N9

Szymon Pach

Dla uproszczenia implementacji użyłem biblioteki Eigenlib (którą załączam w tym pliku), która służy mi do przechowywania wektorów i macierzy.

Do obliczenia macierzy użyłem trzech metod; potęgowej, Rayleigha i iteracyjnej metodą QR.

Wartości własne:

(0.025574373,0)

(8.5485129,0)

(-4.5740872,0)

Wartosci wlasne:
(0.0255743726343174,0)
(8.54851285322277,0)
(-4.57408722585711,0)

Potegowa:
8.54851245880127
-4.57408714294434
0.0255743730813265

Rayleigha
8.54851245880127
-4.57408714294434
0.0255746841430664

QR
8.54851285076584
-4.57408722340015
0.0255743726343184

```
#include<string>
#include<iostream>
#include<cmath>
#include<algorithm>
#include<cstring>
#include<Eigen/Dense>
using namespace Eigen;

void powIteration(Matrix3d* m, double* eVals) {
        VectorXd out(3), tmp(3);
        double prevFactor = 0;
        VectorXd vec(3);
        vec<< 1, 1, 1;          //losowy wektor, wybrałem [1,1,1]
        float currentFactor;
        int counter = 0;
        for(int i = 0 ; i<3;i++) {
                while(true) {
                        std::cout<<"counter = "<<counter<<std::endl;
                        out = *m * vec;
                        out /= out.norm();
                        tmp = *m * out;
                        currentFactor = out.dot(tmp);
                        vec = out;
                        std::cout<<fabs(prevFactor-currentFactor)<<std::endl;
                        std::cout<<"lambda "<<i<<" = "<<prevFactor<<std::endl;
                        if(fabs(prevFactor-currentFactor) <= 1e-8) {
                                        eVals[i] = prevFactor;
                                vec <<1,1,1;
                                break;

```

```
                                        }
                                        prevFactor = currentFactor;
                                        counter++;
                                }       // jako ,że A = suma po i lambda(i) * eigenvector(i) * eigenvector(i) transponowany
                                        //zrobimy tak,żeby lambda(i) się zerowała i obliczymy następne lambda
                                *m = *m - (prevFactor * out*out.transpose());
                                counter=0;
                                std::cout<<*m<<std::endl;
                        }
}
void RayleighIteration(Matrix3d* m, double* eVals) {

                VectorXd out(3);
                double prevFactor = 30; //jakaś wartość poczatkowa, niech bedzie 30
                VectorXd vec(3);
                vec<< 1, 1, 1;          //losowy wektor, wybrałem [1,1,1]
                float currentFactor;
                int counter = 0;
                for(int i = 0 ; i<2;i++) {
                                while(true) {
                                        std::cout<<"counter = "<<counter<<std::endl;
                                        out = (*m -prevFactor*Matrix3d::Identity()).inverse()* vec;
                                        out /= out.norm();
                                        currentFactor = vec.dot(*m*vec)/vec.dot(vec);
                                        vec = out;
                                        std::cout<<fabs(prevFactor-currentFactor)<<std::endl;
                                        std::cout<<"lambda "<<i<<" = "<<prevFactor<<std::endl;
                                        if(fabs(prevFactor-currentFactor) <= 1e-8) {
                                                        eVals[i] = prevFactor;
                                                vec <<1,1,1;
                                                break;

                                        }
                                        prevFactor = currentFactor;
                                        counter++;
                                }
                                counter = 0;
                                if(i == 0)// Trace(A) = suma wartosci wlasnych.Szacujemy przyblizenie drugiej
                                        prevFactor = m->trace() - prevFactor;
                                if(i == 1)   // ostatnia wyliczamy odejmujac od trace(A) 2 pozostale wartosci wlasne
                                        eVals[2] = m->trace() - eVals[0] - eVals[1];
                                std::cout<<*m<<std::endl;
                }
}
void iterativeQR(Matrix3d* m, double* eVals) {
                Matrix3d Bprev = *m, Bcurrent = Matrix3d::Identity(),Q,R;
                int counter=0;
                while(true) {
                                Q = Bprev.colPivHouseholderQr().matrixQ();
                                R = Q.inverse() * Bprev;
                                Bcurrent = R * Q;
                                if(fabs(Bcurrent(0) - Bprev(0)) <= 1e-8)
                                        break;
                                Bprev = Bcurrent;
                                counter++;
                }
                for(int i=0;i<3;i++)
                                eVals[i] = Bcurrent(i*3 +i);
}
void initMatrix(Matrix3d* m) {
                *m <<   1, 2, 3,
                                2, 4, 5,
                                3, 5, -1;
}
int main () {
                //macierz zapisana column wise
```

```
97.             std::string labels[3] = {"Potegowa", "Rayleigha", "QR"};
98.             Matrix3d* A = new Matrix3d();
99.             initMatrix(A);
100.            std::cout<<*A<<std::endl;
101.            double eigVals[9] = {0}; //tablica przechowujaca wartosci wlasne liczone
102.                                                                    //3 metodami,w sumie 9
103.            //potegowa
104.            powIteration(A, eigVals);
105.            //Rayleigha
106.            initMatrix(A);
107.            RayleighIteration(A, eigVals+3);
108.
109.            //QR
110.            initMatrix(A);
111.            iterativeQR(A, eigVals+6);
112.            std::cout<<"Wartosci wlasne: "<<A->eigenvalues()<<std::endl;
113.            for(int i=0;i<3;i++) {
114.                    std::cout<<labels[i]<<std::endl;
115.                    for(int j=0;j<3;j++)
116.                            std::cout<<eigVals[i*3 +j]<<std::endl;
117.            }
118. }
```