Practicum NMB : Eigenwaardenproblemen

Jona Beysens & Arnout Devos

vrijdag 25 apil 2014

Opgave 1

Kies $Q^{(0)} \in \mathbb{R}^{m \times d}$ met orthonormale kolommen.
for $k = 1, 2, ...$ **do**
$AZ = Q^{(k-1)}$
$Q^{(k)}R^{(k)} = Z$
$A^{(k)} = {Q^{(k)}}^T A Q^{(k)}$
end for
$x = diag(A^{(k)})$

Dit algoritme is de gelijktijdige inverse iteratie voor het berekenen van de d kleinste eigenwaarden en bijhorende eigenvectoren van de matrix A. Het is een aangepaste versie van de gelijktijdige iteratie waarbij gebruik gemaakt wordt van de inverse van A. De inverse wordt echter nooit expliciet berekend maar er wordt telkens een stelsel opgelost naar Z. De eigenwaarden van A^{-1} zijn de inversen van de eigenwaarden van A en de eigenvector horende bij λ_i is dezelfde als die horende bij $\frac{1}{\lambda_i}$:

Bewijs.

$$Ax = \lambda x \tag{1}$$
$$A^{-1}Ax = \lambda A^{-1}x \tag{2}$$
$$\frac{1}{\lambda} = A^{-1}x \tag{3}$$

□

De eigenvectoren horende bij de kleinste eigenwaarden van A komen nu in de eerste kolommen van Q te staan. Daardoor zal $A^{(k)}$ convergeren naar een diagonaalmatrix waarvan de eerste diagonaalelementen de kleinste eigenwaarden van A zijn.

Opgave 2

a) Als μ een eigenwaarde van A goed benadert, worden de grootste eigenwaarde van $(A - \mu I)^{-1}$ en het conditiegetal van $(A - \mu I)$ zeer groot. Daardoor zal bij het aanleggen van een perturbatie op de matrix $(A - \mu I)$ de oplossing $\hat{y}$ van het stelsel sterk afwijken van het originele probleem. Maar omdat de grootste eigenwaarde zo dominant is, ligt de oplossing van het geperturbeerd

probleem $\hat{y}$ bijna in dezelfde richting als y. De grootte van $\hat{y}$ verschilt echter sterk van deze van y. Door te normaliseren verschillen $\hat{y}$ en y enkel nog in richting. Dit verschil is miniem.

b) Om het residu van $Ax - \rho x$ te minimaliseren lossen we het kleinste kwadratenprobleem op:

Bewijs.

$$\min_{\rho \in \mathbb{R}} \|Ax - \rho x\|_2 \Leftrightarrow x^T(Ax - x\rho) = 0 \tag{4}$$

$$x^T Ax = x^T \rho x \tag{5}$$

$$\rho = \frac{x^T Ax}{x^T x} \tag{6}$$

□

Deze oplossing voor ρ komt overeen met het Ragleigh quotiënt $r(x)$.

Opgave 3

a) **JONA**

b) **JONA** Arnout oplossing: Om, gegeven een symmetrische matrix $A \in \mathbb{R}^{m \times m}$, de eigenwaarde te berekenen die het dichtst bij een getal α gelegen is, kunnen we gebruik maken van *inverse iteratie*. De convergentie is lineair en zal veel sneller gebeuren naarmate α dichter bij de gewilde eigenwaarde gelegen is. De rekenkost is $\mathcal{O}(m^3)$ doordat er een stelsel moet worden opgelost bij het inverteren van de matrix $A - \alpha I$.

De oplossing van opgave 2. Zie figuur 1.

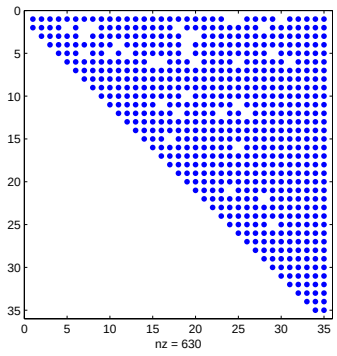
Opgave 4

In figuur 1 zien we de structuur van de matrix *mat1.txt* na reductie naar de Hessenberg vorm. Doordat de matrix niet perfect symmetrisch is, staan er nog waarden in de bovendriehoek die verschillend zijn van 0. Als we kijken naar de grootte van de waarden in die bovendriehoek dan zijn deze allemaal klein. Als we alle waarden die in absolute waarde kleiner dan 10^{-14} zijn gelijk aan 0 stellen dan verkrijgen we de vorm in figuur 2. Deze tridiagonale vorm is typisch voor reductie van een symmetrische matrix naar een Hessenberg vorm.

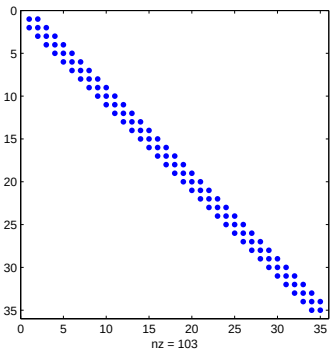
/////oplossing2/////

Indien de matrix niet gereduceerd zou worden naar Hessenberg vorm, moet in elke iteratie van het QR algoritme de factorisatie berekend worden van een volle matrix. Dit vergt $\mathcal{O}(m^3)$ flops. De convergentie naar machinenauwkeurigheid ϵ_{mach} gebeurt meestal in $\mathcal{O}(m)$ stappen wat het totale vereiste werk op $\mathcal{O}(m^4)$ flops brengt. Het rekenwerk van de QR factorisatie van een matrix in Hessenbergvorm bedraagt $\mathcal{O}(m^2)$ flops. Hierdoor komt het totale werk op $\mathcal{O}(m^3)$ flops. Aangezien de Hessenbergvorm van de matrix *mat*1.*txt* een tridiagonale vorm heeft zal er zelfs nog minder werk nodig zijn. Het rekenwerk voor een

QR factorisatie van een tridiagonale matrix is immers $\mathcal{O}(m)$ flops waarmee het totale rekenwerk op $\mathcal{O}(m^2)$ flops komt.Test



Figuur 1: Vorm van de matrix *mat1.txt* door reductie naar Hessenberg vorm

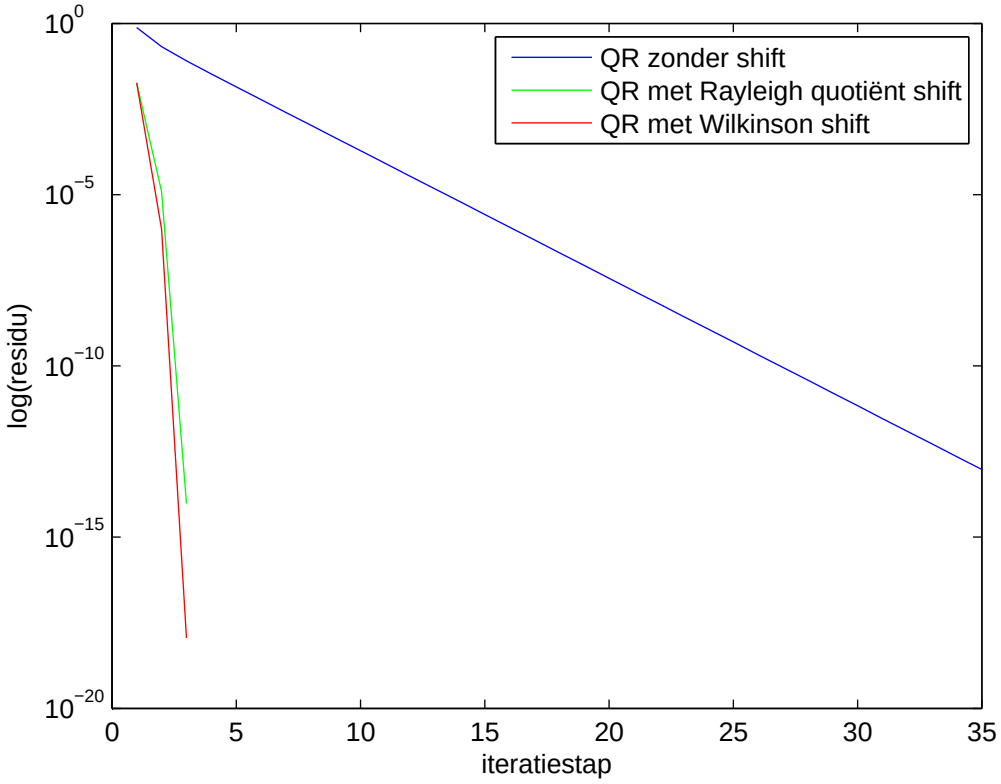


Figuur 2: Tridiagonale vorm van de matrix *mat1.txt* na reductie naar Hessenberg vorm en verwaarlozing van elementen $< 10^{-13}$

Indien de matrix niet gereduceerd zou worden naar Hessenberg vorm, moet in elke iteratie van het QR algoritme de factorisatie berekend worden van een volle matrix. Dit vergt $\mathcal{O}(m^3)$ flops. De convergentie naar machinenauwkeurigheid ϵ_{mach} gebeurt meestal in $\mathcal{O}(m)$ stappen wat het totale vereiste werk op $\mathcal{O}(m^4)$ flops brengt. Het rekenwerk van de QR factorisatie van een matrix in Hessenbergvorm bedraagt $\mathcal{O}(m^2)$ flops. Hierdoor komt het totale werk op $\mathcal{O}(m^3)$ flops. Aangezien de Hessenbergvorm van de matrix $mat1.txt$ een tridiagonale vorm heeft zal er zelfs nog minder werk nodig zijn. Het rekenwerk voor een QR factorisatie van een tridiagonale matrix is immers $\mathcal{O}(m)$ flops waarmee het totale rekenwerk op $\mathcal{O}(m^2)$ flops komt.

Opgave 5

Op figuur 3 kunnen we duidelijk zien dat de *QR-methode zonder shift* lineair convergeert wat overeenkomt met de theorie.



Figuur 3: Convergentie van de verschillende QR methodes: zonder shift, Rayleigh quotiënt shift en Wilkinson shift op logaritmische schaal.

In tabel 1 zien we dat voor de matrix mat1 er voor de *QR iteratie met Rayleigh quotiënt shift* en de *QR iteratie met Wilkinson shift* er eerst lineaire convergentie optreedt. Wanneer bij deze algoritmes het residu voldoende klein is geworden, verloopt de convergentie meer en meer kubisch. Dit komt overeen met de theorie die zegt dat $q_m^{(k)}$ kubisch convergeert tot een eigenvector. Daardoo convergeert ook het bijbehorende Rayleigh quotiënt $r(q_m^{(k)}) = A_{mm}^{(k)}$ kubisch. Naarmate het aantal iteraties toeneemt wordt de invloed van de kubische convergentie steeds groter omdat het aantal elementen dat nog moet convergeren naar een eigenwaarde in elke iteratiestap kleiner wordt.

Opgave 6

Het verband tussen het *QR-algoritme met Rayleigh quotiënt shift* en de *Rayleigh quotiënt iteratie*, is dat de veronderstelde waarden voor de eigenvector en eigenwaarde bij het QR-algoritme, μ en $q_m^{(k)}$, dezelfde zijn als de waarden die berekend worden bij de Rayleigh quotiënt iteratie met startvector e_m. Er wordt dus een Rayleigh quotiënt iteratie toegepast op de laatste eigenwaarde van A. De Rayleigh quotiënt iteratie convergeert kubisch, dus de convergentie van het

stap	zonder shift	Rayleigh quotiënt shift	Wilkinson shift
1	0.763590501238868	0.0183598152348892	0.0183598152348892
2	0.210113423731095	1.23415266649239e-05	9.83319921427331e-07
3	0.0808375414586785	9.55801949807706e-15	1.13833614396543e-18
⋮	⋮		
10	0.000192468492953297		
11	8.16489243297394e-05		
12	3.46381851762602e-05		
13	1.46948586913677e-05		
14	6.23416040757481e-06		
15	2.64479155115201e-06		
16	1.12203213434041e-06		
⋮	⋮		
32	1.23546560557316e-12		
33	5.24136923576435e-13		
34	2.22361119093024e-13		
35	9.43350202212816e-14		

Tabel 1: Convergentie van het residu bij de berekening van eigenwaarden volgen verschillende methodes.

QR algoritme met Rayleigh shift moet ook kubisch zijn. De Rayleigh shift heeft geen invloed op de andere eigenwaarden, dus er wordt verwacht dat deze lineair convergeren net als bij gelijktijdige iteratie het geval is. //////EXTRA INFO HIER NOG + VB FIGUUR/////

Opgave 7

ARNOUT

Opgave 9

Dit is de code van matlab:

```
function [ V,D ] = opgave_9( A,tol )
V = eye(size(A,1));
D = A;
s = sort(eig(A),'descend');
ready = false;
i=0;
error = zeros(50,1);
while ~ready
    i = i + 1;
    for p=1:(size(A,1)-1)
        for q = (p+1):size(A,1)
            theta = (1/2)*atan(2*D(p,q)/(D(q,q)-D(p,p)));
            J = eye(size(A,1));
            J(p,p) = cos(theta);
            J(p,q) = sin(theta);
            J(q,p) = -sin(theta);
```

```
            J(q,q) = cos(theta);
            D = J'*D*J;
            V = V * J;
        end
    end
    d = sort(diag(D),'descend');
    [residu,index] = max(abs(d-s));
    residu_rel = residu/norm(s(index))
    error(i) = residu_rel;
    if(residu_rel < tol)
        ready = true;
    end
end
semilogy(error);
steps = i


end
```