

%%%%%%%%%% Teil a) und b)

%% Zunaechst die Testmatrizen vom Blatt generieren

```
function [A B] = testmatrizen(d)
```

```
n = length(d);
S = triu(diag(n:-1:1,0) + ones(n,n)); %%% HINWEIS: diag(v) = diag(v,k) mit k = 0
A = S*diag(d,0)*inv(S);
```

```
z = diag(sqrt(1:n),0) + ones(n,n);
[Q R] = qr(z);
B = Q*diag(d,0)*Q';
```

```
end
```

```
%
%% Potenzenmethode
function res = potenzenmethode(A,y_0,k)
```

```
res = zeros(1,k);
y_0=y_0/norm(y_0);
for j = 1:k
    y_1 = A*y_0;
    rayl = y_0'*y_1;
    res(j) = rayl;
    y_0 = y_1/norm(y_1);
end
```

```
end
%
%% Inverse Potenzenmethode
function res = inversepotenzenmethode(A,y_0,mu,k)
```

```
n = length(A);
[Q R] = qr(mu*eye(n)-A);
res = zeros(1,k);
y_0=y_0/norm(y_0);
for j = 1:k
    y_1 = R\((Q'*y_0);
    rayl = y_0'*y_1;
    res(j) = mu-1/rayl;
    y_0 = y_1/norm(y_1);
end
```

```
end
```

```
end
%
%% Ausgabeprogramm pp2ab
clear all, close all
```

```
format long
```

```
d1 = (1:10)';
d2 = [ones(9,1);2];
d3 = 1-2.^(-(1:.5:5));
```

```
[A1 B1] = testmatrizen(d1);
[A2 B2] = testmatrizen(d2);
[A3 B3] = testmatrizen(d3);
```

% Die groessten Eigenwerte von A und B sind jeweils die groessten Eintraege der pos.

% Vektoren d:

```
l1 = max(d1);
l2 = max(d2);
l3 = max(d3);
```

```
k = 5; % Anzahl Iterationsschritte
```

```
y0 = ones(10,1); % fuer 1 und 2
z0 = ones(9,1); % fuer 3
```

```
resA1 = potenzenmethode(A1,y0,k);
resA2 = potenzenmethode(A2,y0,k);
resA3 = potenzenmethode(A3,z0,k);
resB1 = potenzenmethode(B1,y0,k);
resB2 = potenzenmethode(B2,y0,k);
```

```

resB3      = potenzenmethode(B3,z0,k);
resA1inv   = inversepotenzenmethode(A1,y0,9.6,k);
resA2inv   = inversepotenzenmethode(A2,y0,1.9,k);
resA3inv   = inversepotenzenmethode(A3,z0,.98,k);
resB1inv   = inversepotenzenmethode(B1,y0,9.6,k);
resB2inv   = inversepotenzenmethode(B2,y0,1.8,k);
resB3inv   = inversepotenzenmethode(B3,z0,.98,k);

```

```
% Fehler plotten fuer PM
```

```

figure
hold on
subplot(2,3,1)
plot(1:k,abs(resA1-l1*ones(1,k)))
subplot(2,3,4)
plot(1:k,abs(resB1-l1*ones(1,k)))
subplot(2,3,2)
plot(1:k,abs(resA2-l2*ones(1,k)))
subplot(2,3,5)
plot(1:k,abs(resB2-l2*ones(1,k)))
subplot(2,3,3)
plot(1:k,abs(resA3-l3*ones(1,k)))
subplot(2,3,6)
plot(1:k,abs(resB3-l3*ones(1,k)))

```

```
% Fehler plotten fuer inv. PM
```

```

figure
hold on
subplot(2,3,1)
plot(1:k,abs(resA1inv-l1*ones(1,k)))
subplot(2,3,4)
plot(1:k,abs(resB1inv-l1*ones(1,k)))
subplot(2,3,2)
plot(1:k,abs(resA2inv-l2*ones(1,k)))
subplot(2,3,5)
plot(1:k,abs(resB2inv-l2*ones(1,k)))
subplot(2,3,3)
plot(1:k,abs(resA3inv-l3*ones(1,k)))
subplot(2,3,6)
plot(1:k,abs(resB3inv-l3*ones(1,k)))
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
%%%%%%%%% Teil c)
```

```

function [A diag] =pp2c(A)
n=size(A,1);
n1=n-1;
    for i=1:n1
        alphi=-sign(A(i,i))*sqrt(A(i,i)^2+A(i+1,i)^2);
        u=alphi*(alphi-A(i,i));
        u1=A(i,i)-alphi;
        u2=A(i+1,i);
        diag(i)=alphi;
        tmp=(u1*A(i,i+1)+u2*A(i+1,i+1))/u;
        A(i,i+1)=A(i,i+1)-tmp*u1;
        A(i+1,i+1)=A(i+1,i+1)-tmp*u2;
        if i<n1
            tmp=u2*A(i+1,i+2)/u;
            A(i,i+2)=A(i,i+2)-tmp*u1;
            A(i+1,i+2)=A(i+1,i+2)-tmp*u2;
        end
        u=sqrt(2*u);
        A(i,i)=u1/u;
        A(i+1,i)=u2/u;
    end
    diag(n)=A(n,n);

```

```
end
```

```

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
%%%%%%%%% Teil d)
```

```
% QR-Algo mit normalem Shift funktioniert fuer die angegebene Matrix nicht, deshalb hier nur der Algorithmus mit Wilkinson-Shift
```

```
%% QR-Zerlegung mit Wilkinson Shift (lasse fuer normalen Shift ueberfluessige Zeilen weg)
```

```
function res = qr_mit_wilkinson(H,mu)
```

```
% das mu muss eigentlich nicht mit in den Funktionsaufruf, sondern wird eigentlich vom Programm selbst
ausgerechnet
% hier ist das nur als billiger Trick eingefuehrt, damit wir von aussen Vorgeben koennen, gegen welchen
Eigenwert das Programm als erstes konvergiert (s. Aufruf unten)
    n = length(H);
    TOL = 0;
    j = 1;

while(abs(H(n,n-1)) > TOL)

    [H R] = qr(H-mu*eye(n)); % in einem schnellen Programm muesste hier die QR aus Teil c) verwendet werden
    H = R*H + mu*eye(n); % in einem schnellen Programm muesste hier die spezielle RQ-Multiplikation in O(n)
    stehen (siehe Uebung)

    Hnn = H(n,n);
    Hn1n1 = H(n-1,n-1);
    d = (Hn1n1 - Hnn)/2;
    TOL = eps*(abs(Hnn) + abs(Hn1n1));
    mu = Hnn + d - sign(d)*sqrt(d^2 + H(n-1,n)^2);
    res(j) = Hnn;
    j = j+1;

end

end

%
%% Ausgabeprogramm pp2d
clear all
close all

A = diag(-2*ones(1,20)) + diag(ones(1,19),-1) + diag(ones(1,19),1);

mu = -.02; % mu ist nahe am betragsmaessig kleinsten EW von A
res = qr_mit_wilkinson(A,mu) % hier benutzen wir den oben erwaehten schmutzigen Trick um von uns aus zu
wissen, was der erste berechnete Eigenwert ist
k = length(res)

plot(1:k,abs(res(k)*ones(1,k)-res))
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% % % % % Teil e)
% Testmatrix und Vektor
A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20];
b = [21 22 23 24]';
% Singulaerwertzerlegung
[U S V] = svd(A);
% Definiert den Rang der Matrix als Anzahl aller Singulaerwerte, die groesser als eine numerisch sehr kleine
Zahl sind
TOL = 10e-3;
k = 1;
sigma1 = S(1,1);
while(S(k+1,k+1)/sigma1 > TOL)
    k = k+1;
end
% Invertiere Sigma
Splus = S';
for j=1:k
    Splus(j,j) = 1/Splus(j,j);
end
% Berechne x
x = V*Splus*U'*b;

norm(A*x-b)
```

