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 № 3**

по дисциплине «Вычислительная математика»

Выполнили:

Букова Анна

Ланг Татьяна

Пустовских Дмитрий

Группа: АВТ-813

Преподаватель:

Лукоянычев

Алексей Викторович

Новосибирск

2020

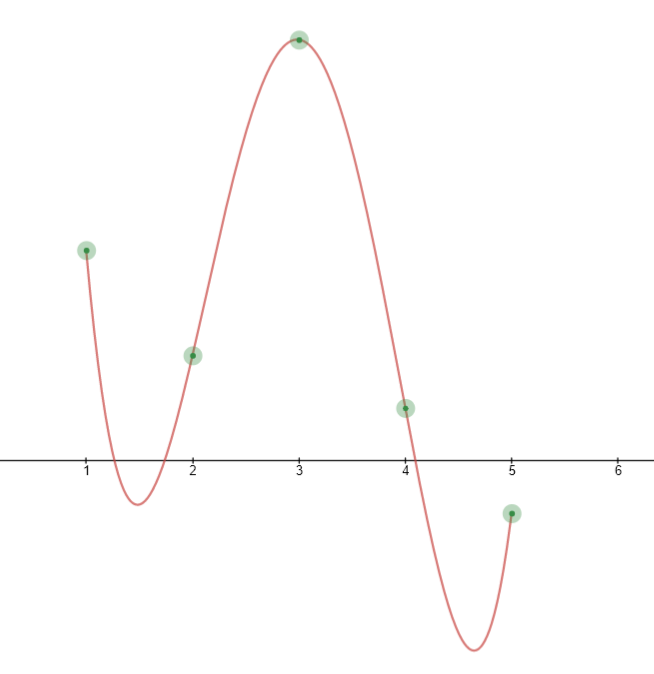
1. **Исходные данные**

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

1. **Результаты вычислений**

Интерполяционного многочлена с помощью формулы Лагранжа:

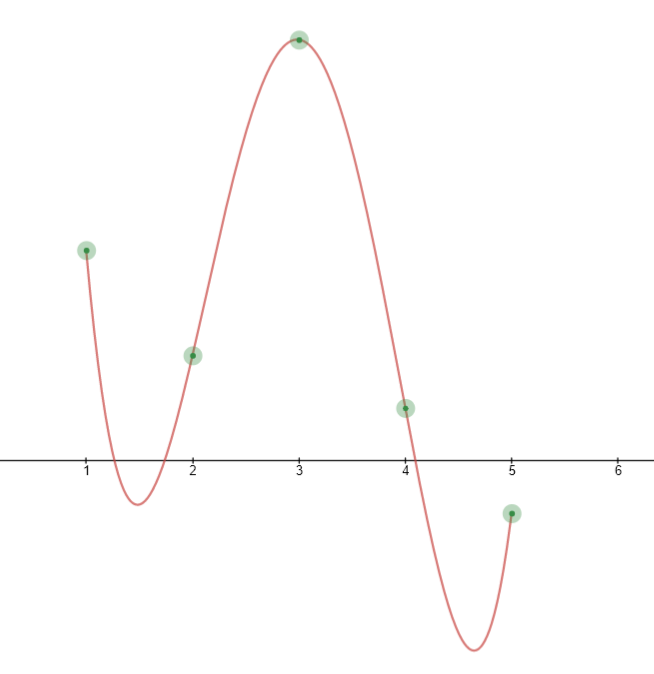
Сократив данное выражение получим:



Интерполяционного многочлена с помощью формулы Ньютона:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| =4  =2  =8  =1  =-1 | =-2  =6  =-7  =-2 | =4  =-6.5  =2.5 | =-3.5  =3 | .625 |

Из этой таблицы получаем многочлен:

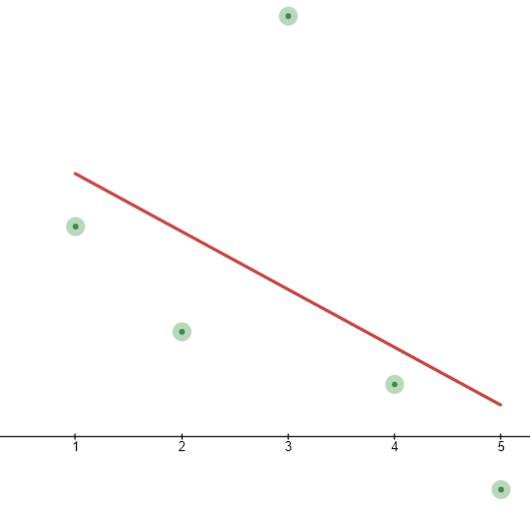


Сглаживающих многочленов 1, 2, 3 степени по методу наименьших квадратов:

1 степень: A= b =

Вычислив СЛАУ получим:

lsolve(A,b) = S(x) =

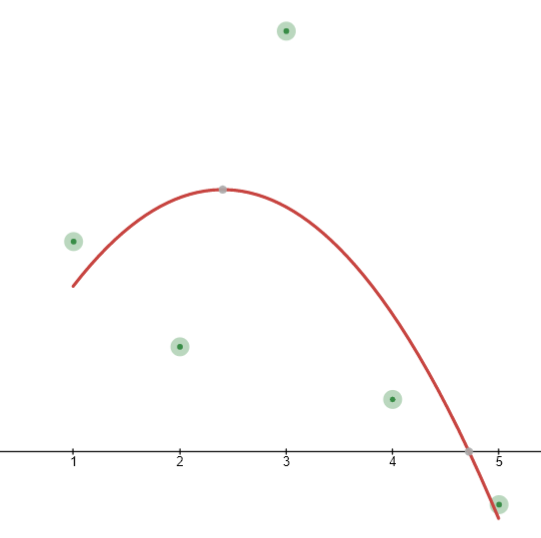


2 степень:

A= b =

Вычислив СЛАУ получим:

lsolve(A,b) = S(x) =

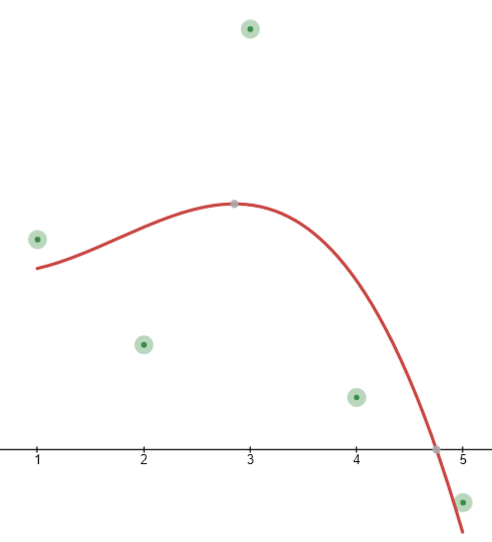


3 степень:

A= b =

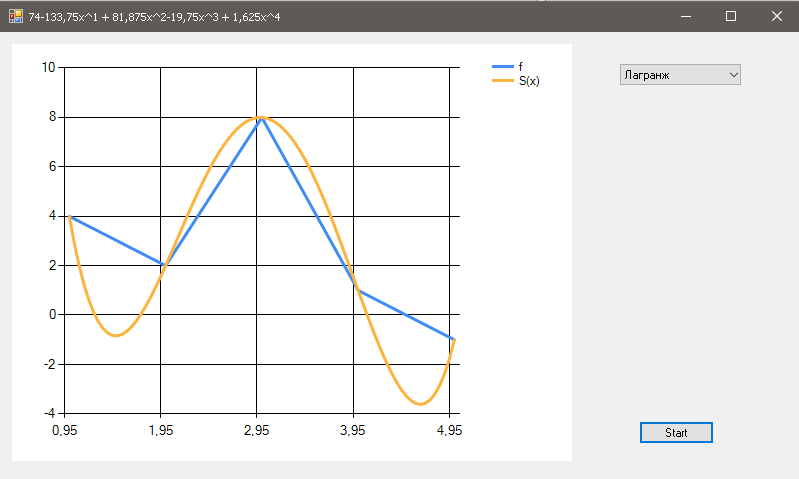
Вычислив СЛАУ получим:

lsolve(A,b) = S(x) =

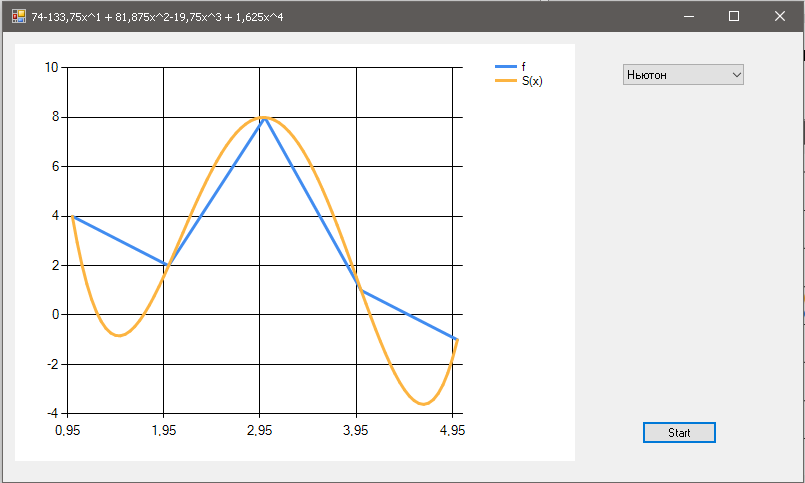


1. **Результаты работы программ**

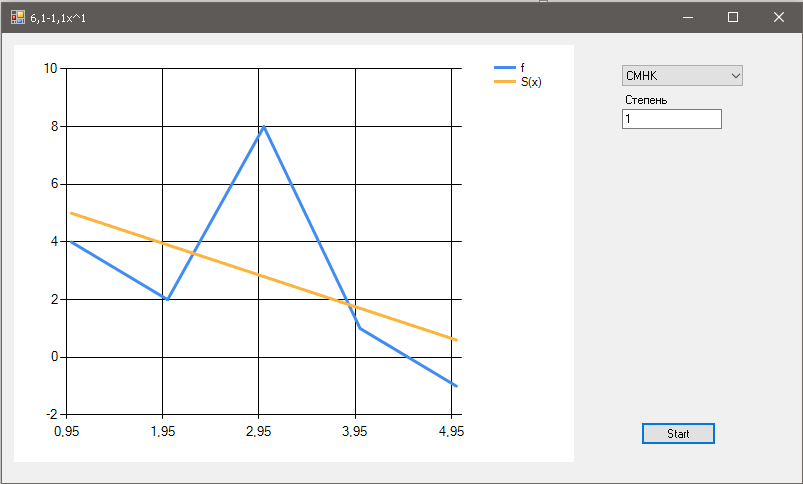
Лагранж:

****

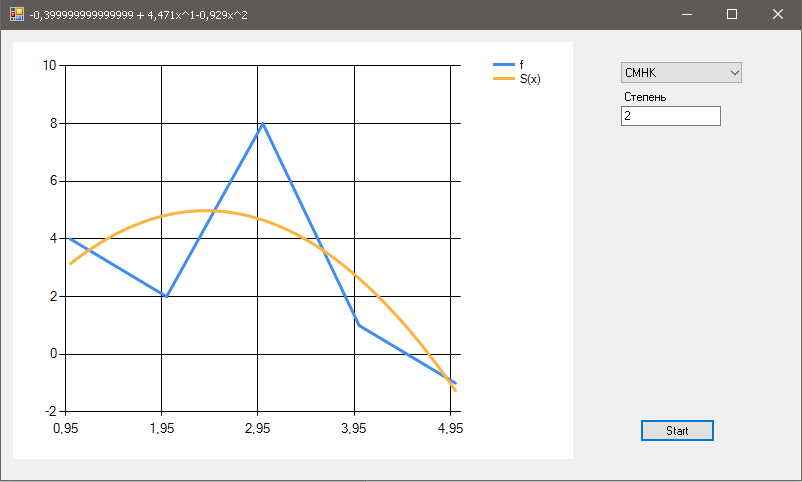
Ньютон:

****

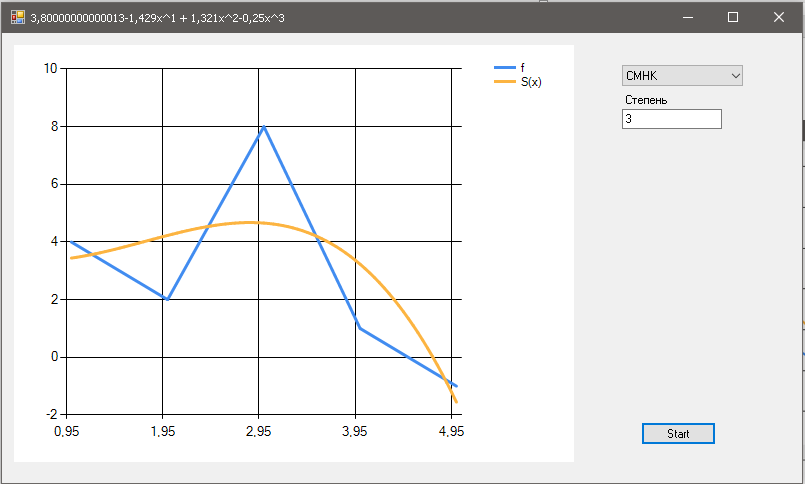
Сглаживанием 1 степени:

****

Сглаживанием 2 степени:

****

Сглаживанием 3 степени:

****

1. **Выводы**

В данной лабораторной работе мы познакомились с различными методами интерполяция. Также практически реализовали методы на языке С#. При рассмотрении все методы в результате свей работы мы получаем один и тот же полином. Если выбирать метод интерполяции, то скорее всего лучше Ньютон, так как при увеличении количества точек, достаточно дополнить таблицу разделенных разностей.

1. **Тексты программ**

**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;

}

}

}

**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;

}

}

}