Wojciech Ładyga - zadanie 9

Język technologia: c++, GSL

Aby wyliczyć jawnie współczynniki wielomianu interpolacyjnego na podstawie danych wejściowych zawartych w tablicach xTab[N] i yTab[N] można wprost stworzyć układ równań i go rozwiązać.

Taka metoda da nam postać:
a4*(-1.23)^4 + a3*(-1.23)^3 + a2*(-1.23)^2 + a1*(-1.23)^1 + a0*(-1.23)^0 = 1.5129
a4*(-1.19)^4 + a3*(-1.19)^3 + a2*(-1.19)^2 + a1*(-1.19)^1 + a0*(-1.19)^0 = 1.4161
a4*(-0.74)^4 + a3*(-0.74)^3 + a2*(-0.74)^2 + a1*(-0.74)^1 + a0*(-0.74)^0 = 0.5476
a4*(0.11)^4 + a3*(0.11)^3 + a2*(0.11)^2 + a1*(0.11)^1 + a0*(0.11)^0 = 0.0121
a4*(2.56)^4 + a3*(2.56)^3 + a2*(2.56)^2 + a1*(2.56)^1 + a0*(2.56)^0 = 6.5536

Jak widać taki układ równań można umieścić w macierzy i rozwiązać np. za pomocą dekompozycji LU jak ja to zrobiłem. Następnie wyniki zapisałem do wektora a przechowującego współczynniki wielomianu. Ustaliłem precyzję równą 5 `setprecision(5)` gdyż większa nic nam nie daje.

Kod programu:

```
/*
* @Author: Wojciech Ladyga
* @Date: 2018-12-24
* @Description: Zad 9
*/
#include <iostream>
#include <iomanip>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_eigen.h>
using namespace std;

/*linki pomocnicze
    -wyklady
    -https://pl.wikipedia.org/wiki/Interpolacja_wielomianowa
    -http://www.algorytm.org/procedury-numeryczne/interpolacja-wielomianowa.html
    - dokumentacja GSL 2.5
    */

const int N = 5;
//dane wejsciowe
double xTab[N] = {-1.23, -1.19, -0.74, 0.11, 2.56};
double yTab[N] = {1.5129, 1.4161, 0.5476, 0.0121, 6.5536};

//test - ok
//const int N = 6;
//double xTab[N] = {2, 3, 6, 7, 8, 10};
//double yTab[N] = {0, 2, 3, 5, 1, 2};
```

```
gsl_vector *lu(gsl_vector *x, gsl_vector *y, gsl_vector *a, gsl_permutation
*permutation, gsl_matrix *countMat);
void wspolczynniki();
int main()
{
    wspolczynniki();
    return 0;
}

//funkcja przelicza wektor zawierający współczynniki
gsl_vector *lu(gsl_vector *x, gsl_vector *y, gsl_vector *a, gsl_permutation
*permutation, gsl_matrix *countMat)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            gsl_matrix_set(countMat, i, j, pow(gsl_vector_get(x, i), N - 1 - j));
        }
    }

    //rozwiązujemy macierz stosując dekompozycje LU
    int tmp;
    gsl_linalg_LU_decomp(countMat, permutation, &tmp);
    gsl_linalg_LU_solve(countMat, permutation, y, a);

    return a;
}

//funkcja przyjmuje dane i wyświetla wynik operacji
void wspolczynniki()
{

    gsl_vector *a = gsl_vector_alloc(N);                    //vector na
wspolczynniki
    gsl_permutation *permutation = gsl_permutation_alloc(N); //wektor permutacji
do dekompozycji LU
    //macierz pozwalająca umiescic odpowiednie wartosci typu ax^n
    gsl_matrix *countMat = gsl_matrix_alloc(N, N);

    //kopiuje zawartoc tablic do wektorow
    gsl_vector *x = gsl_vector_alloc(N);
    gsl_vector *y = gsl_vector_alloc(N);
    gsl_vector_view tmp_x = gsl_vector_view_array(xTab, N);
    gsl_vector_view tmp_y = gsl_vector_view_array(yTab, N);
    gsl_vector_memcpy(x, &tmp_x.vector);
    gsl_vector_memcpy(y, &tmp_y.vector);

    //wypisz wyniki
    for (int i = N - 1; i >= 0; i--)
    {
```

```
        cout << "a" << N - 1 - i << " = " << setprecision(5) << fixed <<
gsl_vector_get(lu(x, y, a, permutation, countMat), i) << endl;
    }

    //zwalnianie pamięci
    gsl_vector_free(x);
    gsl_vector_free(y);
    gsl_matrix_free(countMat);
    gsl_vector_free(a);
}
```

Wyniki działania programu to:

```
a0 = -0.00000
a1 = 0.00000
a2 = 1.00000
a3 = -0.00000
a4 = 0.00000
```