Alg. Meth. i.d. Numerik Übung 2

Florian Burndorfer, Jakob Fromherz,

Luna Sandner, Franz Scharnreitner

June 11, 2021

1 QRFact

```
function [A,D,p,k] = QRFact(A)

[m,n]=size(A);
n_=n;
D = zeros([min(m,n),1]);

p=1:n;

for j=n:-1:1
    sigma_(j) = dot(A(:,j),A(:,j));
    sigma(j) = sigma_(j);
    if sigma(j)==0
        temp=p(j);
        p(j) = p(n_);
        p(n_)=temp;
        n_= n_-1;
    end
end

for j=1:n_



   piv = j-1;
   val=-1;
   for l=j:n_
       div=sigma(p(l))/sigma_(p(l));

       if div>val
           val=div;
           piv=l;
       end
   end

   if piv < j
       k=j-1;
```

```
        return;
    end

    temp=p(j);
    p(j)=p(piv);
    p(piv)=temp;


    sigma(p(j)) = dot(A(j:m,p(j)),A(j:m,p(j)));

    if sigma(p(j))<m*eps^2 * sigma_(p(j))
        k=j-1;
        return;
    end

    D(j)= - sign(A(j,p(j)))*sqrt(sigma(p(j)));
    A(j,p(j))= A(j,p(j))-D(j);

    for i=j+1:n_
        gamma = dot(A(j:m,p(i)),A(j:m,p(j)))/(-D(j)*A(j,p(j)));
        A(j:m,p(i))=A(j:m,p(i))-gamma*A(j:m,p(j));
        sigma(p(i))=sigma(p(i))-(A(j,p(i)))^2;
        if sigma(p(i))<m*eps*sigma_(p(i))
            sigma(p(i))=dot(A(j+1:m,p(i)),A(j+1:m,p(i)));
        end
    end
end
k=n_;

end
```

2 QRSolve

```
function [x] = QRSolve(B,D,p,k,b)

    [m,n] = size(B);
    if(k < n)
        x = zeros(n,1);
        return
    end


    c = b;
    for j=1:k
        v=zeros(m,1);
        if(j>1)
            v(1:j-1)=0;
        end
        v(j:m)=B(j:m,p(j));

        c=transpose(eye(m)-(2/dot(v,v))*(v*transpose(v)))*c;
    end

    pt(p)=1:length(p); %inverse permutation vector
    R=B(:,p);
    x=zeros(n,1);

    x(k)=c(k)/D(k);
    for i=k-1:-1:1
        x(i)=(c(i)-dot(R(i,i+1:k),x(i+1:k)))/D(i);
    end
    x=x(pt);
end
```

3 COMPQ/COMPR

```
function [Q] = compQ(B,p,k)
[m,~]=size(B);
Q=eye(m);

for j=1:k

    v=zeros(m,1);
    if(j>1)
        v(1:j-1)=0;
    end
    v(j:m)=B(j:m,p(j));

    P=eye(m)-(2/dot(v,v))*(v*transpose(v));
    Q=Q*P;
end
end
```

```
function [R] = compR(B,D,p,k)

R=triu(B(:,p));
R=full(spdiags(D,0,R));
%R=R(:,pt);


end
```

4 Tests

4.1 QRFact

Das Skript

```
A1 = generateRandMatrix(2,2,2)

[B1,D1,p1,k1] = QRFact(A1)


A2 = generateRandMatrix(10,5,7)

[B2,D2,p2,k2] = QRFact(A2)


A3 = generateRandMatrix(1000,100,75)

[B3,D3,p3,k3] = QRFact(A3)
```

erstellt zufällige Matrizen ausgewählter Größen, und berechnet eine QR-Faktorisierung davon, in output.txt ist der Output davon zu finden.

4.2 QRSolve

Mit dem Skript

```
m = randi(100)
n = randi(m)
A = generateRandMatrix(m,n,m);
b = rand(m,1);
[B,D,p,k] = QRFact(A);
xqr = QRSolve(B,D,p,k,b)
x = linsolve(A,b)

err = max(abs(xqr - x))
relErr  = max(abs(xqr - x)./x)
```

lässt sich die Abweichung von *linsolve* und *QRSolve* berechnen. Die absolute Abweichung liegt meistens in der Größenordnung $10^{-13} - 10^{-15}$, die relative in der Größenordnung 10^{-13}

4.3 compQ/compR

Durch Ausführen des Skripts:

```
A = generateRandMatrix(randi(100),randi(100),randi(100));
size(A)
[B,D,p,k] = QRFact(A);

Q = compQ(B,p,k);
R = compR(B,p,k);
[Q_m,R_m] = qr(A);

norm(Q*R-A(:,p))/norm(A)
norm(Q_m*R_m-A(:,p))/norm(A)
```

lässt sich der relative Fehler der beiden QR-Zerlegungen von zufälligen Matrizen abschätzen. Es ergibt sich, dass die beiden Fehler in der selben Größenordnung sind, meist ist sogar unsere Methode stabiler.

Ein paar Werte:

$A \in \mathbb{R}^{70\times 79}$, rel. Fehler QRFact: 1.0926, rel. Fehler qr: 1.5907

$A \in \mathbb{R}^{77\times 98}$, rel. Fehler QRFact: 1.0002, rel. Fehler qr: 0.5608

$A \in \mathbb{R}^{36\times 27}$, rel. Fehler QRFact: 1.0656, rel. Fehler qr: 1.6987