Zadanie Numeryczne 8

Zadanie polegało na znalezieniu wartości własnych macierzy A z dokładnością 10^{-8}

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & -1 \end{bmatrix}$$

Korzystając z metody potęgowe, Rayleigha i metody iteracyjnej QR.

Metoda potęgowa oraz metoda potęgowa z ilorazem Rayleigha są napisane w c++ aby je skompilować i uruchomić można skorzystać z komendy g++ ZN8.cpp oraz ZN8R.cpp. Metoda iteracyjna QR została napisana w MATLAB.

Metoda potęgowa służy do wyznaczenia maksymalnej co do modułu wartości własnej i odpowiadającego jej wektora własnego. Iteracje rozpoczyna się od wybrania dowolnego wektora x_0, następnie mnoży się macierz A przez wektor x_0 otrzymując nowy wektor, który należy znormalizować. Wybieram największą wartość co do modułu z otrzymanego wektora i dziele otrzymany wektor przez tę wartość. Iteracje powtarzam aż osiągnę zadaną dokładność. W przypadku macierzy A aby uzyskać wartość własną z dokładnością 10^{-8} należało wykonać 28 iteracji a otrzymany wynik to:

$$\lambda = 8.54851282$$

Metoda potęgowa wzbogacona o iloraz Rayleigha

$$\lambda = \frac{x^T A x}{x^T x}$$

Początkowo przebiega tak samo, jednak po wyznaczeniu wektora własnego x to z niego korzystamy aby policzyć wartość własną największą co do modułu. A aby uzyskać wartość własną z dokładnością 10^{-8} należało wykonać 12 iteracji, jest to o wiele mniej niż w zwykłej metodzie potęgowej dlatego możemy zauważyć, że dzięki zastosowaniu ilorazu Rayleigha dokładny wynik otrzymamy dzięki mniejszej ilości iteracji.

$$\lambda = 8.548512833$$

W metodzie potęgowej przy każdej iteracji wykonujemy mnożenie macierzy 3x3 przez wektor co ma złożoność obliczeniowa $O(n^3)$. Wykonujemy też trzy dzielenia oraz przepisanie wektora z do wektora x. W metodzie potęgowej z ilorazem Rayleigha mnożenie macierzy 3x3 i wektora wykonujemy dwa razy w każdej iteracji. Następnie aby policzyć licznik i mianownik ilorazu Rayleigha wykonujemy mnożenie dwóch wektorów co zajmuje 5 operacji 3 mnożenia i 2 dodawania.

Metoda iteracyjna QR

Dzięki tej metodzie możemy wyznaczyć wszystkie wartości własne macierzy. Polega ona na iteracyjnym rozkładnie macierzy metoda QR a następnie stworzenie nowej macierzy poprzez pomnożenie otrzymanych z rozkładu macierzy R i Q. Iteracje powtarza się aż uzyska się zadaną dokładność. Wartości własne to liczby na diagonali macierzy A.

Rozkład QR został wykonany za pomocą wbudowanej funkcji w programie MATLAB

Otrzymane wartości własne:

$\lambda_1 =$ `8.548512838045500` $\lambda_2 =$ `-4.574087210679819` $\lambda_3 =$ `0.025574372634319`

Metoda Potęgowa

```
#include <iostream>
#include<math.h>

int main()
{
    double A[3][3] = { 1,2,3,
                       2,4,5,
                       3,5,-1 };

    double x[3] = { 1,1,1 };
    double z[3] = { 0,0,0 };
    double lambda = 0;
    double temp = 0;
    double eps = 10e-8;
    int iteracje = 0;

   do {
        iteracje++;
         for (int i = 0; i < 3; i++) {
            z[i] = 0;
            for (int j = 0; j < 3; j++) {

                 z[i] += A[i][j] * x[j];
             }
         }
         temp = lambda;

         lambda = fabs(z[0]);
         for (int i = 1; i < 3; i++) {
             if (fabs(z[i]) > lambda)
                 lambda = fabs(z[i]);
         }

         for (int i = 0; i < 3; i++) {
             z[i] = z[i] / lambda;
         }


         for (int i = 0; i < 3; i++) {
             x[i] = z[i];
         }

   } while (fabs(lambda - temp)>eps);

     printf("lambda: %.9f \n", lambda);
     printf("iteracje: %d \n", iteracje);

    return 0;
```

Metoda Potęgowa wraz z zastosowaniem ilorazem Rayleigha

```
#include <iostream>
#include<math.h>

int main()
{
    double A[3][3] = { 1,2,3,
        2,4,5,
        3,5,-1 };
```

```
double x[3] = { 1,1,1 };
double z[3] = { 0,0,0 };
double temp = 0;
double eps = 10e-8;
int iteracje = 0;
double lambda_Rayleigha = 0;
double licznik = 0;
double mianownik = 0;
double s[3];
double t;
do {

iteracje++;
for (int i = 0; i < 3; i++) {
    z[i] = 0;
    for (int j = 0; j < 3; j++) {

        z[i] += A[i][j] * x[j];
    }
}

t = fabs(z[0]);
for (int i = 1; i < 3; i++) {
    if (fabs(z[i]) > t)
        t = fabs(z[i]);
}

for (int i = 0; i < 3; i++) {
    z[i] = z[i] / t;
}

for (int i = 0; i < 3; i++) {
    x[i] = z[i];
}
for (int i = 0; i < 3; i++) {
    s[i] = 0;
}

for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 3; j++) {

        s[i] += A[i][j] * x[j];
    }
}

licznik = 0;

for (int i = 0; i < 3; i++) {
    licznik += s[i] * x[i];
}

double mianownik = 0;

for (int i = 0; i < 3; i++) {
    mianownik += x[i] * x[i];
}
temp = lambda_Rayleigha;
lambda_Rayleigha = licznik / mianownik;
```

```
    } while (fabs(lambda_Rayleigha - temp) > eps);

    printf("lambda: %.9f \n", lambda_Rayleigha);
    printf("iteracje: %d \n", iteracje);

    return 0;
}
```

Metoda iteracyjna QR

```
format long

A =
     1     2     3
     2     4     5
     3     5    -1

for i=1:16,
[Q,R]=qr(A); A= R*Q
end,

A =
   6.714285714285712   4.536023611499572   0.358568582800325
   4.536023611499572  -2.714285714285713  -0.267261241912429
   0.358568582800324  -0.267261241912429  -0.000000000000000
A =
   7.965255157437563  -2.704379160450078  -0.001134971667876
  -2.704379160450076  -3.990828582167332  -0.001954653854755
  -0.001134971667876  -0.001954653854756   0.025573424729766
A =
   8.376053031349258   1.494449357450079   0.000003450637556
   1.494449357450082  -4.401627403952673  -0.000011702534970
   0.000003450637556  -0.000011702534970   0.025574372603413
A =
   8.498669370249129  -0.807212314584437  -0.000000010371943
  -0.807212314584434  -4.524243742883449  -0.000000066698470
  -0.000000010371943  -0.000000066698470   0.025574372634318
A =
   8.534203664090111   0.433092384544391   0.000000000031072
   0.433092384544394  -4.559778036724432  -0.000000000374964
   0.000000000031072  -0.000000000374964   0.025574372634319
A =
   8.544412881295312  -0.231916972578239  -0.000000000000093
  -0.231916972578236  -4.569987253929633  -0.000000000002100
  -0.000000000000093  -0.000000000002100   0.025574372634319
```

```
A =
   8.547338752160456   0.124120426119306   0.000000000000000
   0.124120426119309  -4.572913124794777  -0.000000000000012
   0.000000000000000  -0.000000000000012   0.025574372634319
A =
   8.548176681669915  -0.066417857801694  -0.000000000000000
  -0.066417857801692  -4.573751054304234   0.000000000000000
  -0.000000000000000  -0.000000000000000   0.025574372634319
A =
   8.548416604129494   0.035539120632863   0.000000000000000
   0.035539120632866  -4.573990976763811  -0.000000000000000
   0.000000000000000  -0.000000000000000   0.025574372634319
A =
   8.548485296557152  -0.019016160055515  -0.000000000000000
  -0.019016160055512  -4.574059669191469   0.000000000000000
  -0.000000000000000  -0.000000000000000   0.025574372634319
A =
   8.548504963621360   0.010175068652204   0.000000000000000
   0.010175068652207  -4.574079336255675  -0.000000000000000
   0.000000000000000  -0.000000000000000   0.025574372634319
A =
   8.548510594395651  -0.005444417328953  -0.000000000000000
  -0.005444417328950  -4.574084967029967   0.000000000000000
  -0.000000000000000  -0.000000000000000   0.025574372634319
A =
   8.548512206510969   0.002913166680979   0.000000000000000
   0.002913166680982  -4.574086579145287  -0.000000000000000
   0.000000000000000  -0.000000000000000   0.025574372634319
A =
   8.548512668066456  -0.001558759891898  -0.000000000000000
  -0.001558759891895  -4.574087040700776   0.000000000000000
  -0.000000000000000  -0.000000000000000   0.025574372634319
A =
   8.548512800211746   0.000834051946097   0.000000000000000
   0.000834051946100  -4.574087172846065  -0.000000000000000
   0.000000000000000  -0.000000000000000   0.025574372634319
A =
   8.548512838045500  -0.000446279537609  -0.000000000000000
  -0.000446279537606  -4.574087210679819   0.000000000000000
  -0.000000000000000  -0.000000000000000   0.025574372634319
```