Міністерство освіти, науки, молоді та спорту України
Національний технічний університет України
«Київський політехнічний інститут»
Фізико-технічний інститут

Лабораторна робота №5

Методи обчислень
«Інтерполяція»

Варіант 4

Підготував:
студент 3 курсу
групи ФІ-84
Коломієць Андрій Юрійович

Викладач:
Стьопочкіна Ірина Валеріївна

Київ – 2021

Вхідні данні

$x\sqrt{x}$	[0,4]

Хід виконання

Побудова таблиці значень функції

Відрізок інтерполяції розбити не менш ніж на 10 вузлів потрібно.

Використовуючи аналітичне задання функції, визначене варіантом, побудуємо таблицю значень функції у вузлах на відповідному відрізку інтерполяції.

У зазначеній лабораторній роботі, кількість вузлів буде дорівнювати нехай 12.

```
Program start...

Input number of nodes:
12
```

```
Find table of values and result of function on curent values:

value   result function
    0             0
 0.363636   0.219281
 0.727273   0.62022
 1.09091    1.13942
 1.45455    1.75425
 1.81818    2.45164
 2.18182    3.22276
 2.54545    4.06114
 2.90909    4.96176
 3.27273    5.92059
 3.63636    6.93427
   4            8
```

Інтерполяція Лагранжом

Тепер побудуємо за таблично заданою функцією інтерполяційний поліном $P_n(x)$ у формі Лагранжа (фомула в реалізації її виводу на екран досить завелика, аби помістится на аркуші паперу, тому нижче кожен доданок виділено окремим кольором)

```
Image of interpolation polynom Lagrange:

Function(x)=((x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x -
1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x -
3.636364)(x - 4.000000)/ (0.000000 - 0.363636)(0.000000 - 0.727273)(0.000000
- 1.090909)(0.000000 - 1.454545)(0.000000 - 1.818182)(0.000000 -
2.181818)(0.000000 - 2.545455)(0.000000 - 2.909091)(0.000000 -
3.272727)(0.000000 - 3.636364)(0.000000 - 4.000000))*0.000000+
+( (x - 0.000000)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x -
2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x -
4.000000)/ (0.363636 - 0.000000)(0.363636 - 0.727273)(0.363636 -
```

```
1.090909)(0.363636 - 1.454545)(0.363636 - 1.818182)(0.363636 -
2.181818)(0.363636 - 2.545455)(0.363636 - 2.909091)(0.363636 -
3.272727)(0.363636 - 3.636364)(0.363636 - 4.000000))*0.219281+
+( (x - 0.000000)(x - 0.363636)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x -
2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x -
4.000000)/ (0.727273 - 0.000000)(0.727273 - 0.363636)(0.727273 -
1.090909)(0.727273 - 1.454545)(0.727273 - 1.818182)(0.727273 -
2.181818)(0.727273 - 2.545455)(0.727273 - 2.909091)(0.727273 -
3.272727)(0.727273 - 3.636364)(0.727273 - 4.000000))*0.620220+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.454545)(x - 1.818182)(x -
2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x -
4.000000)/ (1.090909 - 0.000000)(1.090909 - 0.363636)(1.090909 -
0.727273)(1.090909 - 1.454545)(1.090909 - 1.818182)(1.090909 -
2.181818)(1.090909 - 2.545455)(1.090909 - 2.909091)(1.090909 -
3.272727)(1.090909 - 3.636364)(1.090909 - 4.000000))*1.139417+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.818182)(x -
2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x -
4.000000)/ (1.454545 - 0.000000)(1.454545 - 0.363636)(1.454545 -
0.727273)(1.454545 - 1.090909)(1.454545 - 1.818182)(1.454545 -
2.181818)(1.454545 - 2.545455)(1.454545 - 2.909091)(1.454545 -
3.272727)(1.454545 - 3.636364)(1.454545 - 4.000000))*1.754248+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x -
2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x -
4.000000)/ (1.818182 - 0.000000)(1.818182 - 0.363636)(1.818182 -
0.727273)(1.818182 - 1.090909)(1.818182 - 1.454545)(1.818182 -
2.181818)(1.818182 - 2.545455)(1.818182 - 2.909091)(1.818182 -
3.272727)(1.818182 - 3.636364)(1.818182 - 4.000000))*2.451636+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x -
1.818182)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x -
4.000000)/ (2.181818 - 0.000000)(2.181818 - 0.363636)(2.181818 -
0.727273)(2.181818 - 1.090909)(2.181818 - 1.454545)(2.181818 -
1.818182)(2.181818 - 2.545455)(2.181818 - 2.909091)(2.181818 -
3.272727)(2.181818 - 3.636364)(2.181818 - 4.000000))*3.222759+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x -
1.818182)(x - 2.181818)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x -
4.000000)/ (2.545455 - 0.000000)(2.545455 - 0.363636)(2.545455 -
0.727273)(2.545455 - 1.090909)(2.545455 - 1.454545)(2.545455 -
1.818182)(2.545455 - 2.181818)(2.545455 - 2.909091)(2.545455 -
3.272727)(2.545455 - 3.636364)(2.545455 - 4.000000))*4.061141+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x -
1.818182)(x - 2.181818)(x - 2.545455)(x - 3.272727)(x - 3.636364)(x -
4.000000)/ (2.909091 - 0.000000)(2.909091 - 0.363636)(2.909091 -
0.727273)(2.909091 - 1.090909)(2.909091 - 1.454545)(2.909091 -
1.818182)(2.909091 - 2.181818)(2.909091 - 2.545455)(2.909091 -
3.272727)(2.909091 - 3.636364)(2.909091 - 4.000000))*4.961762+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x -
1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.636364)(x -
4.000000)/ (3.272727 - 0.000000)(3.272727 - 0.363636)(3.272727 -
0.727273)(3.272727 - 1.090909)(3.272727 - 1.454545)(3.272727 -
1.818182)(3.272727 - 2.181818)(3.272727 - 2.545455)(3.272727 -
2.909091)(3.272727 - 3.636364)(3.272727 - 4.000000))*5.920587+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x -
1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x -
4.000000)/ (3.636364 - 0.000000)(3.636364 - 0.363636)(3.636364 -
0.727273)(3.636364 - 1.090909)(3.636364 - 1.454545)(3.636364 -
1.818182)(3.636364 - 2.181818)(3.636364 - 2.545455)(3.636364 -
2.909091)(3.636364 - 3.272727)(3.636364 - 4.000000))*6.934275+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x -
1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x -
3.636364)/ (4.000000 - 0.000000)(4.000000 - 0.363636)(4.000000 -
0.727273)(4.000000 - 1.090909)(4.000000 - 1.454545)(4.000000 -
1.818182)(4.000000 - 2.181818)(4.000000 - 2.545455)(4.000000 -
2.909091)(4.000000 - 3.272727)(4.000000 - 3.636364))*8.00000
```

Інтерполяція Spline функціями

Тепер побудуємо за таблично заданою функцією інтерполяційний поліном $P_n(x)$ у формі Spline функцій (кожен рядок відповідає коефіцієнтам Spline функції):

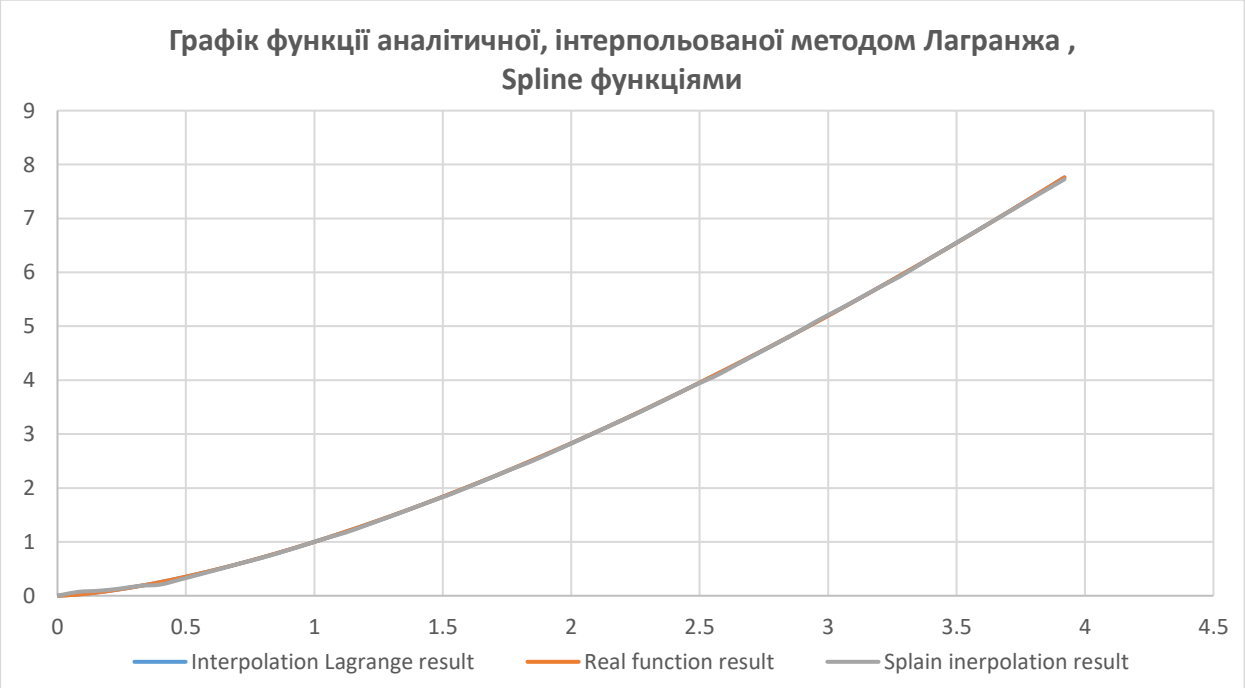
Coefficient of all spline function:

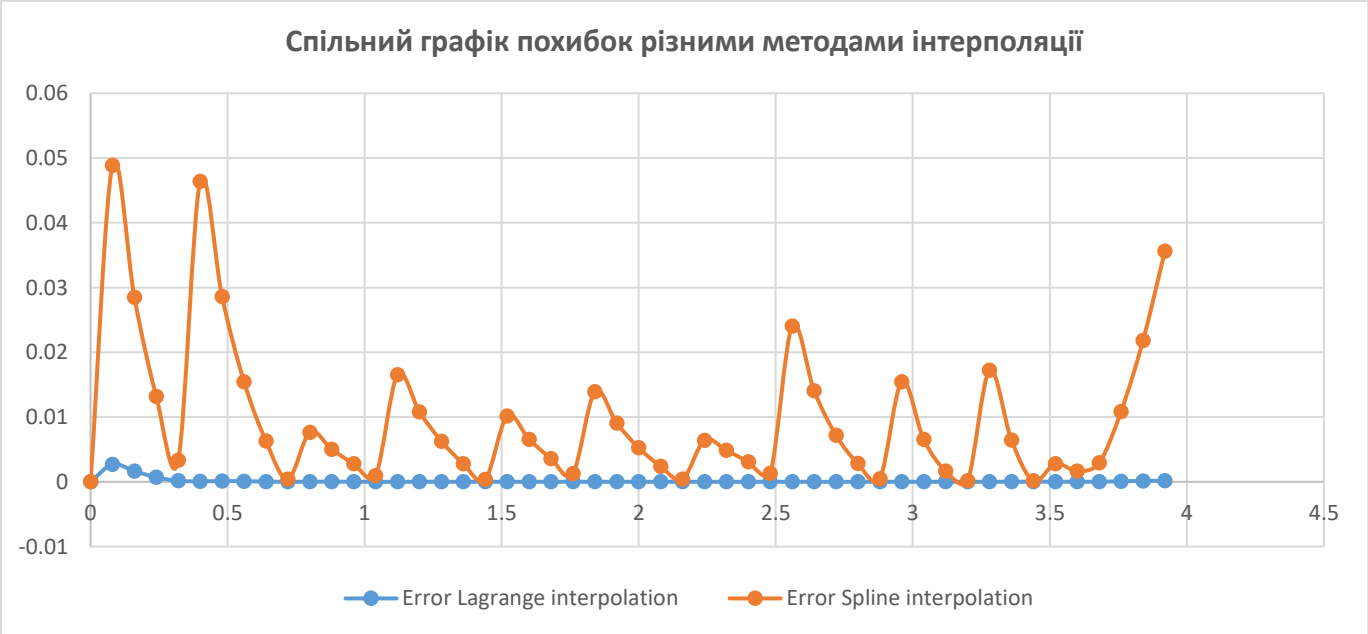
a	b	c	d
0	0.606809	-0.491145	1.32201
0.219281	0.843627	0.951052	-0.657036
0.62022	1.33192	0.234285	0.0807778
1.13942	1.58346	0.322406	-0.0749925
1.75425	1.83192	0.240596	-0.0120238
2.45164	2.0417	0.227479	-0.0289916
3.22276	2.2329	0.195852	0.0107954
4.06114	2.41172	0.207629	-0.07951
4.96176	2.57145	0.120891	0.16153
5.92059	2.72405	0.297105	-0.336159
6.93427	2.80452	-0.0696132	0.0638121

Вказана послідовність коефіцієнтів відповідає відповідно їх порядку в такій системі:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \ i = 0, 1, \ldots, n-1.$$

Графіки функцій та похибок





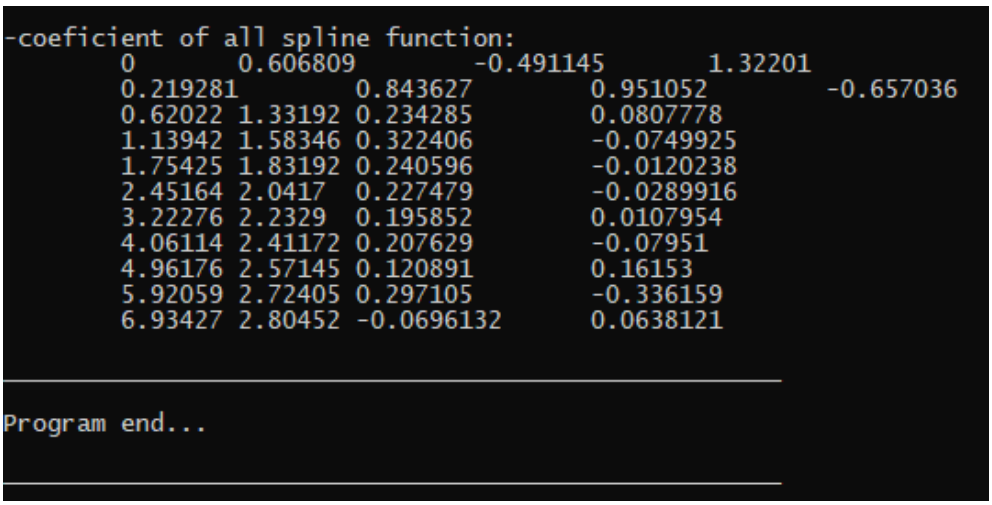
Обчислення похибок відбувалося з кроком: `0.01`.

Максимальні значення похибок від аналітично заданої функції можна визначити з графіків.

Результати роботи

```
Program start...

Input number of nodes:
12

Find table of values and result of function on curent values:

value    result function
0        0
0.363636        0.219281
0.727273        0.62022
1.09091 1.13942
1.45455 1.75425
1.81818 2.45164
2.18182 3.22276
2.54545 4.06114
2.90909 4.96176
3.27273 5.92059
3.63636 6.93427
4       8

Lagrange interpolation:

-image of interpolation polynom:

Function(x)=
+( (x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (0.000000 - 0.363636)(0.000000 - 0.727273)(0.000000 - 1.090909)(0.000000 - 1.454545)(0.000000 - 1.818182)(0.000000 - 2.181818)(0.000000 - 2.545455)(0.000000 - 2.909091)(0.000000 - 3.272727)(0.000000 - 3.636364)(0.000000 - 4.000000))*0.000000+
+( (x - 0.000000)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (0.363636 - 0.000000)(0.363636 - 0.727273)(0.363636 - 1.090909)(0.363636 - 1.454545)(0.363636 - 1.818182)(0.363636 - 2.181818)(0.363636 - 2.545455)(0.363636 - 2.909091)(0.363636 - 3.272727)(0.363636 - 3.636364)(0.363636 - 4.000000))*0.219281+
+( (x - 0.000000)(x - 0.363636)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (0.727273 - 0.000000)(0.727273 - 0.363636)(0.727273 - 1.090909)(0.727273 - 1.454545)(0.727273 - 1.818182)(0.727273 - 2.181818)(0.727273 - 2.545455)(0.727273 - 2.909091)(0.727273 - 3.272727)(0.727273 - 3.636364)(0.727273 - 4.000000))*0.620220+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (1.090909 - 0.000000)(1.090909 - 0.363636)(1.090909 - 0.727273)(1.090909 - 1.454545)(1.090909 - 1.818182)(1.090909 - 2.181818)(1.090909 - 2.545455)(1.090909 - 2.909091)(1.090909 - 3.272727)(1.090909 - 3.636364)(1.090909 - 4.000000))*1.139417+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (1.454545 - 0.000000)(1.454545 - 0.363636)(1.454545 - 0.727273)(1.454545 - 1.090909)(1.454545 - 1.818182)(1.454545 - 2.181818)(1.454545 - 2.545455)(1.454545 - 2.909091)(1.454545 - 3.272727)(1.454545 - 3.636364)(1.454545 - 4.000000))*1.754248+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (1.818182 - 0.000000)(1.818182 - 0.363636)(1.818182 - 0.727273)(1.818182 - 1.090909)(1.818182 - 1.454545)(1.818182 - 2.181818)(1.818182 - 2.545455)(1.818182 - 2.909091)(1.818182 - 3.272727)(1.818182 - 3.636364)(1.818182 - 4.000000))*2.451636+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (2.181818 - 0.000000)(2.181818 - 0.363636)(2.181818 - 0.727273)(2.181818 - 1.090909)(2.181818 - 1.454545)(2.181818 - 1.818182)(2.181818 - 2.545455)(2.181818 - 2.909091)(2.181818 - 3.272727)(2.181818 - 3.636364)(2.181818 - 4.000000))*3.222759+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.909091)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (2.545455 - 0.000000)(2.545455 - 0.363636)(2.545455 - 0.727273)(2.545455 - 1.090909)(2.545455 - 1.454545)(2.545455 - 1.818182)(2.545455 - 2.181818)(2.545455 - 2.909091)(2.545455 - 3.272727)(2.545455 - 3.636364)(2.545455 - 4.000000))*4.061141+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 3.272727)(x - 3.636364)(x - 4.000000)/ (2.909091 - 0.000000)(2.909091 - 0.363636)(2.909091 - 0.727273)(2.909091 - 1.090909)(2.909091 - 1.454545)(2.909091 - 1.818182)(2.909091 - 2.181818)(2.909091 - 2.545455)(2.909091 - 3.272727)(2.909091 - 3.636364)(2.909091 - 4.000000))*4.961762+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.636364)(x - 4.000000)/ (3.272727 - 0.000000)(3.272727 - 0.363636)(3.272727 - 0.727273)(3.272727 - 1.090909)(3.272727 - 1.454545)(3.272727 - 1.818182)(3.272727 - 2.181818)(3.272727 - 2.545455)(3.272727 - 2.909091)(3.272727 - 3.636364)(3.272727 - 4.000000))*5.920586+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 4.000000)/ (3.636364 - 0.000000)(3.636364 - 0.363636)(3.636364 - 0.727273)(3.636364 - 1.090909)(3.636364 - 1.454545)(3.636364 - 1.818182)(3.636364 - 2.181818)(3.636364 - 2.545455)(3.636364 - 2.909091)(3.636364 - 3.272727)(3.636364 - 4.000000))*6.934273+
+( (x - 0.000000)(x - 0.363636)(x - 0.727273)(x - 1.090909)(x - 1.454545)(x - 1.818182)(x - 2.181818)(x - 2.545455)(x - 2.909091)(x - 3.272727)(x - 3.636364)/ (4.000000 - 0.000000)(4.000000 - 0.363636)(4.000000 - 0.727273)(4.000000 - 1.090909)(4.000000 - 1.454545)(4.000000 - 1.818182)(4.000000 - 2.181818)(4.000000 - 2.545455)(4.000000 - 2.909091)(4.000000 - 3.272727)(4.000000 - 3.636364))*8.000000

-input value, where you want to get result of interpolation:
3

-result of interpolation:5.19616

-error of interpolation (maximum value of difference root):0.00268123
```

```
Splain interpolation:

-coeficient A of splain polynoms:
0
0.219281
0.62022
1.13942
1.75425
2.45164
3.22276
4.06114
4.96176
5.92059
6.93427

-coeficient h of splain polynoms:
0.363636
0.363636
0.363636
0.363636
0.363636
0.363636
0.363636
0.363636
0.363636
0.363636

-right vector of system, which we must solve:
0
1.49868
0.975627
0.788975
0.6811
0.608315
0.554881
0.513481
0.480172
0.452617
0

-matrix of system, which we must solve:
1.45455 0.363636        0       0       0       0       0       0       0       0       0       0
0.363636        1.45455 0.363636        0       0       0       0       0       0       0       0       0
0       0.363636        1.45455 0.363636        0       0       0       0       0       0       0       0
0       0       0.363636        1.45455 0.363636        0       0       0       0       0       0       0
0       0       0       0.363636        1.45455 0.363636        0       0       0       0       0       0
0       0       0       0       0.363636        1.45455 0.363636        0       0       0       0       0
0       0       0       0       0       0.363636        1.45455 0.363636        0       0       0       0
0       0       0       0       0       0       0.363636        1.45455 0.363636        0       0       0
0       0       0       0       0       0       0       0.363636        1.45455 0.363636        0       0
0       0       0       0       0       0       0       0       0.363636        1.45455 0.363636        0
0       0       0       0       0       0       0       0       0       0.363636        1.45455 0.363636
0       0       0       0       0       0       0       0       0       0       0.363636        1.45455

-solved system:
-0.253382
1.01353
0.320644
0.386872
0.301549
0.279958
0.251486
0.240021
0.2005
0.278453
-0.0696132
```



Код

```
#include<iostream>
#include <string>
#include <math.h>
#include <fstream>

using namespace std;

//define block

#define type  long double
#define                                                                    line
cout<<endl<<"_____________________________________________________"<<endl;

// general function block

void null_vector(type* vector, int size)
{
	for (int i = 0; i < size; i++)
	{
		vector[i] = 0;
	}
}

void output_vector(type* vector, int size)
{
	for (int i = 0; i < size; i++)
	{
		cout << vector[i] << endl;
	}
}

type* copy_vector(type* vector_in, type* vector_out, int size)
{
	for (int i = 0; i < size; i++)
	{
		vector_out[i] = vector_in[i];
	}

	return vector_out;
}

void  null_matrix(type** matrix, int size)
{
	for (int i = 0; i < size; i++)
	{
```

```
            for (int j = 0; j < size; j++)
            {
                  matrix[i][j] = 0;
            }
      }
}

void  output_matrix(type** matrix, int size)
{
      for (int i = 0; i < size; i++)
      {
            for (int j = 0; j < size; j++)
            {
                  cout << matrix[i][j] << "\t";
            }
            cout << endl;
      }
}

type** copy_matrix(type** matrix_in, type** matrix_out, int size)
{
      for (int i = 0; i < size; i++)
      {
            for (int j = 0; j < size; j++)
            {
                  matrix_out[i][j] = matrix_in[i][j];
            }
      }
      return matrix_out;
}

void output_paralel_vector(type* vector_first, type* vector_second, int size)
{
      for (int i = 0; i < size; i++)
      {
            cout << vector_first[i]<<"\t"<< vector_second[i]<< endl;
      }
}

type function(type value)
{
      return value * sqrt(value);
}

void data_table(type* vector_value, type* vector_result, type interval, int
size)
{
      null_vector(vector_value, size);
      null_vector(vector_result, size);

      type temp_interval = interval / (size-1);

      for (int i = 0; i < size; i++)
      {
            switch (i)
            {
            case 0:
            {
                  vector_value[0] = 0;
                  break;
            }
            case 1:
            {
                  vector_value[1] = temp_interval;
```

```
                  break;
            }

            default:
                  vector_value[i] = vector_value[i-1]+temp_interval;
                  break;
            }

      }

      for (int i = 0; i < size; i++)
      {
            vector_result[i] = function(vector_value[i]);
      }
}

//lagrange interpolation algorithm

string image_of_polynom_lagrange(type* value, type* result, int size)
{
      string image_of_polynom = " ";



      for (int i = 0; i < size; i++)
      {
            string temp_first = " ";
            string temp_second = " ";

            string L=" ";

            for (int j = 0; j < size; j++)
            {
                  if (j != i)
                  {
                        temp_first = temp_first + "(x - " + to_string(value[j])
+ ")";
                        temp_second = temp_second + "(" + to_string(value[i]) +
" - " + to_string(value[j]) + ")";
                  }
            }

            L = "("+temp_first + "/" + temp_second+")";

            image_of_polynom    =    image_of_polynom+"\n"+"+"    +L    +"*"+
to_string(result[i])+"+";
      }

      return image_of_polynom;
}

type lagrange(type* value, type* result, int size, type choose_elements)
{

      type result_of_lagrange_polynom_on_current_element = 0.0;


      for (int i = 0; i <size; i++)
      {
            type L = 1.0;

            for (int j = 0; j <size; j++)
            {
                  if (j != i)
```

```
                {
                        L *= (choose_elements - value[j]) / (value[i] -
value[j]);
                }
            }

            result_of_lagrange_polynom_on_current_element=
result_of_lagrange_polynom_on_current_element+L * result[i];
      }

      return result_of_lagrange_polynom_on_current_element;
}


//splain interpolation algorithm

type * method_Gause(type **matrix, type *vector, int size)
{
      type *solve, max;
      int m, index; m = 0;

      const type epsilone = 0.0001;

      solve = new type[size];

      while (m < size)
      {
            // Поиск строки с максимальным a[i][k]
            max = abs(matrix[m][m]);
            index = m;
            for (int i = m + 1; i < size; i++)
            {
                  if (abs(matrix[i][m]) > max)
                  {
                        max = abs(matrix[i][m]);
                        index = i;
                  }
            }
            // Перестановка строк

            if (max < epsilone)
            {
                  // нет ненулевых диагональных элементов
                  cout << "The solution cannot be obtained, because of the zero
column:";
                  cout << index << "Matrix" << endl;
                  return 0;
            }
            for (int j = 0; j < size; j++)
            {
                  type temp = matrix[m][j];
                  matrix[m][j] = matrix[index][j];
                  matrix[index][j] = temp;
            }
            type temp = vector[m];
            vector[m] = vector[index];
            vector[index] = temp;

            // Нормализация уравнений
            for (int i = m; i < size; i++)
            {
                  type temp = matrix[i][m];
                  if (abs(temp) < epsilone) continue; // для нулевого
коэффициента пропустить
```

```
                for (int j = 0; j < size; j++)
                        matrix[i][j] = matrix[i][j] / temp;
                vector[i] = vector[i] / temp;
                if (i == m)  continue; // уравнение не вычитать само из себя
                for (int j = 0; j < size; j++)
                        matrix[i][j] = matrix[i][j] - matrix[m][j];
                vector[i] = vector[i] - vector[m];
            }
            m++;
        }
        // обратная подстановка
        for (m = size - 1; m >= 0; m--)
        {
            solve[m] = vector[m];
            for (int i = 0; i < m; i++)
                vector[i] = vector[i] - matrix[i][m] * solve[m];
        }
        return solve;
}

void   coefitient_splain(type*   vector_result,   type*   vector_value,type**
coeficient_matrix,int size)
{
        //data block
        int space = size - 1;

        //creation block
        type* a = new type[space];              null_vector(a, space);
        type* b = new type[space];              null_vector(b, space);
        type* c = new type[space];              null_vector(c, space);
        type* d = new type[space];              null_vector(d, space);
        type* h = new type[space];              null_vector(h, space);
        type*right_vector = new type[space];    null_vector(right_vector, space);

        type** matrix = new type*[space];

        for (int i = 0; i < space; i++)
        {
            matrix[i] = new type[space];
        }

        type** matrix_coeficient = new type*[space];

        for (int i = 0; i < space; i++)
        {
            matrix_coeficient[i] = new type[4];
        }

        null_matrix(matrix, space);

        for (int i = 0; i < space; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                matrix[i][j] = 0;
            }
        }

        ////// interpolation algorithm block start //////

        for (int i = 0; i < space; i++) //exelent
        {
            a[i] = vector_result[i];
            h[i] = vector_value[i + 1] - vector_value[i];
```

```
	}

	cout << endl << "-coeficient A of splain polynoms:" << endl;
	output_vector(a, space);
	cout << endl << "-coeficient h of splain polynoms:" << endl;
	output_vector(h, space);


	right_vector[0] = 0; right_vector[space - 1] = 0;

	for (int i = 1; i < space - 1; i++)
	{
		right_vector[i] = 3 * (((vector_result[i + 1] - vector_result[i])
/ h[i + 1]) - ((vector_result[i] - vector_result[i - 1]) / h[i]));
	}

	cout << endl << "-right vector of system, which we must solve:" << endl;
	output_vector(right_vector, space);

	for (int i = 0; i < space; i++)
	{
		for (int j = 0; j < space; j++)
		{
			if (i == j && i > 0 && i < space - 1)
			{
				matrix[i][j - 1] = h[i];
				matrix[i][j] = 2 * (h[i] + h[i + 1]);
				matrix[i][j + 1] = h[i + 1];
			}
			if (i == j && i == 0)
			{
				matrix[i][j] = 2 * (h[i] + h[i + 1]);
				matrix[i][j + 1] = h[i + 1];
			}
			if (i == j && i == space - 1)
			{
				matrix[i][j] = 2 * (h[i - 1] + h[i]);
				matrix[i][j - 1] = h[i];
			}
		}
	}

	cout << endl << "-matrix of system, which we must solve:" << endl;
	output_matrix(matrix, space);

	cout << endl << "-solved system:" << endl;
	output_vector(method_Gause(matrix, right_vector, space), space);

	copy_vector(method_Gause(matrix, right_vector, space), c, space);

	d[space - 1] = -(c[space - 1]) / (3 * h[space - 1]);
	b[space - 1] = ((vector_result[space - 1] - vector_result[space - 2]) /
h[space - 1]) - (2 * h[space - 1] * c[space - 1] / 3);

	for (int i = 0; i < space; i++)
	{
		if (i != space - 1)
		{
			d[i] = (c[i + 1] - c[i]) / (3 * h[i]);
			b[i] = ((vector_result[i + 1] - vector_result[i]) / h[i]) -
(h[i] * (c[i + 1] + 2 * c[i]) / 3);
		}

		matrix_coeficient[i][0] = a[i];
```

```
            matrix_coeficient[i][1] = b[i];
            matrix_coeficient[i][2] = c[i];
            matrix_coeficient[i][3] = d[i];
      }

      cout << endl << "-coeficient of all spline function:" << endl;

      for (int i = 0; i < space; i++)
      {
            for (int j = 0; j < 4; j++)
            {
                  cout << "\t" << matrix_coeficient[i][j];
                  coeficient_matrix[i][j] = matrix_coeficient[i][j];
            }
            cout << endl;
      }


      ////// interpolation algorithm block end //////

      //deliting block

      for (int i = 0; i < space; i++)
      {
            delete matrix_coeficient[i];
      }

      for (int i = 0; i < space; i++)
      {
            delete matrix[i];
      }

      delete[] matrix_coeficient;
      delete[] matrix;
      delete[] right_vector;
      delete[] h;
      delete[] d;
      delete[] c;
      delete[] b;
      delete[] a;

}

type         result_of_splain_interpolation(type**         matrix_coeficient,
type*vector_value, type variable,int size)
{

      int i = 0;

      while (vector_value[i] < variable&&i<size-1) //old vatiable -1
      {
            i++;
      }

      return matrix_coeficient[i][0] + matrix_coeficient[i][1] * (variable -
vector_value[i]) + matrix_coeficient[i][2] * powf((variable - vector_value[i]),
2) + matrix_coeficient[i][3] * pow((variable - vector_value[i]), 3);
}

// main function

int main()
{
```

```
      //data block start

      type start = 0;
      type end = 4;

      type value;
      type interval = end - start;

      int number_of_nodes;

      //data block end

      //program start

      line

      cout << endl << "Program start..." << endl;

      line

      cout << endl << "Input number of nodes:" << endl;
      cin >> number_of_nodes;

      type* vector_value = new type[number_of_nodes];
      type*vector_result = new type[number_of_nodes];

      line

      cout << endl << "Find table of values and result of function on curent
values:" << endl;

      data_table(vector_value, vector_result, interval, number_of_nodes );

      cout << endl << "value"<<"\t result function" << endl;

      output_paralel_vector(vector_value, vector_result, number_of_nodes);

      line

      cout << endl << "Lagrange interpolation:" << endl;

      cout << endl << "-image of interpolation polynom:" << endl;

      string  temp  =  image_of_polynom_lagrange(vector_value,  vector_result,
number_of_nodes);

      temp[0] = ' ';
      temp[temp.length() - 1] = ' ';

      cout<<endl<<"Function(x)="<<temp<<endl;

      cout  <<  endl  <<  "-input  value,  where  you  want  to  get  result  of
interpolation:" << endl;
      cin >> value;
      cout << endl << "-result of interpolation:" << lagrange(vector_value,
vector_result, number_of_nodes, value) << endl;

      type error_lagrange = 0.0;

      {
      fstream fout;          fout.open("graph.xls", ios::out);

            fout  <<  "\t"  <<  "Value"<<"\t"  <<"Error"<<"\t"<<"Interpolation
result"<<"\t"<<"Function result"<< endl;
```

```
	for (type i = 0; i <= 4.0; i += 0.08)
	{

		fout << "\t" << i << "\t" << abs(lagrange(vector_value, vector_result, number_of_nodes, i) - function(i)) << "\t" << lagrange(vector_value, vector_result, number_of_nodes, i) << "\t" << function(i) << endl;

		if (error_lagrange > abs(lagrange(vector_value, vector_result, number_of_nodes, i) - function(i)))
		{
			continue;
		}
		if (error_lagrange < abs(lagrange(vector_value, vector_result, number_of_nodes, i) - function(i)))
		{
			error_lagrange = abs(lagrange(vector_value, vector_result, number_of_nodes, i) - function(i));

		}
	}


	fout.close();

	}

	cout << endl << "-error of interpolation (maximum value of difference root):" << error_lagrange<< endl;

	line

	type** matrix_coeficient = new type*[number_of_nodes-1];

	for (int i = 0; i < number_of_nodes-1; i++)
	{
		matrix_coeficient[i] = new type[4];
	}

	cout << endl << "Splain interpolation:" << endl;

	coefitient_splain( vector_result, vector_value, matrix_coeficient, number_of_nodes);

	{
		fstream fout;		fout.open("graph_addition.xls", ios::out);

		/*fout << "\t" << "Value"  << "\t" << "Interpolation result" << "\t" << "Function result"<<"\t"<<"Splain inerpolation result"<< endl;

		for (type i = 0; i <4.0; i += 0.08)
		{

			fout << "\t" << i << "\t" << lagrange(vector_value, vector_result, number_of_nodes, i) << "\t" << function(i)<< "\t"<<result_of_splain_interpolation(matrix_coeficient, vector_value, i, number_of_nodes-1) << endl;


		}*/
```

```
            fout << "\t" << "Value" << "\t" << "Error Lagrange
interpolation"<<"\t"<< "Error Spline interpolation" << "\t" << "Interpolation
result" << "\t" << "Function result" << endl;

            for (type i = 0; i <= 4.0; i += 0.08)
            {

                  fout << "\t" << i << "\t" << abs(lagrange(vector_value,
vector_result,     number_of_nodes,     i)     -     function(i))     <<"\t"<<
abs(result_of_splain_interpolation(matrix_coeficient,     vector_value,     i,
number_of_nodes     -     1)     -     function(i))     <<     "\t"     <<
result_of_splain_interpolation(matrix_coeficient,       vector_value,       i,
number_of_nodes - 1) << "\t" << function(i) << endl;



            }



            fout.close();

      }

      for (int i = 0; i < number_of_nodes - 1; i++)
      {
            delete matrix_coeficient[i];
      }

      delete[] matrix_coeficient;

      delete[] vector_result;
      delete[] vector_value;

      line
      cout << endl << "Program end..." << endl;
      line

      //program end


      return 0;
}
```