METODY NUMERYCZNE

ZADANIE 3

Artur Guniewicz

3. Dana jest macierz

$$\mathbf{A} = \begin{bmatrix} \frac{19}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & -\frac{17}{12} \\ \frac{13}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & -\frac{11}{12} & \frac{13}{12} \\ \frac{5}{6} & \frac{5}{6} & \frac{5}{6} & -\frac{1}{6} & \frac{5}{6} & \frac{5}{6} \\ \frac{5}{6} & \frac{5}{6} & -\frac{1}{6} & \frac{5}{6} & \frac{5}{6} & \frac{5}{6} \\ \frac{13}{12} & -\frac{11}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & \frac{13}{12} \\ -\frac{17}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & \frac{19}{12} \end{bmatrix}. \quad (4)$$

Przy użyciu metody potęgowej znajdź jej dwie największe na moduł wartości własne i odpowiadające im wektory własne.

Metoda: metoda potęgowa

Metoda potęgowa to iteracyjna metoda obliczania wektorów i wartości własnych. Oznacza to, że z każdą iteracją otrzymujemy coraz dokładniejszy wynik.

Algorytm wygląda następująco:

1. Znajdź takie y_1, że jego norma = 1
2. Krok iteracyjny:

$$Ay_k = z_k$$

$$y_{k+1} = \frac{z_k}{||z_k||}$$

3. Zakończ iterować w momencie gdy osiągnięte zostanie dane przybliżenie.
4. y - wektor własny; $||z_k||$ - wartość własna

5. Dokonaj ortogonalizacji z

$$Ay_k = z_k$$

$$z_k = z_k - \sum_{i=1}^{n-1} e_i (e_i^T z_k)$$

dla drugiej największej co do modułu wartości własnej: $z_k = z_k - e1(e_1^T z_k)$

6. Wylicz następne y

$$y_{k+1} = \frac{z_k}{||z_k||}$$

Kod programu:

```
/*
************************************************************
*                                                          *
*               Artur Guniewicz - Zadanie 3                *
*                                                          *
************************************************************
*/

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
#include <vector>
#include <gsl/gsl_math.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_blas.h>
#include <stdio.h>
#include <stdlib.h>

using namespace std;
```

```
const int ile_iter = 128;
const int N = 6;

int main()
{
   int i;
   double lambda[2];
   double A_data[] = {19.0 / 12.0, 13.0 / 12.0, 5.0 / 6.0, 5.0 / 6.0,
13.0 / 12.0, -17.0 / 12.0,
                      13.0 / 12.0, 13.0 / 12.0, 5.0 / 6.0, 5.0 / 6.0,
-11.0 / 12.0, 13.0 / 12.0,
                      5.0 / 6.0, 5.0 / 6.0, 5.0 / 6.0, -1.0 / 6.0, 5.0
/ 6.0, 5.0 / 6.0,
                      5.0 / 6.0, 5.0 / 6.0, -1.0 / 6.0, 5.0 / 6.0, 5.0
/ 6.0, 5.0 / 6.0,
                      13.0 / 12.0, -11.0 / 12.0, 5.0 / 6.0, 5.0 / 6.0,
13.0 / 12.0, 13.0 / 12.0,
                      -17.0 / 12.0, 13.0 / 12.0, 5.0 / 6.0, 5.0 / 6.0,
13.0 / 12.0, 19.0 / 12.0};

   gsl_matrix_view A = gsl_matrix_view_array(A_data, N, N);
   gsl_matrix *A_pom = gsl_matrix_alloc(N, N);
   gsl_vector *wynik = gsl_vector_alloc(N);
   gsl_vector *w = gsl_vector_alloc(N);
   gsl_vector *w_pom = gsl_vector_alloc(N);

   gsl_matrix_memcpy(A_pom, &A.matrix);

   cout << setprecision(8) << fixed;

   // Pierwsza wartość własna (na moduł)
   gsl_vector_set_all(wynik, 1.0); // wektor wynik składa się z samych
1

   for (i = 0; i < ile_iter; i++)
   {
       gsl_blas_dgemv(CblasNoTrans, 1.0, A_pom, wynik, 0.0, w);
       lambda[0] = gsl_blas_dnrm2(w) / gsl_blas_dnrm2(wynik); // dnrm2
-> liczy normę euklidesową z wektora ||x|| = sqrt(sum(xi^2))
```

```
        gsl_vector_scale(w, 1.0 / gsl_blas_dnrm2(w));           // scale
-> mnoży wektor w przez skalar
        gsl_vector_memcpy(wynik, w);                            //
kopiuje zawartość wektora w do wektora wynik
    }

    cout << "Wektor wlasny:" << endl;
    gsl_vector_fprintf(stdout, wynik, "%.8f"); // wypisuje na wyjście
wektor wynik
    cout << "Dla |lambda_0| = " << lambda[0] << "\n" << endl << endl;

    // Druga wartość własna (na moduł)

    gsl_vector_set(wynik, 1, -gsl_vector_get(wynik, 1)); // get zwraca 1
element wektora wynik; set ustawia wartość na 1 miejscu w wektorze
wynik

    for (i = 0; i < N; i++)
    {
        gsl_vector_set(w_pom, i, (double)fabs(gsl_vector_get(wynik,
i)));
        gsl_vector_set(w, i, gsl_vector_get(wynik, i));
    }

    int j;
    double pom1, pom2;

    for (i = 0; i < ile_iter; i++)
    {
        pom1 = 0;
        pom2 = 0;
        gsl_vector_memcpy(w, wynik);
        gsl_blas_dgemv(CblasNoTrans, 1.0, A_pom, w, 0.0, wynik);

        for (j = 0; j < N; j++)
            pom1 += (gsl_vector_get(w_pom, j) * gsl_vector_get(wynik,
j));

        // z = z-e(e^T*z)
```

```
        for (j = 0; j < N; j++)
        {
            pom2 = gsl_vector_get(wynik, j) - gsl_vector_get(w_pom, j) *
pom1;
            gsl_vector_set(wynik, j, pom2);
        }

        pom1 = 0;
        for (j = 0; j < N; j++)
            pom1 += (gsl_vector_get(wynik, j) * gsl_vector_get(wynik,
j));

        lambda[1] = sqrt(pom1);

        gsl_vector_scale(wynik, 1.0 / lambda[1]);
    }

    cout << "Wektor wlasny:" << endl;
    gsl_vector_fprintf(stdout, wynik, "%.8f");
    cout << "Dla |lambda_1| = " << lambda[1] << "\n" << endl;

    gsl_matrix_free(A_pom);
    gsl_vector_free(w);
    gsl_vector_free(w_pom);
    gsl_vector_free(wynik);

    return 0;
}
```

Kompilacja:

g++ Zadanie3.cpp -ansi -pedantic -Wall -I/<ścieżka dostępu do biblioteki gls> -lgsl -lgslcblas -lm -o Zadanie3 && ./Zadanie3

Wyniki:

```
Wektor wlasny:
0.40824829
0.40824829
0.40824829
0.40824829
0.40824829
0.40824829
Dla |lambda_0| = 4.00000000


Wektor wlasny:
0.70710696
-0.00000055
0.00000000
0.00000000
0.00000018
-0.70710660
Dla |lambda_1| = 3.00000000
```

Komentarz:

W programie zastosowałem funkcje z biblioteki GSL. Jest to bardzo szybki i efektowny (przez co bardzo dobry) sposób.

Dokumentacja ww biblioteki:
https://www.gnu.org/software/gsl/doc/html/index.html
Działania ważniejszych funkcji opisane w komentarzach w kodzie programu.