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

Zadanie: Wyznaczenie funkcji aproksymującej za pomocą wielomianów ortogonalnych Grama

Wzory:

Aby wyznaczyć wielomiany ortogonalne dla $n + 1$ węzłów, które nie są równoodległe, posłużono się zależnością rekurencyjną:

$$\varphi_{j+1}(x) = (x - \alpha_{j+1})\varphi_j - \beta_j\varphi_{j-1}$$

Przy założeniu, że:

$$\varphi_0 = 1,\ \varphi_{-1} = 0$$

Gdzie:

- φ_{j+1}, φ_j, φ_{j-1} - wielomiany ortogonalne na zbiorze punktów x_i
- α_{j+1} - współczynnik wyznaczany ze wzoru:

$$\alpha_{j+1} = \frac{\sum_{i=0}^{n} x_i\varphi_j^2(x_i)}{\sum_{i=0}^{n} \varphi_j^2(x_i)}$$

- β_j - współczynnik wyznaczany ze wzoru:

$$\beta_j = \frac{\sum_{i=0}^{n} x_i\varphi_j(x_i)\varphi_{j-1}(x_i)}{\sum_{i=0}^{n} \varphi_{j-1}^2(x_i)}$$

Przy tak zdefiniowanym wielomianie ortogonalnym, funkcja aproksymująca $y(x)$ ma postać:

$$y(x) = \sum_{k=0}^{m} b_k\varphi_k(x)$$

Gdzie:

- m - stopień wielomianu
- b_k - współczynnik obliczany ze wzoru:

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

$$c_k = \sum_{i=0}^{n} y_i\varphi_k(x_i)$$

$$s_k = \sum_{i=0}^{n} \varphi_k^2(x_i)$$

Implementacja:

- Oznaczenia w kodzie:

```
37      int m = 2; //stopien wielomianu
38      int wezly = 8;
39      int n = wezly - 1;
40      vector <double> F(n + 1, 0.0);
41      vector <double> Alfa(m + 1, 10000000);
42      vector <double> Beta(m + 1, 10000000);
43      vector <double> C(m + 1, 0);
44      vector <double> S(m + 1, 0);
45      vector <double> B(m + 1, 0);
46      vector <double> pomoc(m + 1, 0.0);
47      vector <vector <double>> macierz(n + 1, pomoc); //macierz
48      vector <pair <double, double>> punkty;
49      punkty.push_back(make_pair(1.0, 2.0));
50      punkty.push_back(make_pair(2.0, 4.0));
51      punkty.push_back(make_pair(3.0, 3.0));
52      punkty.push_back(make_pair(4.0, 5.0));
53      punkty.push_back(make_pair(5.0, 6.0));
54      punkty.push_back(make_pair(6.0, 9.0));
55      punkty.push_back(make_pair(7.0, 11.0));
56      punkty.push_back(make_pair(8.0, 11.0));
57
```

- Funkcja obliczająca wartości wielomianów ortogonalnych φ_{j+1}:

```
160 double fi(int j, double x, vector <pair <double, double>>& punkty, vector <double>& A, vector <double>& B)
161 {
162     double Fi = 0.0;
163     if (j == -1)
164     {
165         return 0;
166     }
167     else if (j == 0)
168     {
169         return 1;
170     }
171     else if (j == 1)
172     {
173         Fi = (x - alfa(j, punkty, A, B)) * fi(j - 1, x, punkty, A, B);
174         return Fi;
175     }
176     else
177     {
178         Fi = (x - alfa(j, punkty, A, B)) * fi(j - 1, x, punkty, A, B) - beta(j - 1, punkty, A, B) * fi(j - 2, x, punkty, A, B);
179         return Fi;
180     }
181 }
```

W celu zwiększenia wygody, zapis uproszczono stosując j zamiast $j + 1$. Analogicznie j przechodzi na $j - 1$, a $j - 1$ na $j - 2$. Zgodnie z założeniami, przy $j + 1 = 0$, funkcja zwraca 1, a przy $j + 1 = (-1)$, funkcja zwraca 0. Rozpatrzono również przypadek w którym $j + 1 = 1$, wtedy $j - 1 = -1$, co oznacza, że $\varphi_{j-1} = 0$ - element sumy zawierający φ_{j-1} jest równy 0, więc można go pominąć. Dzięki temu uproszczeniu uniknięto problemu z dzieleniem przez 0 przy obliczaniu β. W pozostałych przypadkach zastosowano wyżej podany wzór z uwzględnieniem przesunięcia w indeksie j.

- Funkcja wyznaczająca α_{j+1}:

```
124  double alfa(int j, vector <pair<double, double>>& punkty, vector <double> & A, vector <double>& B)
125  {
126      if (A[j] == 10000000)
127      {
128          double Alfa1 = 0.0;
129          double Alfa2 = 0.0;
130          double Alfa = 0.0;
131          for (auto i : punkty)
132          {
133              Alfa1 += i.first * pow(fi(j - 1, i.first, punkty, A, B), 2);
134              Alfa2 += pow(fi(j - 1, i.first, punkty, A, B), 2);
135          }
136          Alfa = Alfa1 / Alfa2;
137          A[j] = Alfa;
138      }
139      return A[j];
140  }
```

Zgodnie ze wzorem zapisanym powyżej, wyznaczono wartości α. Wiedząc, że α_{j+1}dla każdego punktu jest takie samo, obliczane wartości zapisano w odpowiedniej tablicy w celu zmniejszenia ilości wywołań funkcji φ (zamiast wielokrotnie obliczać to samo a_{j+1}dla każdego punktu, wykorzystano obliczona już wartość). Wartość obliczana jest wyłącznie gdy w tablicy znajduje się 10000000 - znacznik informujący o nie obliczeniu tej wartości.

- Funkcja wyznaczająca β_j:

```
142  double beta(int j, vector <pair <double, double>>& punkty, vector <double>& A, vector <double>& B)
143  {
144      if (B[j] == 10000000)
145      {
146          double Beta1 = 0.0;
147          double Beta2 = 0.0;
148          double Beta = 0.0;
149          for (auto i : punkty)
150          {
151              Beta1 += i.first * (fi(j, i.first, punkty, A, B) * fi(j - 1, i.first, punkty, A, B));
152              Beta2 += pow(fi(j - 1, i.first, punkty, A, B), 2);
153          }
154          Beta = Beta1 / Beta2;
155          B[j] = Beta;
156      }
157      return B[j];
158  }
```

Podobnie jak α_{j+1}, wartości β_j zostają zapisane w tablicy.

- Funkcje obliczające c_k i s_k:

```
15   double s(int n, int k, vector <pair <double, double>> & punkty, vector <double>& A, vector <double>& B)
16   {
17       double wynik = 0.0;
18       for (int i = 0; i <= n; ++i)
19       {
20           wynik += pow(fi(k, punkty[i].first, punkty, A, B), 2);
21       }
22       return wynik;
23   }
24
25   double c(vector <pair <double, double>>& punkty, int n, int k, vector <double>& A, vector <double>& B)
26   {
27       double wynik = 0.0;
28       for (int i = 0; i <= n; ++i)
29       {
30           wynik += punkty[i].second * fi(k, punkty[i].first, punkty, A, B);
31       }
32       return wynik;
33   }
```

- Obliczanie b_k:

```
70      for (int i = 0; i <= n; ++i)
71      {
72          C[i] = c(punkty, n, i, Alfa, Beta);
73          S[i] = s(n, i, punkty, Alfa, Beta);
74          B[i] = C[i] / S[i];
75      }
```

- Wyznaczenie macierzy pomocniczej:

```
63  for (int i = 0; i <= n; ++i)
64  {
65      for (int k = 0; k <= m; ++k)
66      {
67          macierz[i][k] = fi(k, punkty[i].first, punkty, Alfa, Beta);
68      }
69  }
```

- Obliczenie wartości funkcji aproksymującej w danych punktach:

```
76      for (int i = 0; i <= n; ++i)
77      {
78          for (int k = 0; k <= m; ++k)
79          {
80              F[i] += B[k] * macierz[i][k]; //k - stopien
81          }
82      }
```

Aby nie zwiększać ilości wywołań, użyto wartości z wyznaczonej wcześniej macierzy pomocniczej.

Cały kod:

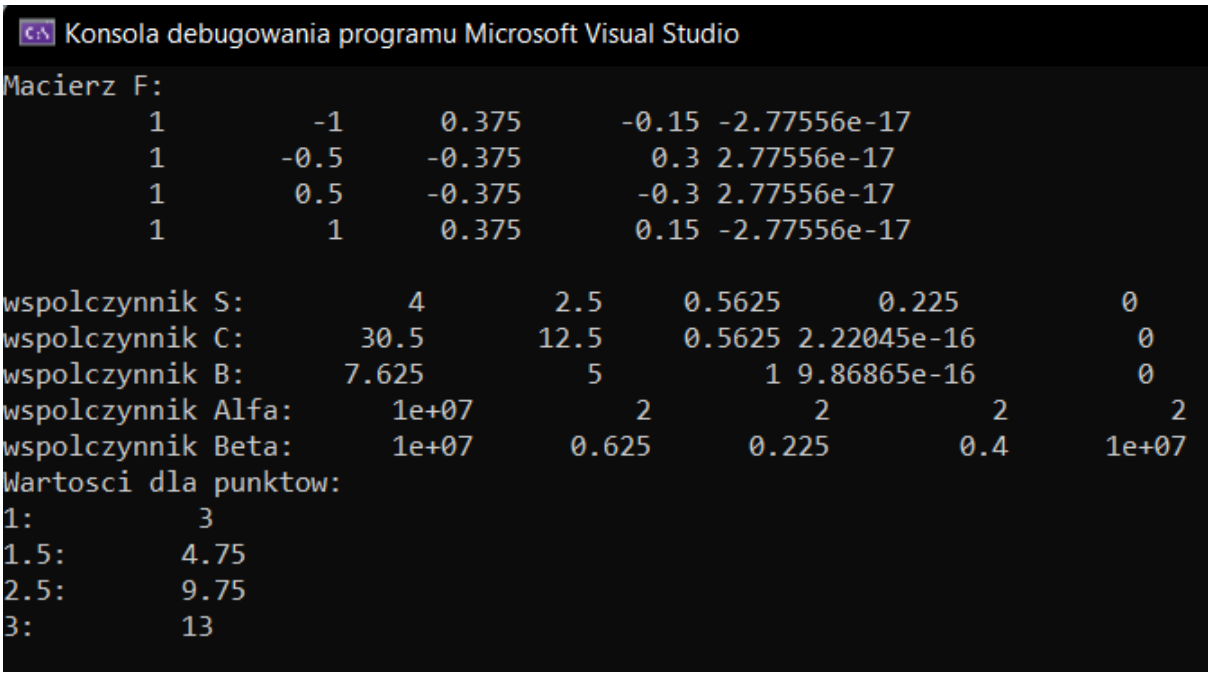
```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <iomanip>
#include <vector>

using namespace std;

double alfa(int j, vector <pair<double, double>>& punkty, vector <double>& A, vector <double>& B);

double beta(int j, vector <pair <double, double>>& punkty, vector <double>& A, vector <double>& B);

double fi(int j, double x, vector <pair <double, double>>& punkty, vector <double>& A, vector <double>& B);

double s(int n, int k, vector <pair <double, double>> & punkty, vector <double>& A, vector <double>& B)
{
    double wynik = 0.0;
    for (int i = 0; i <= n; ++i)
    {
        wynik += pow(fi(k, punkty[i].first, punkty, A, B), 2);
    }
    return wynik;
}

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

int main()
{
    int m = 1; //stopien wielomianu
    int wezly = 8;
    int n = wezly - 1;
    vector <double> F(n + 1, 0.0);
    vector <double> Alfa(m + 1, 10000000);
    vector <double> Beta(m + 1, 10000000);
    vector <double> C(m + 1, 0);
    vector <double> S(m + 1, 0);
    vector <double> B(m + 1, 0);
    vector <double> pomoc(m + 1, 0.0);
    vector <vector <double>> macierz(n + 1, pomoc); //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 i = 0; i <= n; ++i)
    {
        for (int k = 0; k <= m; ++k)
        {
            macierz[i][k] = fi(k, punkty[i].first, punkty, Alfa, Beta);
        }
    }
    for (int i = 0; i <= m; ++i)
    {
        C[i] = c(punkty, n, i, Alfa, Beta);
        S[i] = s(n, i, punkty, Alfa, Beta);
        B[i] = C[i] / S[i];
```

```
    }
    for (int i = 0; i <= n; ++i)
    {
        for (int k = 0; k <= m; ++k)
        {
            F[i] += B[k] * macierz[i][k]; //k - stopien
        }
    }
    cout << "Macierz F:\n";
    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 << "\nwspolczynnik Alfa: ";
    for (auto i : Alfa)
    {
        cout << setw(10) << right << i << " ";
    }
```

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

double alfa(int j, vector <pair<double, double>>& punkty, vector <double> & A, vector <double>& B)
{
    if (A[j] == 10000000)
    {
        double Alfa1 = 0.0;
        double Alfa2 = 0.0;
        double Alfa = 0.0;
        for (auto i : punkty)
        {
            Alfa1 += i.first * pow(fi(j - 1, i.first, punkty, A, B), 2);
            Alfa2 += pow(fi(j - 1, i.first, punkty, A, B), 2);
        }
        Alfa = Alfa1 / Alfa2;
        A[j] = Alfa;
    }
    return A[j];
}

double beta(int j, vector <pair <double, double>>& punkty, vector <double>& A, vector <double>& B)
{
    if (B[j] == 10000000)
    {
        double Beta1 = 0.0;
        double Beta2 = 0.0;
        double Beta = 0.0;
```

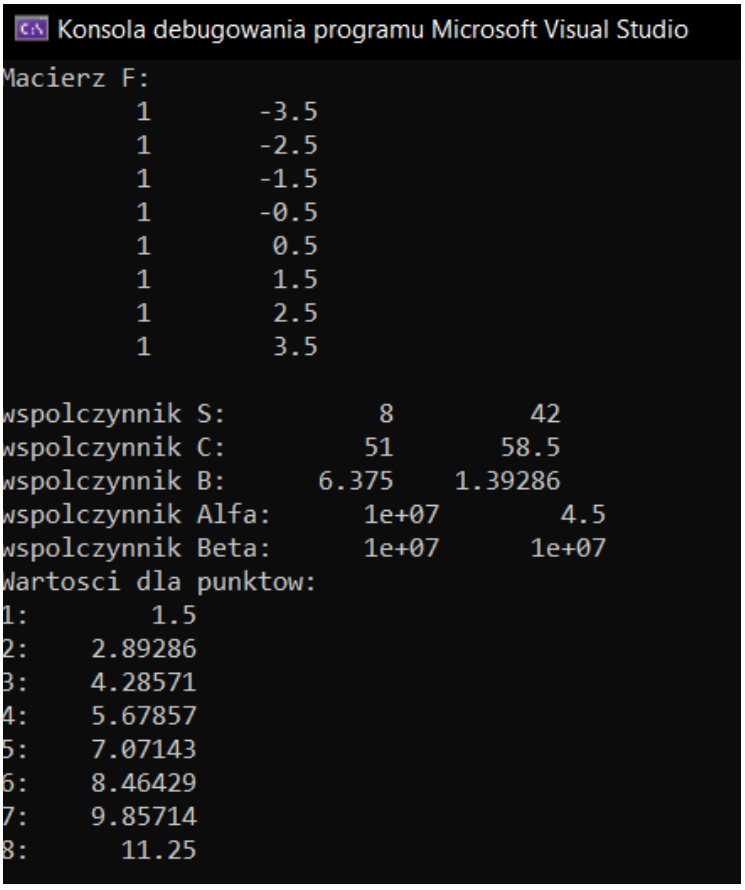
```
        for (auto i : punkty)
        {
            Beta1 += i.first * (fi(j, i.first, punkty, A, B) * fi(j - 1, i.first, punkty, A, B));
            Beta2 += pow(fi(j - 1, i.first, punkty, A, B), 2);
        }
        Beta = Beta1 / Beta2;
        B[j] = Beta;
    }
    return B[j];
}

double fi(int j, double x, vector <pair <double, double>>& punkty, vector <double>& A, vector <double>& B)
{
    double Fi = 0.0;
    if (j == -1)
    {
        return 0;
    }
    else if (j == 0)
    {
        return 1;
    }
    else if (j == 1)
    {
        Fi = (x - alfa(j, punkty, A, B)) * fi(j - 1, x, punkty, A, B);
        return Fi;
    }
    else
    {
        Fi = (x - alfa(j, punkty, A, B)) * fi(j - 1, x, punkty, A, B) - beta(j - 1, punkty, A, B) * fi(j - 2, x, punkty, A, B);
        return Fi;
    }
}
```

Wynik dla przykładu z poprzedniego laboratorium po usunięciu 1 węzła:

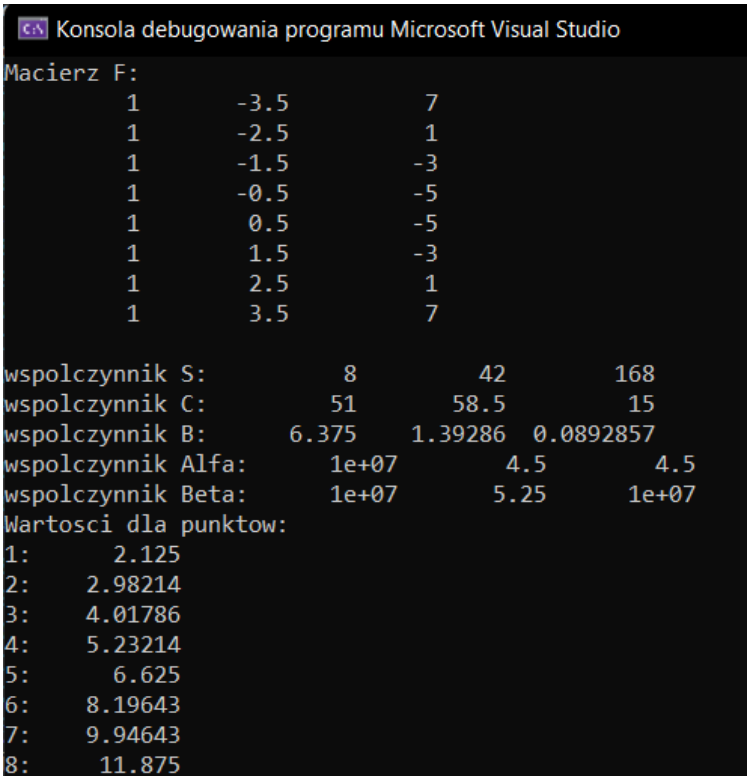


Wartość $1e + 07$ współczynników α i β oznacza, że nie były one obliczane przez funkcję. Faktycznie α_0 i β_0 zostały pominięte (α i β liczone są od j = 1), a do obliczenia φ_m wykorzystane jest β_{m-1}, co oznacza, że β_m nie jest obliczane.

Wynik dla przykładu z laboratorium 8, $m = 1$:



Wynik dla przykładu z laboratorium 8, $m = 2$:



Źródła

- http://galaxy.agh.edu.pl/~chwiej/mn/aproksymacja.pdf
- http://aproksymacja-wielomianami.eprace.edu.pl/267,Aproksymacja_za_pomoca_wielomianow_ortogonalnych.html