Wydział WIMiIP	Imię i nazwisko: Zuzanna Będkowska	Rok: 2	Grupa: 1	Data: 01.05.2022
Metody Numeryczne	Temat: Aproksymacja wielomianami ortogonalnymi Grama dla równoodległych węzłów			

Zadanie: Wyznaczenie funkcji aproksymującej za pomocą wielomianów ortogonalnych Grama

Zgodnie z poleceniem napisano kod, który na podstawie wyznaczonych wielomianów ortogonalnych, aproksymuje funkcję i zwraca jej wartości w podanych punktach. Aby skonstruować funkcję aproksymującą $y(x)$ dla $n+1$ punktów, posłużono się wzorem:

$$y(x) = \sum_{k=0}^{m} b_k F_k(\frac{x - x_0}{h})$$

Gdzie:

- m - stopień wielomianu
- F_k - wielomian ortogonalny grama stopnia k
- x_0 - współrzędna x pierwszego podanego punktu
- h - odległość między punktami
- b_k - czynnik wyznaczany ze wzoru:

$$b_i = \frac{c_k}{s_k}$$

$$c_k = \sum_{q=0}^{n} y_q F_k(q)$$

$$s_k = \sum_{q=0}^{n} F_k(q)$$

Aby wyznaczyć wielomian F_k, użyto następującego wzoru:

$$F_k = \sum_{s=0}^{k} (-1)^s \binom{k}{s} \binom{k+s}{s} \frac{q^{[s]}}{n^{[s]}}$$

Zanim przystąpiono do implementacji wzorów, wprowadzono następujące zmienne i dodano przykładowe punkty z poprzedniego laboratorium:

```
68    int m = 2; //stopien wielomianu
69    int wezly = 8;
70    int n = wezly - 1;
71    vector <double> F(n + 1, 0);
72    vector <double> C(m + 1, 0);
73    vector <double> S(m + 1, 0);
74    vector <double> B(m + 1, 0);
75    vector <vector <double>> macierz(m + 1, C); //macierz
76    vector <pair <double, double>> punkty;
77    punkty.push_back(make_pair(1.0, 2.0));
78    punkty.push_back(make_pair(2.0, 4.0));
79    punkty.push_back(make_pair(3.0, 3.0));
80    punkty.push_back(make_pair(4.0, 5.0));
81    punkty.push_back(make_pair(5.0, 6.0));
82    punkty.push_back(make_pair(6.0, 9.0));
83    punkty.push_back(make_pair(7.0, 11.0));
84    punkty.push_back(make_pair(8.0, 11.0));
```

W celu wyznaczenia wartości wielomianu ortogonalnego użyto następujących funkcji w kodzie:

- Funkcja obliczająca silnie, potrzebna do wyznaczenia wartości symbolu Newtona - C++ nie posiada funkcji obliczającej silnię, więc zaimplementowano własną:

```
9   int factorial(int n)
10  {
11      int wynik = 1.0;
12      for (int i = 1; i <= n; ++i)
13      {
14          wynik *= i;
15      }
16      return wynik;
17  }
```

  Silnia obliczana jest wyłącznie dla liczb całkowitych, oraz jest liczbą całkowitą, w związku z tym wartość przyjmowana i zwracana jest typu int.

- Funkcja obliczająca wartość symbolu Newtona - do wyznaczenia tej wartości użyto napisanej wcześniej funkcji do obliczania silni. Zgodnie ze wzorem na symbol Newtona, przy założeniu, że $n > k$:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

  Zaimplementowano następującą funkcję obliczającą wartość symbolu Newtona:

```
19  int dwumian(int n, int k)
20  {
21      int wynik = 0;
22      wynik = factorial(n) / (factorial(k) * factorial(n - k));
23      return wynik;
24  }
```

- Funkcja obliczająca wartość wielomianu czynnikowego $x^{[s]}$ - zgodnie ze wzorem na wielomian czynnikowy, przy założeniu, że $x > s$:

$$x^{[s]} = x(x-1)(x-2)\dots(x-s+1)$$

Zaimplementowano funkcję obliczającą wartość tego wielomianu:

```
26  int x_s(int x, int s)
27  {
28      int wynik = 1;
29      for (int i = 0; i < s; ++i)
30      {
31          wynik *= (x - i);
32      }
33      return wynik;
34  }
```

Mając tak zdefiniowane elementy, zaimplementowano funkcję obliczającą wartość wielomianu ortogonalnego Grama F_k:

```
36  double wielomianOrtogonalny(int q, int n, int k)
37  {
38      double F = 0.0;
39      for (int s = 0; s <= k; ++s)
40      {
41          F += pow(-1, s) * dwumian(k, s) * dwumian(k + s, s) * x_s(q, s) / x_s(n, s);
42      }
43      return F;
44  }
```

Dzięki funkcji obliczającej wartości wielomianu ortogonalnego, możliwe było zaimplementowanie obliczeń potrzebnych do wyznaczenia c_k i s_k:

```
46  double s(int n, int k)
47  {
48      double wynik = 0.0;
49      for (int q = 0; q <= n; ++q)
50      {
51          wynik += pow(wielomianOrtogonalny(q, n, k), 2);
52      }
53      return wynik;
54  }
55
56  double c(vector <pair <double, double>>& punkty, int n, int k)
57  {
58      double wynik = 0.0;
59      for (int i = 0; i <= n; ++i)
60      {
61          wynik += punkty[i].second * wielomianOrtogonalny(i, n, k);
62      }
63      return wynik;
64  }
```

Następnie obliczono b_k:

```
98      for (int i = 0; i <= m; ++i)
99      {
100         C[i] = c(punkty, n, i);
101         S[i] = s(n, i);
102         B[i] = C[i] / S[i];
103     }
```

I wyznaczono wartości funkcji aproksymującej dla podanych punktów:

```
104     for (int i = 0; i <= n; ++i)
105     {
106         for (int k = 0; k <= m; ++k)
107         {
108             F[i] += B[k] * wielomianOrtogonalny(i, n, k); //k - stopien
109         }
110     }
```

Należy pamiętać, że zamiast x podstawiono $\frac{x - x_0}{h}$ co dla każdego punktu z zakresu daje liczby całkowite od 0 do n - aby nie zaśmiecać pamięci, wykorzystano licznik pętli.
Wyznaczono również macierz pomocniczą wartości określonych wielomianów Grama dla poszczególnych punktów:

```
91      for (int q = 0; q <= m; ++q)
92      {
93          for (int k = 0; k <= m; ++k)
94          {
95              macierz[q][k] = wielomianOrtogonalny(q, n, k);
96          }
97      }
```

Tutaj również zamiast x podstawiono $\frac{x - x_0}{h}$.

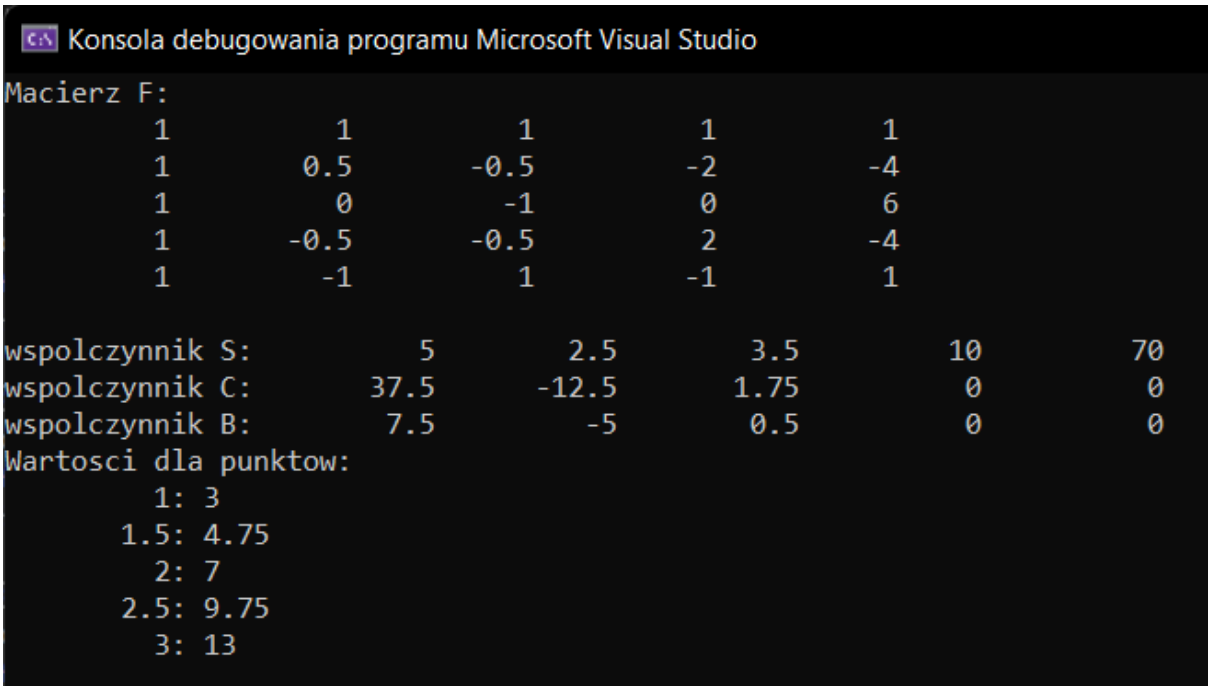
Cały kod:

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <iomanip>
#include <vector>

using namespace std;

int factorial(int n)
{
    int wynik = 1.0;
    for (int i = 1; i <= n; ++i)
    {
        wynik *= i;
    }
    return wynik;
}

int dwumian(int n, int k)
{
    int wynik = 0;
    wynik = factorial(n) / (factorial(k) * factorial(n - k));
    return wynik;
}

int x_s(int x, int s)
{
    int wynik = 1;
    for (int i = 0; i < s; ++i)
    {
        wynik *= (x - i);
    }
    return wynik;
}

double wielomianOrtogonalny(int q, int n, int k)
{
    double F = 0.0;
    for (int s = 0; s <= k; ++s)
    {
        F += pow(-1, s) * dwumian(k, s) * dwumian(k + s, s) * x_s(q, s) / x_s(n, s);
    }
    return F;
}

double s(int n, int k)
{
    double wynik = 0.0;
    for (int q = 0; q <= n; ++q)
    {
        wynik += pow(wielomianOrtogonalny(q, n, k), 2);
    }
    return wynik;
}

double c(vector <pair <double, double>>& punkty, int n, int k)
{
    double wynik = 0.0;
    for (int i = 0; i <= n; ++i)
    {
        wynik += punkty[i].second * wielomianOrtogonalny(i, n, k);
    }
    return wynik;
}

int main()
{
    int m = 2; //stopien wielomianu
    int wezly = 8;
    int n = wezly - 1;
    vector <double> F(n + 1, 0);
    vector <double> C(m + 1, 0);
    vector <double> S(m + 1, 0);
    vector <double> B(m + 1, 0);
```

```cpp
    vector <vector <double>> macierz(m + 1, C); //macierz
    vector <pair <double, double>> punkty;
    punkty.push_back(make_pair(1.0, 2.0));
    punkty.push_back(make_pair(2.0, 4.0));
    punkty.push_back(make_pair(3.0, 3.0));
    punkty.push_back(make_pair(4.0, 5.0));
    punkty.push_back(make_pair(5.0, 6.0));
    punkty.push_back(make_pair(6.0, 9.0));
    punkty.push_back(make_pair(7.0, 11.0));
    punkty.push_back(make_pair(8.0, 11.0));

    /*punkty.push_back(make_pair(1.0, 3.0));
    punkty.push_back(make_pair(1.5, 4.75));
    punkty.push_back(make_pair(2.0, 7.0));
    punkty.push_back(make_pair(2.5, 9.75));
    punkty.push_back(make_pair(3.0, 13.0));*/
    for (int q = 0; q <= m; ++q)
    {
        for (int k = 0; k <= m; ++k)
        {
            macierz[q][k] = wielomianOrtogonalny(q, n, k);
        }
    }
    for (int i = 0; i <= m; ++i)
    {
        C[i] = c(punkty, n, i);
        S[i] = s(n, i);
        B[i] = C[i] / S[i];
    }
    for (int i = 0; i <= n; ++i)
    {
        for (int k = 0; k <= m; ++k)
        {
            F[i] += B[k] * wielomianOrtogonalny(i, n, k); //k - stopien
        }
    }
    cout << "Macierz F:\n";
```

```cpp
    for (auto i : macierz)
    {
        for (auto j : i)
        {
            cout << setw(10) << right << j << " ";
        }
        cout << "\n";
    }
    cout << "\nwspolczynnik S: ";
    for (auto i : S)
    {
        cout << setw(10) << right << i << " ";
    }
    cout << "\nwspolczynnik C: ";
    for (auto i : C)
    {
        cout << setw(10) << right << i << " ";
    }
    cout << "\nwspolczynnik B: ";
    for (auto i : B)
    {
        cout << setw(10) << right << i << " ";
    }
    cout << "\nWartosci dla punktow:\n";
    for (int i = 0; i <= n; ++i)
    {
        cout << setw(10) << right << punkty[i].first << ": " << F[i] << "\n";
    }
}
```

Wyniki dla przykładu ze źródła:



Porównanie wyników ze źródłem:

Tabl.4

q	x_i	y_i	$F_0(q)$	$F_1(q)$	$F_2(q)$	$F_3(q)$	$F_4(q)$
0	1	3	1	1	1	1	1
1	1.5	4.75	1	0.5	-0.5	-2	-4
2	2	7	1	0	-1	0	6
3	2.5	9.75	1	-0.5	-0.5	2	-4
4	3	13	1	-1	1	-1	1

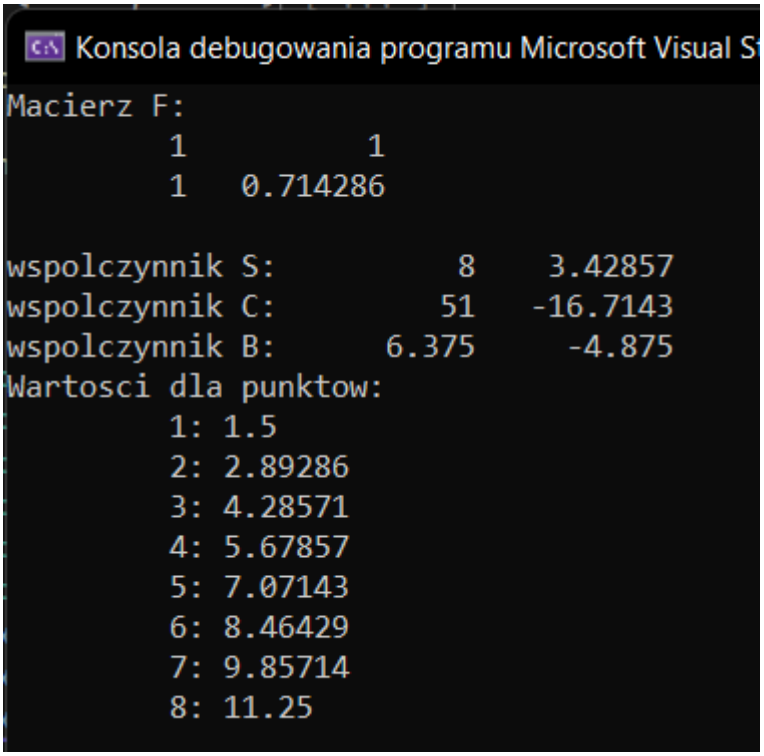
Źródło: [Fortuna Z., " Metody numeryczne ", W - wa, WNT 2001.]

Tabl.5

C_i	37.5	-12.5	1.75	0	0
S_i	5	2.5	3.5	10	70
b_i	7.5	-5	0.5	0	0

Źródło: [Fortuna Z. " Metody numeryczne ", W wa, WNT 2001.]

Wyniki dla przykładu z laboratorium i $m = 1$:



Wyniki dla przykładu z laboratorium i $m = 2$: