[4]/24 + y[3]/(-6) + y[2]/4 + y[1]/(-6) + y[0]/24;

k3 = (-14)\*y[4]/24 + (-13)\*y[3]/(-6) + (-12)\*y[2]/4 + (-11)\*y[1]/(-6) + (-10)\*y[0]/24;

k2 = 71\*y[4]/24 + 59\*y[3]/(-6) + 49\*y[2]/4 + 41\*y[1]/(-6) + 35\*y[0]/24;

k1 = (-154)\*y[4]/24 + (-107)\*y[3]/(-6) + (-78)\*y[2]/4 + (-61)\*y[1]/(-6) + (-50)\*y[0]/24;

k0 = 120\*y[4]/24 + (60)\*y[3]/(-6) + 40\*y[2]/4 + (30)\*y[1]/(-6) + 24\*y[0]/24;

for (int x = 1; x < 6; x++)

{

P = k4\*Math.Pow(x, 4) + k3\*Math.Pow(x, 3) + k2\*Math.Pow(x, 2) + k1\*x + k0;

Console.Write("x = {0} P(x) = {1}\n", x, P);

}

}

1. **Интерполяционная формула кубических сплайнов.**

На каждом из отрезков [xi-1,xi], i=1,2,…n, будем искать сплайн-функцию s(x)=si(x) в виде полинома третьей степени:

Si(x)=ai+bi(x-xi)+ci/2\*(x-xi)2+d/6\*(x-xi)3, где i=1,2,..n, а xi-1<x<xi, ai, bi, ci, di- искомые коэффициенты.

ai=fi(xi)

Составляем систему уравнений для сi:

hi-1ci-1+2(hi+hi+1)ci+hi+1ci+1=6((yi+1-yi)/hi+1-(yi-yi-1)/hi)

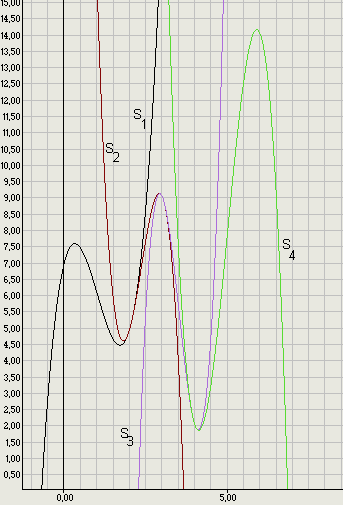
Найдя значение сi, можно посчитать di и bi :

di=(ci-ci-1)/hi; bi=hici/2-hidi/6+(yi-yi-1)/hi

Таким образом, посчитав значение коэффициентов, будет найден единственный кубический сплайн.

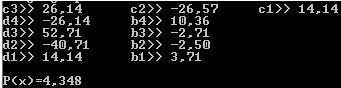
При подстановке значений получил полином:

Графики функций:



Результаты программы:

Для x= 3.63:



Для x= 2.12:



Для x= 5:



Листинг класса программы:

class CubicSpline

{

SplineTuple[] splines; // Сплайн

// Структура, описывающая сплайн на каждом сегменте сетки

private struct SplineTuple

{

public double a, b, c, d, x;

}

// Построение сплайна

// x - узлы сетки, должны быть упорядочены по возрастанию, кратные узлы запрещены

// y - значения функции в узлах сетки

// n - количество узлов сетки

public void BuildSpline(double[] x, double[] y, int n)

{

// Инициализация массива сплайнов

splines = new SplineTuple[n];

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

{

splines[i].x = x[i];

splines[i].a = y[i];

}

splines[0].c = splines[n - 1].c = 0.0;

// Решение СЛАУ относительно коэффициентов сплайнов c[i] методом прогонки для трехдиагональных матриц

// Вычисление прогоночных коэффициентов - прямой ход метода прогонки

double[] alpha = new double[n - 1];

double[] beta = new double[n - 1];

alpha[0] = beta[0] = 0.0;

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

{

double hi = x[i] - x[i - 1];

double hi1 = x[i + 1] - x[i];

double A = hi;

double C = 2.0 \* (hi + hi1);

double B = hi1;

double F = 6.0 \* ((y[i + 1] - y[i]) / hi1 - (y[i] - y[i - 1]) / hi);

double z = (A \* alpha[i - 1] + C);

alpha[i] = -B / z;

beta[i] = (F - A \* beta[i - 1]) / z;

}

// Нахождение решения - обратный ход метода прогонки

for (int i = n - 2; i > 0; --i)

{

splines[i].c = alpha[i] \* splines[i + 1].c + beta[i];

Console.Write("c{1}>> {0:F}\t", splines[i].c, i);

}

Console.WriteLine();

// По известным коэффициентам c[i] находим значения b[i] и d[i]);

for (int i = n - 1; i > 0; --i)

{

double hi = x[i] - x[i - 1];

splines[i].d = (splines[i].c - splines[i - 1].c) / hi;

splines[i].b = hi \* (2.0 \* splines[i].c + splines[i - 1].c) / 6.0 + (y[i] - y[i - 1]) / hi;

Console.Write("d{1}>> {0:F}\t", splines[i].d, i);

Console.WriteLine("b{1}>> {0:F}\t",splines[i].b,i);

}

}

// Вычисление значения интерполированной функции в произвольной точке

public double Interpolate(double x)

{

if (splines == null)

{

return double.NaN; // Если сплайны ещё не построены - возвращаем NaN

}

int n = splines.Length;

SplineTuple s;

if (x <= splines[0].x) // Если x меньше точки сетки x[0] - пользуемся первым эл-тов массива

{

s = splines[0];

}

else if (x >= splines[n - 1].x) // Если x больше точки сетки x[n - 1] - пользуемся последним эл-том массива

{

s = splines[n - 1];

}

else // Иначе x лежит между граничными точками сетки - производим бинарный поиск нужного эл-та массива

{

int i = 0;

int j = n - 1;

while (i + 1 < j)

{

int k = i + (j - i) / 2;

if (x <= splines[k].x)

{

j = k;

}

else

{

i = k;

}

}

s = splines[j];

}

double dx = x - s.x;

// Вычисляем значение сплайна в заданной точке по схеме Горнера (в принципе, "умный" компилятор применил бы схему Горнера сам, но ведь не все так умны, как кажутся)

return s.a + (s.b + (s.c / 2.0 + s.d \* dx / 6.0) \* dx) \* dx;

}

}

Блок-схема:

