Звіт з предмету Чисельні методи
Пустовіта Дмитра
КА-32

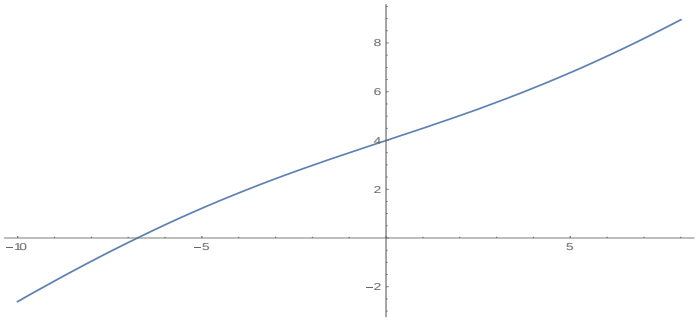
1. **Умова Задачі (Варіант 16)**

   Побудувати інтерполяційні поліноми Лагранжа, Нютона (вперед та назад) та реалізувати інтерполяцію кубічним сплайном для функції:

$$y = x + \frac{8}{1 + e^{\frac{x}{4}}}$$

2. **Математичне розв'язання задачі**

   Графік даної функції на проміжку [-10 ; 8]:



Аналітичний вигляд шостої похідної:

$$\frac{\partial^6 f}{\partial x^6} = 8\left(\frac{45e^{3x/2}}{256(1+e^{x/4})^7} - \frac{225e^{5x/4}}{512(1+e^{x/4})^6} + \frac{195e^{x}}{512(1+e^{x/4})^5} - \frac{135e^{3x/4}}{1024(1+e^{x/4})^4} + \frac{31e^{x/2}}{2048(1+e^{x/4})^3} - \frac{e^{x/4}}{4096(1+e^{x/4})^2}\right)$$

Вважатимемо що ми маємо значення функції в точках -10; -5; -3;-1;1;3;5;8}.

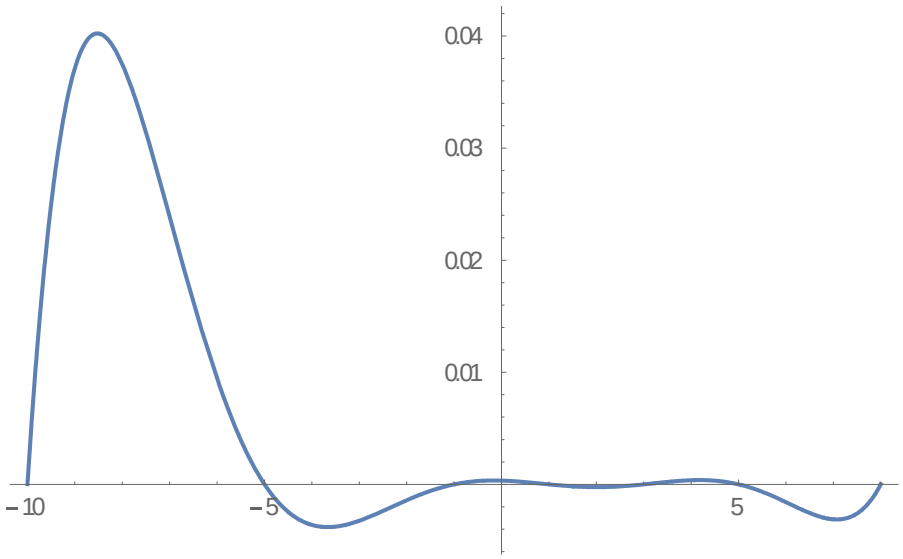
Інтерполяційний поліном у формі Лагранжа визначається наступним чином:

$$L_n(x) = \sum_{k=0}^{n} \frac{\prod_{j\neq k}(x - x_j)}{\prod_{j\neq k}(x_k - x_j)} f(x_k)$$

Для вузлів {-10; -5; -3;-1;1;3;5;8} отримали такий поліном:

$$L_6(x) = 3.9996626165 + 0.5001243803x + 0.000353172x^2 + 0.0024726034x^3 - 0.0000158802x^4 - 0.0000089908x^5 + 9.17 \times 10^{-8}x^6$$

При цьому маємо таку похибку ($err(x) = f(x) - L_6(x)$):



Поліном у формі Ньютона обчислюється так:

$P_n(x) = (x - x_0)f(x_0, x_1) + (x - x_0)(x - x_1)f(x_0, x_1, x_2) + \cdots + (x - x_0)(x - x_1) \dots (x - x_{n-1})f(x_0, x_1, \dots, x_n)$, де $f(x_0, x_1, \dots, x_n)$ поділені різниці. Їх можна отримати наступним чином:

де $f\left(x_j, x_{j+1}, \dots, x_k\right) = \sum_{i=j}^{k} \frac{f(x_i)}{\prod_{t \neq i}(x_i - x_t)}$

Відповідно якщо переставити вузли $\{x_0, x_1, \dots, x_n\}$ то можна отримати формулу інтерполювання назад:

$P_n(x) = (x - x_n)f(x_n, x_{n-1}) + (x - x_n)(x - x_{n-1})f(x_n, x_{n-1}, x_{n-2}) + \cdots + (x - x_n)(x - x_{n-1}) \dots (x - x_1)f(x_n, x_{n-1}, \dots, x_0)$,

Можна довести що інтерполяційні поліноми у формі Лагранжа , Ньютона при інтерполюванні вперед та Ньютона при інтерполюванні назад є одним і тим самим поліномом.

При інтерполяції кубічним сплайном знайдемо інтерполяційний поліном 3 степеня для кожного з відрізків $[x_{i-1}; x_i]$, де $i = \overline{1, n}$.

Для зручності запишемо інтерполяційний поліном у вигляді:

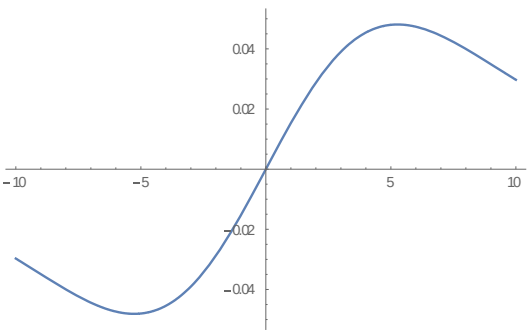
$$S_i(x) = a_i + b_i(x - x_i) + \frac{c_i}{2}(x - x_i)^2 + \frac{d_i}{6}(x - x_i)^3$$

Накладемо на S такі умови:

- Неперервність першої та другої похідної
- $S(x_i) = f(x_i)$
- Граничні умови: $S''(x_0) = f''(x_0)$ та $S''(x_n) = f''(x_n)$

Задамо граничні умови для сплайну на кінцях відрізку:

$f''(x) = 8(\frac{e^{x/2}}{8(1+e^{x/4})^3} - \frac{e^{x/4}}{16(1+e^{x/4})^2})$, тобто отримали $f''(-10) = -0.02973391792,$ та



$f''(8) \approx 0.03998125053$, до слова графік $f''(x)$ схематично виглядає наступним чином:
З цих умов отримуємо наступні формули для обчислення коефіцієнтів:

$$a_i = f(x_i)$$

$$h_i c_{i-1} + 2(h_i + h_{i+1})c_i + h_{i+1}c_{i+1} = 6\left(\frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i}\right)$$

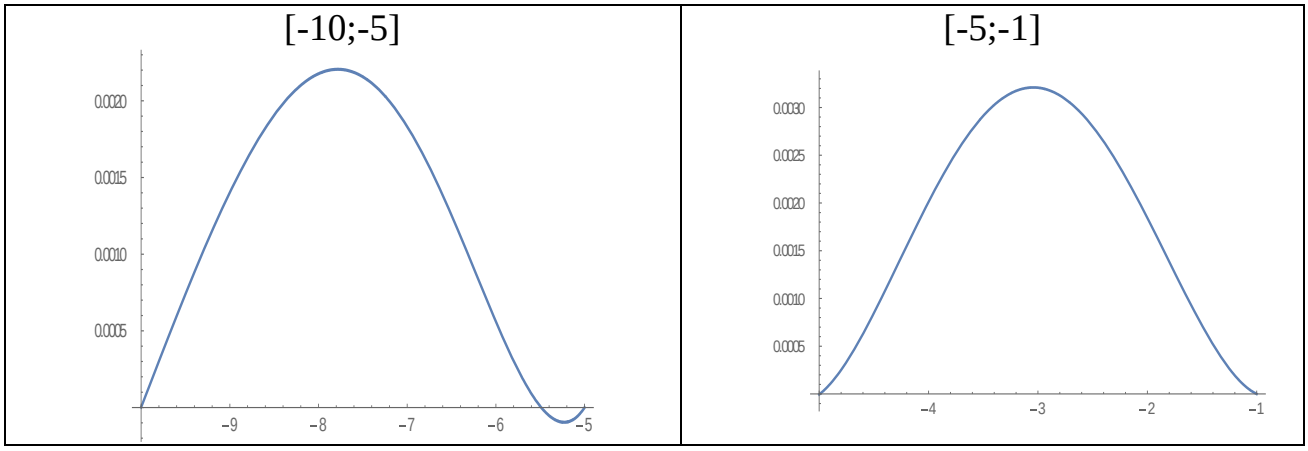
$$d_i = \frac{c_i - c_{i-1}}{h_i}$$

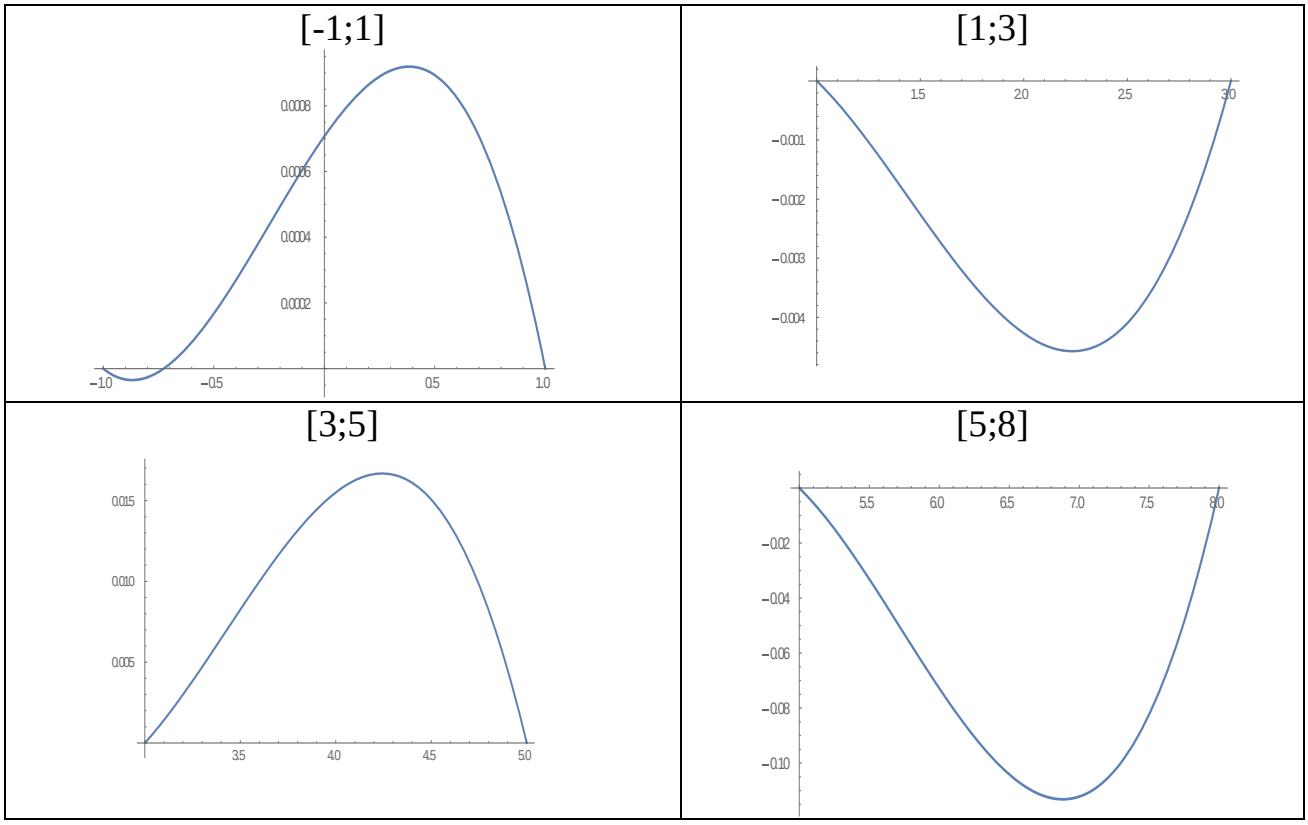
$$b_i = \frac{1}{2}h_i c_i - \frac{1}{6}h_i^2 d_i + \frac{f_i - f_{i-1}}{h_i} = \frac{f_i - f_{i-1}}{h_i} + \frac{h_i(2c_i + c_{i-1})}{6}$$

Врахувавши що $c_0 = f"(x_0)$, а $c_n = f"(x_n)$ коефіцієнти с можна знайти методом прогонки з початковими умовами: $\begin{cases} \alpha_1 = f"(x_0) \\ \beta_1 = 0 \end{cases}$

Для даної функції отримали такі сплайни та похибки:

Проміжок	**Кубічний сплайн**
[-10;-5]	$3.73319066 + 0.33410249x - 0.037552x^2 - 0.000756166x^3$
[-5;-1]	$3.9970965 + 0.492446x - 0.0058835x^2 + 0.0013549x^3$
[-1;1]	$3.999293 + 0.499035999x + 0.000706999x^2 + 0.003552x^3$
[1;3]	$4.0033045 + 0.4869995x + 0.0127435 - 0.0004601666x^3$
[3;5]	$3.7257710 + 0.764536x - 0.07976899x^2 + 0.009819x^3$
[5;8]	$7.47828166 - 1.48698099x + 0.37053549x^2 - 0.020201333x^3$





3. **Результати роботи** (метод Лагранжа)

```
Polinom is:
3.9996626165 + (0.5001243803) * x^1 + (0.0003531720) * x^2 + (0.0024726034) * x^3 + (-
0.0000158802) * x^4 + (-0.0000089908) * x^5 + (0.0000000917) * x^6
err(-10.0000) = 0.0000          err(-4.2500) = 0.0032          err(1.5000) = 0.0002
err(-9.7500) = 0.0144           err(-4.0000) = 0.0036          err(1.7500) = 0.0002
err(-9.5000) = 0.0250           err(-3.7500) = 0.0038          err(2.0000) = 0.0002
err(-9.2500) = 0.0324           err(-3.5000) = 0.0038          err(2.2500) = 0.0002
err(-9.0000) = 0.0371           err(-3.2500) = 0.0036          err(2.5000) = 0.0002
err(-8.7500) = 0.0396           err(-3.0000) = 0.0032          err(2.7500) = 0.0001
err(-8.5000) = 0.0402           err(-2.7500) = 0.0028          err(3.0000) = 0.0000
err(-8.2500) = 0.0394           err(-2.5000) = 0.0024          err(3.2500) = 0.0001
err(-8.0000) = 0.0375           err(-2.2500) = 0.0019          err(3.5000) = 0.0002
err(-7.7500) = 0.0347           err(-2.0000) = 0.0014          err(3.7500) = 0.0003
err(-7.5000) = 0.0314           err(-1.7500) = 0.0010          err(4.0000) = 0.0004
err(-7.2500) = 0.0277           err(-1.5000) = 0.0006          err(4.2500) = 0.0004
err(-7.0000) = 0.0239           err(-1.2500) = 0.0003          err(4.5000) = 0.0003
err(-6.7500) = 0.0200           err(-1.0000) = 0.0000          err(4.7500) = 0.0002
err(-6.5000) = 0.0162           err(-0.7500) = 0.0002          err(5.0000) = 0.0000
err(-6.2500) = 0.0127           err(-0.5000) = 0.0003          err(5.2500) = 0.0003
err(-6.0000) = 0.0094           err(-0.2500) = 0.0003          err(5.5000) = 0.0007
err(-5.7500) = 0.0065           err(0.0000) = 0.0003           err(5.7500) = 0.0011
err(-5.5000) = 0.0040           err(0.2500) = 0.0003           err(6.0000) = 0.0016
err(-5.2500) = 0.0018           err(0.5000) = 0.0002           err(6.2500) = 0.0021
err(-5.0000) = 0.0000           err(0.7500) = 0.0001           err(6.5000) = 0.0026
err(-4.7500) = 0.0014           err(1.0000) = 0.0000           err(6.7500) = 0.0029
err(-4.5000) = 0.0025           err(1.2500) = 0.0001           err(7.0000) = 0.0031
```

err(7.2500) = 0.0031 err(7.7500) = 0.0016
err(7.5000) = 0.0026 **err(8.0000) = 0.0000**

(Метод Нютона вперед)

Polinom is:
3.9996626165 + (0.5001243803) * x^1 + (0.0003531720) * x^2 + (0.0024726034) * x^3 + (-0.0000158802) * x^4 + (-0.0000089908) * x^5 + (0.0000000917) * x^6

err(-10.0000) = 0.0000	err(-3.7500) = 0.0038	err(2.5000) = 0.0002
err(-9.7500) = 0.0144	err(-3.5000) = 0.0038	err(2.7500) = 0.0001
err(-9.5000) = 0.0250	err(-3.2500) = 0.0036	**err(3.0000) = 0.0000**
err(-9.2500) = 0.0324	err(-3.0000) = 0.0032	err(3.2500) = 0.0001
err(-9.0000) = 0.0371	err(-2.7500) = 0.0028	err(3.5000) = 0.0002
err(-8.7500) = 0.0396	err(-2.5000) = 0.0024	err(3.7500) = 0.0003
err(-8.5000) = 0.0402	err(-2.2500) = 0.0019	err(4.0000) = 0.0004
err(-8.2500) = 0.0394	err(-2.0000) = 0.0014	err(4.2500) = 0.0004
err(-8.0000) = 0.0375	err(-1.7500) = 0.0010	err(4.5000) = 0.0003
err(-7.7500) = 0.0347	err(-1.5000) = 0.0006	err(4.7500) = 0.0002
err(-7.5000) = 0.0314	err(-1.2500) = 0.0003	**err(5.0000) = 0.0000**
err(-7.2500) = 0.0277	**err(-1.0000) = 0.0000**	err(5.2500) = 0.0003
err(-7.0000) = 0.0239	err(-0.7500) = 0.0002	err(5.5000) = 0.0007
err(-6.7500) = 0.0200	err(-0.5000) = 0.0003	err(5.7500) = 0.0011
err(-6.5000) = 0.0162	err(-0.2500) = 0.0003	err(6.0000) = 0.0016
err(-6.2500) = 0.0127	err(0.0000) = 0.0003	err(6.2500) = 0.0021
err(-6.0000) = 0.0094	err(0.2500) = 0.0003	err(6.5000) = 0.0026
err(-5.7500) = 0.0065	err(0.5000) = 0.0002	err(6.7500) = 0.0029
err(-5.5000) = 0.0040	err(0.7500) = 0.0001	err(7.0000) = 0.0031
err(-5.2500) = 0.0018	**err(1.0000) = 0.0000**	err(7.2500) = 0.0031
err(-5.0000) = 0.0000	err(1.2500) = 0.0001	err(7.5000) = 0.0026
err(-4.7500) = 0.0014	err(1.5000) = 0.0002	err(7.7500) = 0.0016
err(-4.5000) = 0.0025	err(1.7500) = 0.0002	**err(8.0000) = 0.0000**
err(-4.2500) = 0.0032	err(2.0000) = 0.0002	
err(-4.0000) = 0.0036	err(2.2500) = 0.0002	

(Метод Нютона назад)

Polinom is:
3.9996626165 + (0.5001243803) * x^1 + (0.0003531720) * x^2 + (0.0024726034) * x^3 + (-0.0000158802) * x^4 + (-0.0000089908) * x^5 + (0.0000000917) * x^6

err(-10.0000) = 0.0000	err(-6.0000) = 0.0094	err(-2.0000) = 0.0014
err(-9.7500) = 0.0144	err(-5.7500) = 0.0065	err(-1.7500) = 0.0010
err(-9.5000) = 0.0250	err(-5.5000) = 0.0040	err(-1.5000) = 0.0006
err(-9.2500) = 0.0324	err(-5.2500) = 0.0018	err(-1.2500) = 0.0003
err(-9.0000) = 0.0371	**err(-5.0000) = 0.0000**	**err(-1.0000) = 0.0000**
err(-8.7500) = 0.0396	err(-4.7500) = 0.0014	err(-0.7500) = 0.0002
err(-8.5000) = 0.0402	err(-4.5000) = 0.0025	err(-0.5000) = 0.0003
err(-8.2500) = 0.0394	err(-4.2500) = 0.0032	err(-0.2500) = 0.0003
err(-8.0000) = 0.0375	err(-4.0000) = 0.0036	err(0.0000) = 0.0003
err(-7.7500) = 0.0347	err(-3.7500) = 0.0038	err(0.2500) = 0.0003
err(-7.5000) = 0.0314	err(-3.5000) = 0.0038	err(0.5000) = 0.0002
err(-7.2500) = 0.0277	err(-3.2500) = 0.0036	err(0.7500) = 0.0001
err(-7.0000) = 0.0239	err(-3.0000) = 0.0032	**err(1.0000) = 0.0000**
err(-6.7500) = 0.0200	err(-2.7500) = 0.0028	err(1.2500) = 0.0001
err(-6.5000) = 0.0162	err(-2.5000) = 0.0024	err(1.5000) = 0.0002
err(-6.2500) = 0.0127	err(-2.2500) = 0.0019	err(1.7500) = 0.0002

```
err(2.0000) = 0.0002
err(2.2500) = 0.0002
err(2.5000) = 0.0002
err(2.7500) = 0.0001
err(3.0000) = 0.0000
err(3.2500) = 0.0001
err(3.5000) = 0.0002
err(3.7500) = 0.0003
err(4.0000) = 0.0004
err(4.2500) = 0.0004
err(4.5000) = 0.0003
err(4.7500) = 0.0002
err(5.0000) = 0.0000
err(5.2500) = 0.0003
err(5.5000) = 0.0007
err(5.7500) = 0.0011
err(6.0000) = 0.0016
err(6.2500) = 0.0021
err(6.5000) = 0.0026
err(6.7500) = 0.0029
err(7.0000) = 0.0031
err(7.2500) = 0.0031
err(7.5000) = 0.0026
err(7.7500) = 0.0016
err(8.0000) = 0.0000
```

(Кубічний сплайн)

4.Код програми

```
// Interpolation.cpp : Defines the entry point for the
console application.
//

#include "stdafx.h"
#include "fstream"
#include "math.h"
#include "iostream"
using namespace std;

#define c1 -0.02973391792
#define c2  0.03998125053
//#define c1 0
//#define c2 0

double func(double x){
        return x + 8 / (1 + exp(x / 4));
}

double poll(int n, double *a, double x){
        double s=0;
        for (int i = 0; i < n+1; i++)
                s += a[n-i]*pow(x, i);
        return s;
};
void printpoll(ofstream &fout, int n, double* a){
        fout << "Polinom is:" << endl << a[n];
        for (int i = 1; i < n + 1; i++)
                fout << " + (" << a[n - i] << ") * x^"
<< i;
        fout << endl;
}
void insertSort(double *a, int n){
        int i, j, k;
        double hold;
        for (i = 1; i < n; i++){
                hold = *(a + i);
                j = 0;
                while (hold > *(a + j)) j++;
                if (j < i) {
                        for (k = i; k > j; k--)
                                *(a + k) = *(a + k -
1);
                        *(a + j) = hold;
                }
        }
}
void binsertSort(double *a, int n){
        int i, j, k;
        double hold;
        for (i = 1; i < n; i++){
                hold = *(a + i);
                j = 0;
                while (hold < *(a + j)) j++;
                if (j < i) {
                        for (k = i; k > j; k--)
                                *(a + k) = *(a + k -
1);
                        *(a + j) = hold;
                }
        }
}
void createTable(int n, double* x, double* y, double
f(double)){
        ofstream fout("table.txt");
        int i;
        for (i = 0; i < n; i++){
                *(y + i) = f(*(x + i));
                fout << "X[" << i+1 << "]= " << *(x +
i) << "\tY[" << i+1 << "]= " << *(y + i) << endl;
        }
        fout.close();
}
double rizNewton(int min, int max, double *x, double
*y){
        int i, j;
        double s = 0, p=1;
        for (i = min; i <= max; i++){
                for (j = min; j <= max; j++){
                        if (i != j) p *= (*(x + i) -
*(x + j));
                }
                s += *(y + i) / p;
                p = 1;
        }
        return s;
}

/*
double rizNewton2(int min, int max, double *x, double
*y,int n){//так можна обчислювати в нютоні назад
        int i, j, ti, tj;
        double s = 0, p = 1;
        for (i = n-max; i <= n - min; i++){//але
заміна i=n-i перетворює в rizNewton
                ti = n-i;
                for (j = n - max; j <= n - min; j++){
                        tj = n - j;
                        if (i != j) p *= (*(x + ti) -
*(x + tj));
                }
                s += *(y + ti) / p;
                p = 1;
        }
        return s;
}
```

```
double NewtonUP(int n, double *x, double *y, double
X){
        double s = *y, p = 1;
        int i, j;
        for (i = 1; i < n; i++){
                for (j = 0; j < i; j++) p *= (X - *(x
+ j));
                s += p*rizNewton(0,i,x,y);
                p = 1;
        }
        return s;
}
double NewtonDown(int n, double *x, double *y, double
X){
        double s = *(y+n-1), p = 1;
        int i, j;
        for (i = n-2; i>=0; i--){
                for (j = n - 1; j > i; j--)  p *= (X -
*(x + j));
                s += p*rizNewton(i, n-1, x, y);
                p = 1;
        }
        return s;
}*/

void roots(int n, double *a, double *b)
{
        double *c = new double[n];
        for (int i = 1; i<n + 1; i++) b[i] = 0;
        b[0] = a[0];
        int k = 1;
        for (int i = 1; i<n; i++)
        {
                for (int j = 0; j<n; j++)
                        c[j] = 0;
                c[0] = a[i];
                for (int j = 0; j<k; j++)
                        c[j + 1] = b[j] * a[i];
                k++;
                for (int j = 0; j<k; j++)
                        b[j] += c[j];
        }
        for (int i = 0; i<n; i++)
                if (i % 2 == 0) c[i] = -b[i]; else
c[i] = b[i];
        b[0] = 1;
        for (int i = 0; i<n; i++)
                b[i + 1] = c[i];
}
void Lagrange(int n, double *x, double *y, double *c)
{
        double *a = new double[n], *b = new double[n +
1];
        for (int i = 0; i<n + 1; i++) c[i] = 0;
        for (int i = 0; i<n + 1; i++)
        {
                double p=1;
                int k = 0;
                for (int j = 0; j<n + 1; j++)
                        if (i != j) {
                        a[k] = x[j];
                        k++;
                        p*=(x[i] - x[j]); }
                roots(n, a, b);
                for (int j = 0; j < n + 1; j++)
                        c[j] += b[j]*y[i]/p;
        }
}
void NewtonUP(int n, double *x, double *y, double *a)
{
        double *b = new double[n + 1], *c = new
double[n + 1];
        for (int i = 0; i < n; i++) a[i] = 0;
        a[n] = y[0];
        int k = 0;
        for (int i = 1; i < n + 1; i++)
        {
                b[k] = x[i - 1];
                k++;
                roots(k, b, c);
                //cout << i<< " riz = " <<
rizNewton(0, i, x, y) << endl;
                // формуємо ans
                for (int j = n - i; j < n + 1; j++){
                        a[j] += c[j - n + i] *
rizNewton(0, i, x, y);
                }

        }//for (int j = 0; j < n; j++) cout << " ans["
<< j << "] = " << ans[j];
}
void NewtonBack(int n, double *x, double *y, double
*a)
{
        double *b = new double[n + 1], *c = new
double[n + 1];
        for (int i = 0; i<n; i++) a[i] = 0;
        a[n] = y[n];
        int k = 0;
        for (int i = 1; i<n + 1; i++)
        {
                b[k] = x[n-i+1];
                k++;
                roots(k, b, c);
                //cout << i << " riz = " <<
rizNewton(n - i,n , x, y) << endl;
                for (int j = n - i; j < n + 1; j++){

                a[j] += c[j - n + i] * rizNewton(n-i,
n, x, y);
                }
        }
}
void tochnist(string s, double start, double h,double
fin, double f(double), double*c, double n){
        ofstream fout(s);
        fout.setf(ios::fixed);
        fout.precision(10);
        printpoll(fout, n, c);
        fout.precision(4);
        int i;
        double t=start;
        for (i = 0; t<fin; i++){
                t = start + i*h;
                fout << "err(" << t << ") = " <<
fabs(f(t)-poll(n,c,t)) << endl;
        };
        fout.close();
}
void findCofsSplain3(int n, double *x, double *y,
double*a, double* b, double* c, double* d){
        for (int i = 0; i < n + 1; i++) a[i] = y[i];
        c[0] = c1;
        double *alfa = new double[n+1];
        double *beta = new double[n+1];
        //граничні умови на прогоночні коефіцієнти
        alfa[1] = 0;
        beta[1] = c1;
        for (int i = 1; i<n; i++)
        {       //A[i]c[i-1]+C[i]c[i]+B[i]c[i+1]=F[i]
                double A, B, C, F;
                A = x[i] - x[i - 1];
                B = x[i + 1] - x[i];
                C = 2 * (A + B);
```

```
                F = 6 * ((y[i + 1] - y[i]) / B - (y[i]
- y[i - 1]) / A);
                //обчислення прогоночних коефіцієнтів
                alfa[i + 1] = (-B) / (A*alfa[i] + C);
                beta[i + 1] = (F - A*beta[i]) /
(A*alfa[i] + C);
        }
        c[n] = c2;
        for (int i = n - 1; i>0; i--)
                c[i] = alfa[i + 1] *c[i + 1]+ beta[i +
1];
        delete alfa;
        delete beta;
        for (int i = 1; i<n + 1; i++)
        {
                d[i] = (c[i] - c[i-1]) / (x[i] - x[i -
1]);
                b[i] = (y[i] - y[i - 1]) / (x[i] - x[i
- 1]) + (x[i] - x[i - 1])*(2 *c[i] + c[i-1]) / 6;
        }
}
double Splain(int n, double *x, double*a, double* b,
double* c, double* d, double X){
        for (int i = 1; i < n; i++){
                if ((X <= *(x + i)) && (X >= *(x + i -
1)))
                        return *(a + i) + *(b + i)*(X
- *(x + i)) + *(c + i)*pow(X - *(x + i), 2) / 2 + *(d
+ i)*pow(X - *(x + i), 3) / 6;
        }
}


void printSplain(ofstream &fout, int n, double *x,
double*a, double* b, double* c, double* d, double
f(double)){
        double h = 0.25, e, t=x[0];
        fout.setf(ios::fixed);
        int i;
        for (i = 1; i < n; i++){
                fout << "на проміжку [" << x[i - 1] <<
" ; " << x[i] << "] кубічний сплайн має
вигляд:"<<endl;
                fout << a[i] << " + (" << b[i] << ")(X
- (" << x[i] << ")) + (" << c[i] << ")(X - (" << x[i]
<< "))^2/2 + (" << d[i] << ")(X - (" << x[i] <<
"))^3/6" << endl;
        }
        for (i = 0; t<x[n-1]; i++){
                t = *x + i*h;
                e = fabs(f(t) -Splain(n, x, a, b, c,
d, t));
                fout << "err(" << t << ")= " << e
<<endl;
        };
}
int _tmain(int argc, _TCHAR* argv[])
{
        int i,n;
        ifstream fin("in.txt");
        ofstream fout("Splain.txt");
        fin >> n;
        double *x = new double[n+1], *y = new
double[n+1],*a = new double[n+1],h = 0.25;
        for (i = 0; i < n+1; i++) fin >> *(x + i);
        fin.close();
        binsertSort(x, n + 1);
        createTable(n + 1, x, y, func);
        NewtonBack(n, x, y, a);
        tochnist("NewtonBack.txt", x[n], h, x[0],
func, a,n);

        insertSort(x, n+1);
        createTable(n+1, x, y, func);
        Lagrange(n, x, y, a);
        tochnist("Lagrange.txt", x[0], h, x[n], func,
a,n);
        NewtonUP(n,x,y,a);
        tochnist("NewtonUp.txt", x[0], h,x[n], func,
a,n);
        double *a1 = new double[n + 1], *a2 = new
double[n + 1], *a3 = new double[n + 1], *a4 = new
double[n + 1];
        findCofsSplain3(n + 1, x, y, a1, a2, a3, a4);
        printSplain(fout, n + 1, x, a1, a2, a3, a4,
func);
        system("Pause");
        return 0;
}
```

5. Висновок

Порівнюючи похибки інтерполяції бачимо що похибка залежить від сітки та гладкості функції , зокрема на великих проміжках краще спрацює метод сплайну тоді як на густій рівномірній сітці метод Лагранжа.