Sprawozdanie MNUM
Projekt nr.2
Zadanie 2.24

Kajetan Kaczmarek

23 kwietnia 2018

1. Opis zastosowanych algorytmów :

   (a) W pierwszym zadaniu zaimplementowałem algorytm przeprowadzający faktoryzację QR danej macierzy. Program przyjmuje jako wartości wejściowe tolerancję, macierz do faktoryzacji oraz maksymalną ilość iteracji po których działanie programu ma być przerwane.Metoda bez przesunięć iterowała faktoryzacjęQR i następnie korzystała ze wzoru

   $$A = R * Q$$

   (b) Przy drugim poleceniu zrealizowałem algorytm aproksymacji funkcji przez metodę namniejszych kwadratów.Dla próbek

Y	X
-5.4606	-5
-3.8804	-4
-1.9699	-3
-1.6666	-2
-0.0764	-1
-0.3971	0
-1.0303	1
-4.5483	2
-11.5280	3
-21.6417	4
-34.4458	5

   zastosowałem dwa warianty algorytmu:

   - Z użyciem układu równań normalnych, tj.

     $$A^T A x = A^T b$$

- Z użyciem układu równań liniowych z macierzą R powstałą na skutek rozkładu QR, tj.

$$Rx = Q^T b$$

(c) Kod moich programów

- Kod główny pierwszego programu

```
n = 5;

Iterations = zeros(30,9); % Macierz prczechowuj ca ilo  iteracji

%Poni ej kolejne p tle licz ce wyniki dla r nych rozmiar w macierzy

for i=1:30
    A = Generate_Matrix_Sym(n);
    [X,Iterations(i,1)] = QR_Factorization_NoShift(A,10e-6,10000);


    A = Generate_Matrix_Sym(n);
    [X,Iterations(i,2)] = QR_Factorization_Shift(A,10e-6,10000);

    A = Generate_Matrix_Asym(n);
    [X,Iterations(i,3)] = QR_Factorization_Shift(A,10e-6,10000);
end

n = 10;

for i=1:30
    A = Generate_Matrix_Sym(n);
    [X,Iterations(i,4)] = QR_Factorization_NoShift(A,10e-6,10000);

    A = Generate_Matrix_Sym(n);
    [X,Iterations(i,5)] = QR_Factorization_Shift(A,10e-6,10000)
```

```
;

    A = Generate_Matrix_Asym(n);
    [X,Iterations(i,6)] = QR_Factorization_Shift(A,10e-6,10000);
end

n = 20;

for i=1:30
    A = Generate_Matrix_Sym(n);
    [X,Iterations(i,7)] = QR_Factorization_NoShift(A,10e-6,10000);

    A = Generate_Matrix_Sym(n);
    [X,Iterations(i,8)] = QR_Factorization_Shift(A,10e-6,10000);

    A = Generate_Matrix_Asym(n);
    [X,Iterations(i,9)] = QR_Factorization_Shift(A,10e-6,10000);
end
AVG = mean(Iterations,1);
```

- Faktoryzacja QR

```
function [Q,R] = Factorize_QR(A) %Funkcja realizuj ca faktoryzacje QR na macierzy A, zwracaj ca macierze Q i R
[m, n]=size(A); % Zaczynamy od ocenienia rozmiaru A i inicjalizacji odpowiednich macierzy Q, R i pomocniczej d

Q = zeros(m,n);
R = zeros(n,n);
d = zeros(1,n);

for i=1:n
    Q(:,i)=A(:,i);
    R(i,i)=1;
    d(i)=Q(:,i)'*Q(:,i);
```

```
    for j = i+1:n
        R(i,j) = (Q(:,i)'*A(:,j))/d(i);
        A(:,j) = A(:,j) - R(i,j)*Q(:,i);
    end
end

for i=1:n
    dd=norm(Q(:,i));
    Q(:,i)=Q(:,i)/dd;
    R(i,i:n)=R(i,i:n)*dd;
end
return
```

- Użycie faktoryzacji QR do znalezienia wartości własnych macierzy bez przesunięć

```
function [ev,i] = QR_Factorization_NoShift(A, tol, imax) %Prosty program do liczenia warto ci w asnych bez przesuni
% tol - tolerancja na null elementy
% imax - g rna granica iteracji

n = size(A,1);
i=1;
while i < imax & max(max(A - diag(diag(A)))) > tol
    [Q,R] = Factorize_QR(A); % Iteracyjne u ycie faktoryzacji
    A = R*Q;
    i=i+1;
end
if i > imax
    error('Ilo  iteracji przekroczy a warto  maksymaln ');
end

ev = diag(A);
```

- Użycie faktoryzacji QR do znalezienia wartości własnych macierzy z przesunięciami

```
function [ev,iter] = QR_Factorization_Shift(A, tol, imax) %Program wyliczaj cy aproksymat funkcji u ywaj c rozk adu QR
% tol - tolerancja na null elementy
```

```
% imax - g rna granica iteracji

n = size(A,1);
ISubmatrix = A;
ev = diag(ones(n));
iter = 0;
for k=n:-1:2,
    DK = ISubmatrix;
    i = 0;
    while i<=imax && max(abs(DK(k,1:k-1)))>tol,
        DD = DK(k-1:k,k-1:k);
        [ev1,ev2] = quadpolynroots(1,-(DD(1,1)+DD(2,2)),DD(2,2)*DD(1,1)-DD(2,1)*DD(1,2));
        if abs(ev1 -DD(2,2)) < abs(ev2 -DD(2,2))
            shift = ev1;
        else
            shift = ev2;
        end
        DP = DK - eye(k)*shift;
        [Q,R] = Factorize_QR(DP);
        DK = R*Q + eye(k)*shift;
        i = i+1;
        iter = iter+1;
    end


if i > imax
    error('Ilo  iteracji przekroczy a warto  maksymaln ');
end

ev(k) =DK(k,k);

if k > 2
    ISubmatrix = DK(1:k-1,1:k-1);
else
    ev(1) = DK(1,1);
end
end
```

- Kod główny drugiego programu

- Użycie faktoryzacji QR do znalezienia wartości własnych macierzy bez przesunięć

```
Y = [-5.4606 -3.8804 -1.9699 -1.6666 -0.0764 -0.3971 -1.0303 -4.5483 -11.5280 -21.6417 -34.4458];
X = [ -5 -4 -3 -2 -1 0 1 2 3 4 5]; %

n = 8; %Rz d wielomianu dla kt rego sprawdzamy
for n = 1:9
A  = LLSPQR(X, Y, n);
Ans = polyval(A,X);

disp(n);
disp(norm(Ans - Y));
disp(norm(Ans - Y,Inf));
end
figure;
plot(X,Y);
hold on;
plot(X,Ans);
hold off;
```

- LLSP z użyciem równań normalnych

```
function [A] = LLSPNormals (X1,Y1,n) % Program do wyliczenia aproksymaty funkcji u ywaj c serii r wna  normalnych
X = zeros(n); %Inicjalizujemy warto ci macierzy rozwi za
Y = zeros(n,1);
N1 = size(Y1); %Pobieramy rzmiar macierzy
N = N1(2);

for k = 1:n
    Y(k) = sum(X1.^(k-1).*Y1); % Wyliczamy warto ci macierzy X i Y dla rozwi za
        for j = 1:n
            X(k,j) = sum(X1.^(k+j-2));
        end
end
X(1,1) = N;
A = X^(-1)*Y;
```

```
A = flipud(A); %Odwracamy kolejno    na
    potrzeby funkcji polyval
```

- LLSP z użyciem faktoryzacji QR

```
function [A] = LLSPQR(X1,Y1,n) %Wyliczanie
    aproksymaty funkcji z u yciem
    faktoryzacji QR

X = zeros(n);
Y = zeros(n,1);

N1 = size(Y1);
N = N1(2);

for k = 1:n
    Y(k) = sum(X1.^(k-1).*Y1);
        for j = 1:n
            X(k,j) = sum(X1.^(k+j-2));
        end
end

X(1,1) = N;

[Q,R] = Factorize_QR(X);

A = R\Q'*Y;

A = flipud(A);
```

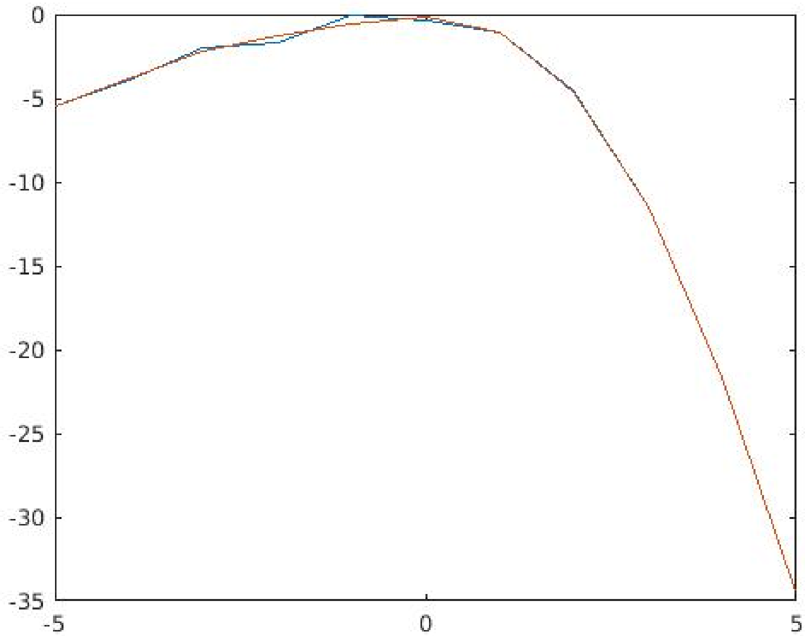
(d) Wyniki :

- Wyniki QR

Rozmiar macierzy	Metoda	Średnia ilość iteracji
5	Bez przesunięć	72.5
5	Z przesunięciami	7.33
5	Z przesunięciami Asymetryczna	9.53
10	Bez przesunięć	215.46
10	Z przesunięciami	14.10
10	Z przesunięciami Asymetryczna	21
20	Bez przesunięć	1824.4
20	Z przesunięciami	28.87
20	Z przesunięciami Asymetryczna	44.73

- Wyniki LLSP

N	Norma Euklidesowa	Norma Czebyszewa
1	34.3326	26.5690
2	24.5832	15.1424
3	7.3647	3.1643
4	1.4390	0.6769
5	1.3958	0.6486
6	0.8501	0.4642
7	0.7595	0.4239
8	0.7069	0.4448
9	0.7410	0.4819



(e) Wnioski : Obydwie metody wydają się oferować dostateczną precyzję. Dla odpowiednio dużej ilości iteracji lub odpowiednio wysokim stopniu wielomianu metody sprawdzają się.Widać także że wraz ze wzrostem macierzy metoda wyznaczania wartości własnych bez przesunięcia staje się zdecydowanie wolniejsza