Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное  
учреждение высшего образования

Новосибирский государственный технический университет

Кафедра Автоматизированных систем управления

**Отчет по лабораторной работе № 4**

по дисциплине «Вычислительная математика»

Выполнили:

Букова Анна

Ланг Татьяна

Пустовских Дмитрий

Группа: АВТ-813

Преподаватель:

Лукоянычев

Алексей Викторович

Новосибирск

2020

1. **Исходные данные**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x** | **1** | **2** | **3** | **4** | **5** |
| **f** | **4** | **2** | **8** | **1** | **-1** |

1. **Результаты вычислений**

Построим систему уравнений для определения коэффициентов кубических сплайнов.

A= b =

Решив систему методом перегонки, получим:

*c =*

*= 0*

*=*

*=*

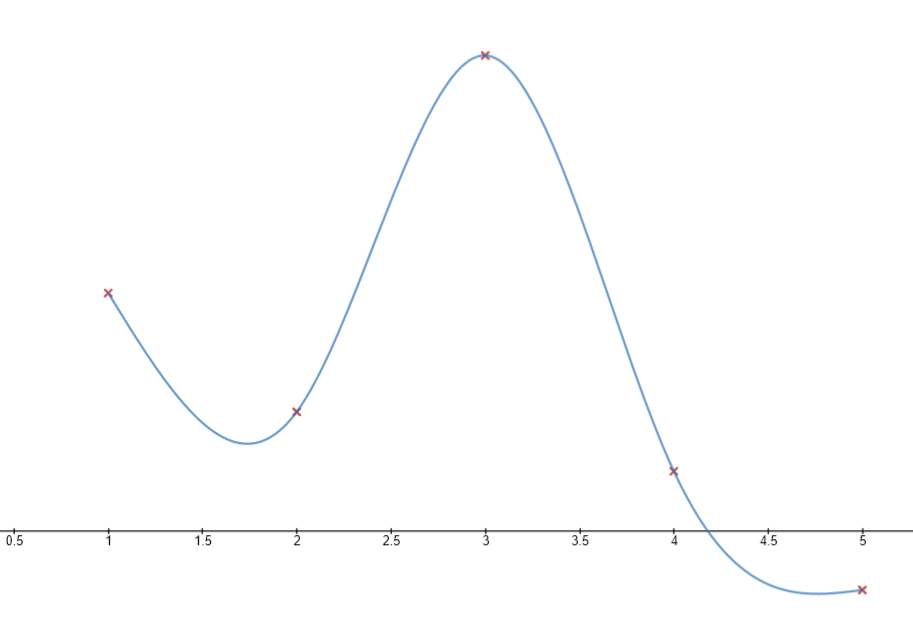
*=*

*= 0*

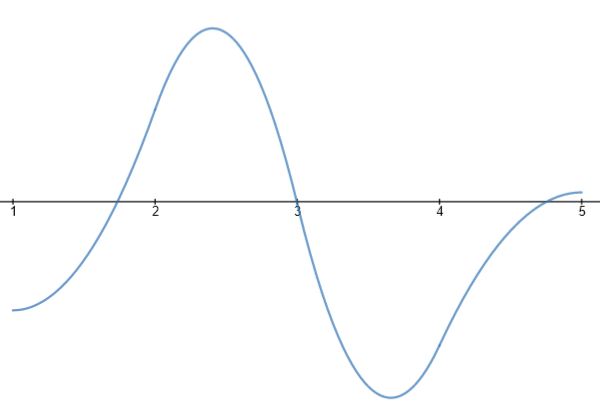
Коэффициенты a, b и d вычисляются после нахождения c по следующим формулам:

Получим коэффициенты:

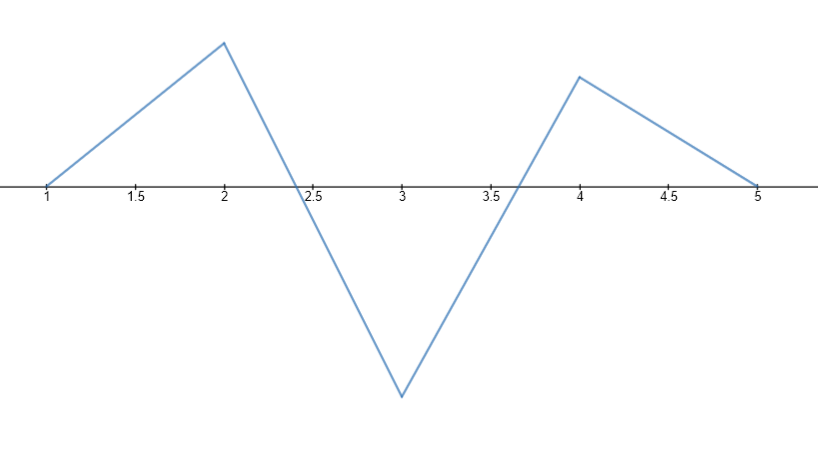
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 4 | -5.1607 | 0 | 3.1607 |
| 2 | 4.32142 | 9.482142 | -7.803571 |
| 8 | -0.125 | -13.92857 | 7.05357 |
| 1 | -6.821428 | 7.23214 | -2.410714 |



1 производная сплайна:



2 производная сплайна:

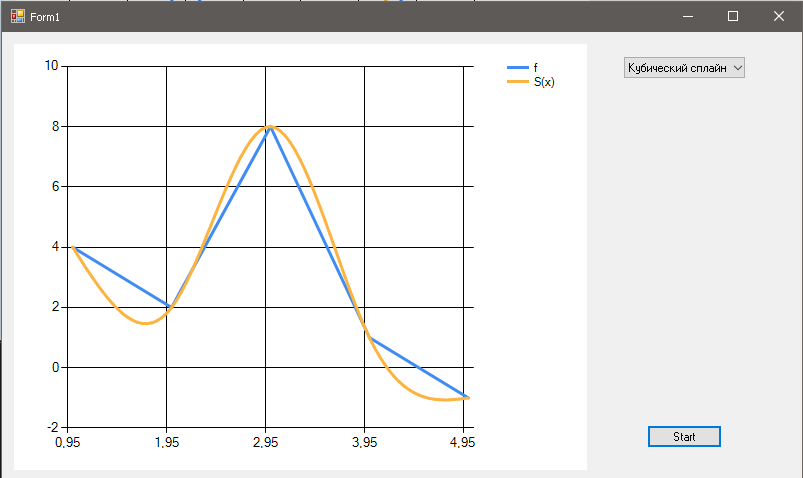


Для вычисления 1 производной были использованы формулы:

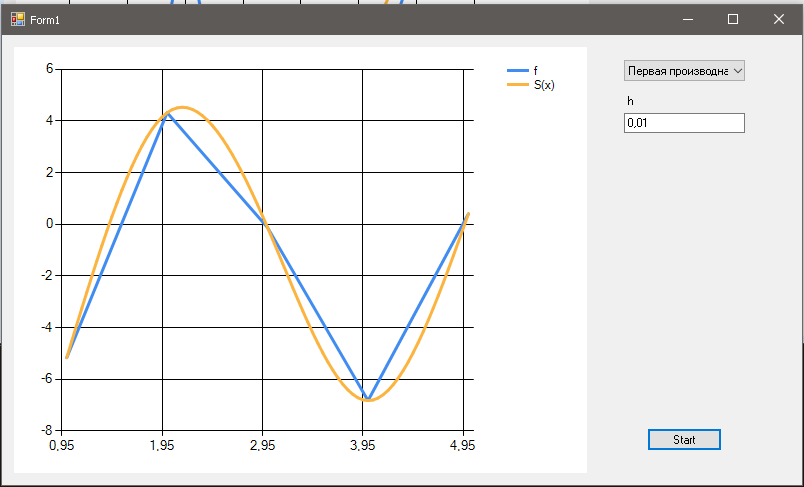
Для вычисления 2 производной были использованы формулы:

1. **Результаты работы программ**

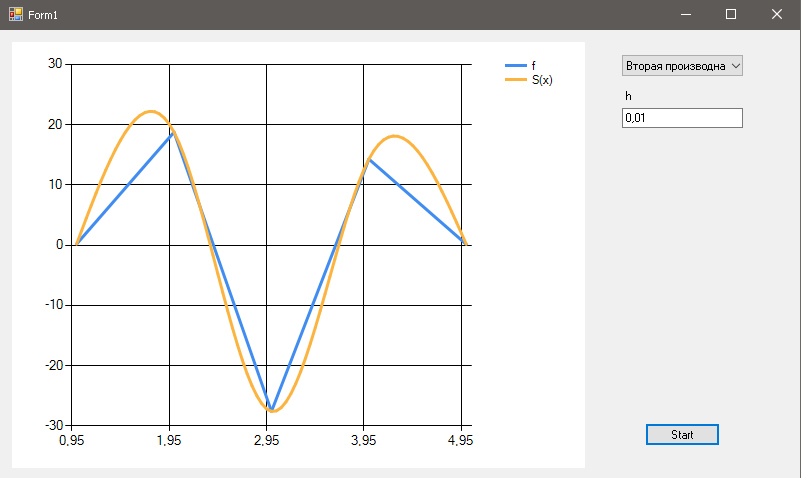
Кубический сплайн:

****

1 производная:

****

2 производная:

****

1. **Выводы**

В данной лабораторной работе мы познакомились с интерполяцией с помощью сплайнами. Также практически реализован на языке С#. Были изучены методы получения 1 и 2 производной.

1. **Тексты программ**

Производные:

public enum Increment { Right, Left, Both\_Sides };

public double Df2(double x, Spline spline, double h, Increment increment)

{

double dF;

switch (increment)

{

case Increment.Right:

dF = (spline.getY(x + h + h) - 2\*spline.getY(x + h) + spline.getY(x)) / (h\*h);

break;

case Increment.Left:

dF = (spline.getY(x) - 2\*spline.getY(x - h) + spline.getY(x - h - h)) / (h\*h);

break;

case Increment.Both\_Sides:

dF = (spline.getY(x + h + h) - 2 \* spline.getY(x) + spline.getY(x - h - h)) / (4 \* h \* h);

break;

default:

dF = 0;

break;

}

return dF;

}

public double Df(double x, Spline spline, double h, Increment increment )

{

double dF;

switch (increment)

{

case Increment.Right:

dF = (spline.getY(x + h) - spline.getY(x)) / h;

break;

case Increment.Left:

dF = (spline.getY(x) - spline.getY(x - h)) / h;

break;

case Increment.Both\_Sides:

dF = (spline.getY(x + h) - spline.getY(x - h)) / (2 \* h);

break;

default:

dF = 0;

break;

}

return dF;

}

Класс Spline:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using lab3\_VM;

namespace lab\_4

{

public class Spline

{

private Polynomial[] polinoms;

private double[,] tabl;

private int count;

public Spline(double[,] tabl, int count)

{

this.tabl = tabl;

this.count = count;

Interpolation inter = new Interpolation(tabl, count);

polinoms = inter.Spline();

}

public double getY(double x)

{

int index = 0;

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

{

if (x <= tabl[0, i])

{

break;

}

index++;

}

double Y = polinoms[index].F(x);

return Y;

}

}}

Класс Polynomial:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab3\_VM

{

public class Polynomial

{

private int Power;

private double[] coefficient = null;

public Polynomial(int pow)

{

this.Power = pow;

coefficient = new double[pow + 1];

for(int i = 0; i < pow + 1; ++i)

{

coefficient[i] = 0;

}

}

public double this[int x]

{

get { return coefficient[x]; }

set { coefficient[x] = value; }

}

public static Polynomial operator \*(Polynomial pol1, Polynomial pol2)

{

var result = new Polynomial(pol1.Power + pol2.Power);

for (int i = 0; i < pol1.Power+1;++i)

{

for(int j = 0; j < pol2.Power+1;++j)

{

result[i + j] = result[i + j] + pol1[i] \* pol2[j];

}

}

return result;

}

public override string ToString()

{

string str = this.coefficient[0].ToString() ;

for(int i = 1; i < this.Power +1; ++i)

{

if( this.coefficient[i] >= 0)

{

str += " + ";

}

str += Math.Round(this.coefficient[i],3).ToString() + "x^" + i.ToString();

}

return str;

}

public static Polynomial operator \*(double pol1, Polynomial pol2)

{

var result = new Polynomial(pol2.Power);

for (int j = 0; j < pol2.Power + 1; ++j)

{

result[j] = result[j] + pol1 \* pol2[j];

}

return result;

}

public static Polynomial operator + ( Polynomial pol1, Polynomial pol2)

{

var result = new Polynomial(Math.Max(pol1.Power, pol2.Power));

for (int i = 0; i < Math.Max(pol1.Power, pol2.Power) + 1; ++i)

{

if(i <= pol1.Power)

{

result[i] += pol1.coefficient[i];

}

if( i <= pol2.Power)

{

result[i] += pol2.coefficient[i];

}

}

return result;

}

public static Polynomial operator +(double pol1, Polynomial pol2)

{

var result = pol2;

result[0] += pol1;

return result;

}

public double F(double x)

{

double summ = 0;

for(int i = 0; i < this.Power + 1 ; ++i)

{

summ += Math.Pow(x,i)\*(this.coefficient[i]);

}

return summ;

}

}

}

Класс Interpolation:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab3\_VM

{

public class Interpolation

{

private double[,] tabl = null;

private int N;

public Interpolation(double[,] tab, int count)

{

tabl = tab;

N = count;

}

public Polynomial Lagrange()

{

Polynomial res = new Polynomial(1);

res[0] = 0;

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

{

Polynomial pol = new Polynomial(0);

pol[0] = 1;

double kf = tabl[1, i];

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

{

if (i != j)

{

Polynomial lag = new Polynomial(1);

lag[0] = (-1) \* tabl[0, j];

lag[1] = 1;

pol = pol \* lag;

kf = kf / (tabl[0, i] - tabl[0, j]);

}

}

res = res + kf \* pol;

}

return res;

}

public Polynomial Newton()

{

Polynomial res = new Polynomial(1);

res[0] = 0;

double[,] tablSeparDiffer = new double[N, N];

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

{

tablSeparDiffer[i, 0] = tabl[1, i];

}

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

{

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

{

tablSeparDiffer[j, i] = (tablSeparDiffer[j + 1, i - 1] - tablSeparDiffer[j, i - 1]) / (tabl[0, j + i] - tabl[0, j]);

}

}

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

{

Polynomial pol = new Polynomial(0);

pol[0] = tablSeparDiffer[0, i];

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

{

Polynomial lag = new Polynomial(1);

lag[0] = (-1) \* tabl[0, j];

lag[1] = 1;

pol = pol \* lag;

}

res = res + pol;

}

return res;

}

public Polynomial Smoothing(int n)

{

Polynomial res = new Polynomial(n);

Matrix matrix = new Matrix(n + 1, n + 1);

Vector vect = new Vector(n + 1);

double[] C = new double[2 \* n + 1];

for (int m = 0; m < 2 \* n + 1; ++m)

{

double sum = 0;

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

{

sum += Math.Pow(tabl[0, i], m);

}

C[m] = sum;

}

for (int k = 0; k < n + 1; ++k)

{

double sum = 0;

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

{

sum += tabl[1, i] \* Math.Pow(tabl[0, i], k);

}

vect[k] = sum;

}

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

{

matrix[i, i] = C[i + i];

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

{

matrix[i, j] = C[i + j];

matrix[j, i] = C[i + j];

}

}

SLAE lsolve = new SLAE(matrix, vect);

Vector aprmn = lsolve.GaussMethod();

for (int i = 0; i < aprmn.N; i++)

{

res[i] = aprmn[i];

}

return res;

}

public Polynomial[] Spline()

{

Polynomial[] polinoms = new Polynomial[N - 1];

double h = tabl[0, 1] - tabl[0, 0];

double[] C = new double[N];

C[0] = 0;

C[N-1] = 0;

Matrix mat = new Matrix(N - 2, N - 2);

Vector vet = new Vector(N - 2);

Vector result;

for (int i = 0; i < mat.M; ++i)

{

mat[i, i] = 4 \* h;

if (i < mat.M - 1)

{

mat[i, i + 1] = h;

mat[i + 1, i] = h;

}

vet[i] = (3 / h) \* (tabl[1, i + 2] - 2 \* tabl[1, i + 1] + tabl[1, i]);

}

SLAE re = new SLAE(mat, vet);

result = re.Sweeps();

for (int i = 0; i < result.N; ++i)

{

C[i + 1] = result[i];

}

Matrix odds = new Matrix(N - 1, 4);

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

{

odds[i, 0] = tabl[1, i];

odds[i, 1] = ((tabl[1, i + 1] - tabl[1, i]) / h) - ((C[i + 1] + 2 \* C[i]) \* h) / 3;

odds[i, 2] = C[i];

odds[i, 3] = (C[i + 1] - C[i]) / (3 \* h);

}

for (int k = 0; k < N - 1; ++ k)

{

Polynomial res = new Polynomial(0);

res[0] = 0;

for (int i = 0; i < 4; ++i)

{

Polynomial pol = new Polynomial(0);

pol[0] = odds[k, i];

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

{

Polynomial pol2 = new Polynomial(1);

pol2[0] = -tabl[0, k];

pol2[1] = 1;

pol = pol \* pol2;

}

res = res + pol;

}

polinoms[k] = res;

}

return polinoms;

}

}

}