Фролова Анастасия

Вычмат, 3 лаба

```
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

const int n = 6;

double Lagrange(double x, double xi[], double yi[])
    {

        double lag = 0;

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

            //вместо двух циклов один
            //в книжке в алгоритме отдельно числитель и знаменатель
            //здесь в s записываем всю дробь
            //потом прибавляем к результату lag (lagrange)

            for (int j = 0; j < n; ++j)
                if (j != i)
                    s *= (x - xi[j]) / (xi[i] - xi[j]);
            lag += yi[i] * s;
        }

        return lag;
    }

    void razn(double xi[], double yi[], double Y[][n])
    {
        for (int m = 1; m < n; m++)
            for (int k = 0; k < n - m; k++)
            {
                if (m == 1)
                {
                    Y[k][m] = (yi[k + 1] - yi[k]) / (xi[k + 1] - xi[k]);
                }
                else
                {
                    Y[k][m] = (Y[k + 1][m - 1] - Y[k][m - 1]) / (xi[k + m] - xi[k]);
```

```
44.                 }
45.             }
46.     }
47.
48.
49.     double NewtoneNF(double x, double xi[], double yi[], double
   Y[][n])
50.     {
51.         double nt = yi[0];
52.         double prod = 1.0;
53.
54.         for (int k = 1; k < n; k++)
55.         {
56.             prod = prod * (x - xi[k - 1]);
57.             nt = nt + (Y[0][k] * prod);
58.         }
59.
60.         return nt;
61.     }
62.
63.     double NewtoneNB(double x, double xi[], double yi[], double
   Y[][n])
64.     {
65.         double nt = yi[n - 1];
66.         double prod = 1.0;
67.
68.         for (int k = 1; k < n; k++)
69.         {
70.             prod = prod * (x - xi[n - k]);
71.             nt = nt + (Y[n - k - 1][k] * prod);
72.         }
73.
74.         return nt;
75.     }
76.     /*
77.     *
78.     * какой-то алгоритм из интернета, работает исправно, но в
   книжке по-другому))
79.     *
80.     double Newtone(double x, int n, double xi[], double yi[])
81.     {
82.
83.         double nt = yi[0];
84.
85.         double prod = 1;
86.
87.         for (int i = 1; i < n; ++i)
88.         {
89.
90.             double F = 0;
```

```
91.             for (int j = 0; j <= i; ++j)
92.             {
93.
94.                 for (int k = 0; k <= i; ++k)
95.                     if (k != j)
96.                         prod *= (xi[j] - xi[k]);
97.
98.                 F += yi[j] / prod;
99.             }
100.
101.
102.            for (int k = 0; k < i; ++k)
103.                F *= (x - xi[k]);
104.            nt += F;
105.        }
106.        return nt;
107.    }*/
108.
109.    int main()
110.    {
111.
112.        const int N = 6;
113.
114.        double xi[N], yi[N];
115.
116.        double x = -0.5;
117.
118.        double Y[n][n];
119.
120.        xi[0] = -1.4;
121.        xi[1] = -0.4;
122.        xi[2] = 0.2;
123.        xi[3] = 1.3;
124.        xi[4] = 2;
125.        xi[5] = 3.3;
126.
127.        yi[0] = 6.50496;
128.        yi[1] = -4.43904;
129.        yi[2] = 4.13952;
130.        yi[3] = 28.51563;
131.        yi[4] = 24;
132.        yi[5] = 6.76863;
133.
134.        cout << setw(12) << "Lagrange(x)" << Lagrange(x, xi, yi)
    << endl;
135.
136.        //cout << setw(12) << "Newton(x)" << Newtone(x, xi, yi);
137.
138.        razn(xi, yi, Y);
139.
```

```
140.         cout << setw(12) << "Newton NB (x)" << NewtoneNB(x, xi,
   yi, Y) << endl;
141.         cout << setw(12) << "Newton NF (x)" << NewtoneNF(x, xi,
   yi, Y) << endl;
142.         cout << endl;
143.
144.     }
```