Numerieke Modellering en Benadering: Practicum 1

Ellen Anthonissen Marte Biesmans

vrijdag 21 april 2016

Opgave 1

De Householder transformatiematrix

$$F = I - 2\frac{vv^*}{v^*v}$$

heeft als eigenwaarden -1 en 1 en als eigenvectoren respectievelijk v en w met $w \perp v$. Deze resultaten zijn als volgt bekomen: De Householder transformatiematrix F is symmetrisch

$$F^* = (I - 2\frac{vv^*}{v^*v})^* = I - 2\frac{vv^*}{v^*v} = F$$

en unitair

$$F^*F = FF^* = (I - 2\frac{vv^*}{v^*v})(I - 2\frac{vv^*}{v^*v}) = I - 4\frac{vv^*}{v^*v} + 4\frac{v(v^*v)v^*}{(v^*v)^2} = I.$$

Omdat F unitair is, moeten de eigenwaarden van F op de complexe eenheidscirkel gelegen zijn. Omdat F reëel en symmetrisch is, zijn de eigenwaarden reële getallen. Hieruit volgt dat de eigenwaarden enkel ± 1 kunnen zijn.

Als we nu Fv uitrekenen, bekomen we

$$Fv = v - 2\frac{vv^*}{v^*v}v = -v.$$

Hieruit volgt dat v en eigenvector is bijhorende bij de eigenwaarde -1. Neem nu w met $w \perp v$ en we rekenen Fw uit, dan bekomen we

$$Fw = w - 2\frac{vv^*}{v^*v}w = w,$$

want $v^*w = 0$. Hieruit volgt das w een eigenvector is bijhorende bij de eigenwaarde 1.

Geometrisch gezien komt dit overeen met een spiegeling over de w-as. Neem een vector a en ontbind die in een compontent volgens de v-as en een component volgens de w-as. De component volgens de v-as zal vermenigvuldigd worden met -1 en die volgens de w-as met 1. Zo bekomen we een spiegeling rond de w-as.

Opgave 2

De functie Householder_explicit wordt weergegeven in onderstaande MATLAB-code. Deze methode genereert de matrix Q door na de impliciete methode een nieuwe loop uit te voeren. Deze loop is gebaseerd op de functie om het product Qx te berekenen, met voor x telkens een eenheidsvector.

Na het expliciet berekenen van de QR-factorisatie van A, wordt de oplossing van het stelsel $Ax = b$ bekomen door $y = Q^*b$ en $x = R\backslash y$ te berekenen.

```
function [Q,R] = Householder_explicit(A)
[m,n] = size(A);
R = A;
V = zeros(m,n);
for k = 1:n
    v=zeros(m,1);
    v(k:m)=R(k:m,k);
    v(k)=v(k)+norm(v);
    norm_v = norm(v,2);
    if norm_v ~= 0
        v = v./norm(v,2);
    end
    V(1:m,k) = v;
    R(k:m,k:n) = R(k:m,k:n) - 2*v(k:m)*(v(k:m)'*R(k:m,k:n));
end

I = eye(m,m);
Q = zeros(m,m);
for l = 1:m
    x = I(1:m,l);
    for j = n:-1:1
        x(j:m) = x(j:m)-2*V(j:m,j)*(V(j:m,j)' * x(j:m));
    end
    Q(:,l) = x;
end
end
```

De functies Householder_implicit en Apply_Q zijn Hier onder weergegeven in MATLAB-code. Na het berekenen van L en R en L toe te passen op b (noem deze vector y), wordt de oplossing van het stelsel $Ax = b$ bekomen door $x = R\backslash y$ te berekenen.

```
function [L,R] = Householder_implicit(A)
[m,n] = size(A);
R = A;
L = zeros(m,n);
for k = 1:n
    v=zeros(m,1);
    v(k:m)=R(k:m,k);
    v(k)=v(k)+norm(v);
    norm_v = norm(v,2);
    if norm_v ~= 0
```

```
        v = v./norm(v,2);
    end
    L(1:m,k) = v;
    R(k:m,k:n) = R(k:m,k:n) - 2*v(k:m)*(v(k:m)'*R(k:m,k:n));
end
end
```

```
function [y] = Apply_Q(L,b)
[m,n] = size(L);
y = zeros(n,1);
y(1:m,1) = b(1:m,1);
for k = 1:n
    y(k:m) = y(k:m) - 2 * L(k:m,k) * ( L(k:m,k)' * y(k:m) );
end
end
```

De tijd om het stelsel $Ax = b$ op te lossen met de expliciete en de impliciete methode worden respectievelijk weergegeven in tabel 1 en tabel 2. Hier zien we dat de tijd om het stelsel op te lossen een orde-grootte kleiner is met de impliciete methode dan met de expliciete methode voor een grotere waarde van n. Voor kleine n is er geen verschil in uitvoeringstijd.

De ordegroottes van de relatieve fout $\|\delta x\|/\|x\|$ van de oplossing voor de expliciete en de impliciete methode worden respectievelijk weergegeven in tabel 3 en tabel 4. De ordegroottes van de verhouding van de norm van het residu op de norm van de b-vector $\|r\|/\|b\|$ van de oplossing voor de expliciete en de impliciete methode worden respectievelijk weergegeven in tabel 5 en 6. Deze waarden vullen we nu in in de vergelijking

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\frac{\|r\|}{\|b\|}.$$

We zien hier dat het linkerlid van de ongelijkheid veel kleiner is dan het rechterlid naarmate κ groter wordt. Voor $\kappa = 1$ zien we namelijk dat het linkerlid ongeveer gelijk is aan het rechterlid. Voor $\kappa = 10^8$ is er veel meer verschil tussen de twee leden. We zien zelfs dat wanneer $\kappa = 1$, de ongelijkheid niet meer helemaal klopt. Dit komt omdat de machine-precisie dan de beperkende factor wordt. We zien ook voor beide methodes dezelfde getallen, dus de mate van achterwaartse stabiliteit van de twee methodes is ongeveer dezelfde.

n \ κ	1	10^4	10^8
10	0,0044	0,0022	0,0025
100	0,1883	0,1403	0,1306
1000	28,2691	27,725	27,9583

Tabel 1: De snelheid van de expliciete methode

n \ κ	1	10^4	10^8
10	0,0050	0,0015	0,0013
100	0,0115	0,0131	0,0163
1000	7,9605	7,7967	7,8586

Tabel 2: De snelheid van de impliciete methode

n \ κ	1	10^4	10^8
10	10^{-16}	10^{-13}	10^{-9}
100	10^{-15}	10^{-13}	10^{-9}
1000	10^{-14}	10^{-13}	10^{-9}

Tabel 3: De ordegrootte van $\|\delta x\|/\|x\|$ van de expliciete methode

n \ κ	1	10^4	10^8
10	10^{-16}	10^{-13}	10^{-9}
100	10^{-15}	10^{-13}	10^{-9}
1000	10^{-15}	10^{-13}	10^{-9}

Tabel 4: De ordegrootte van $\|\delta x\|/\|x\|$ van de impliciete methode

n \ κ	1	10^4	10^8
10	10^{-16}	10^{-13}	10^{-9}
100	10^{-15}	10^{-13}	10^{-9}
1000	10^{-14}	10^{-13}	10^{-10}

Tabel 5: De ordegrootte van $\|r\|/\|b\|$ van de expliciete methode

n \ κ	1	10^4	10^8
10	10^{-16}	10^{-13}	10^{-9}
100	10^{-15}	10^{-13}	10^{-10}
1000	10^{-14}	10^{-13}	10^{-10}

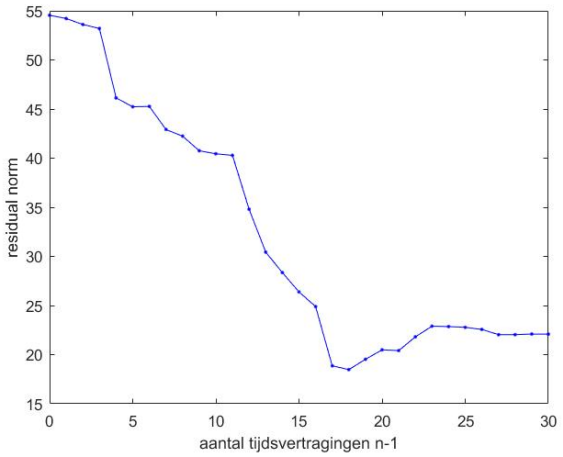
Tabel 6: De ordegrootte van $\|r\|/\|b\|$ van de impliciete methode

Opgave 3

Het kleinste kwadraten probleem $\|b - Ax\|_2$ ziet er uit als volgt:

$$A = \begin{bmatrix} u_n & u_{n-1} & \cdots & u_1 \\ u_{n+1} & u_n & \cdots & u_2 \\ \vdots & \vdots & \ddots & \vdots \\ u_m & u_{m-1} & \cdots & u_{m-n+1} \end{bmatrix}, b = \begin{bmatrix} y_n \\ y_{n+1} \\ \vdots \\ y_m \end{bmatrix},$$

met u_n de n-de waarde uit de vector van de ingangen uit `model.mat`. Dit stelsel $Ax = b$ lossen we op naar x met behulp van een van de algoritmes uit de vorige opgave. Om dit model te valideren stellen we op analoge wijze een matrix B op met de ingangen uit `validate.mat`. We vergelijken de verschillende modellen met behulp van de waarde $\|y - Bx\|_2$ met y de uitgangen uit `validate.mat`. Voor verschillende tijdsvertragingen levert dit de waardes uit figuur 1. Hieruit blijkt dat 18 tijdsvertragingen het meest interessant is, of $n = 19$. Hier bereikt de grafiek namelijk een eerste minimum en meer tijdsvertragingen (en dus meer rekenwerk) lijken het residu niet kleiner te kunnen maken dan bij $n = 19$.



Figuur 1: Validatie van het model bij verschillende tijdsvertragingen

Opgave 4

Neem een vector $x \in \mathbb{R}^n$. Schrijf x als lineaire combinatie van de eigenvectoren van A $q_1, q_2 \ldots q_n$ met bijhorende eigenwaarden $\lambda_1, \lambda_2 \ldots \lambda_n$:

$$x = \sum_{j=1}^{n} a_j q_j,$$

dan is het Rayleigh quotiënt van x:

$$r(x) = \frac{\sum_{j=1}^{n} a_j^2 q_j \lambda_j}{\sum_{j=1}^{n} a_j^2}.$$

Het Rayleigh quotiënt is onafhankelijk van de schaal van x, dus stel $\|a\| = \|[\,a_1\ a_2\ \ldots\ a_n\,]^T\| = 1$, dan is $\sum_{j=1}^{n} a_j^2 = 1$. Dan wordt het Rayleigh quotiënt gelijk aan

$$r(x) = \sum_{j=1}^{n} a_j^2 q_j \lambda_j.$$

Stel nu dat $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, dan is het Rayleigh quotiënt maxiaal voor $a = e_1$ met de waarde λ_{max} en minimaal voor $a = e_n$ met de waarde λ_{min}. Dus het Rayleigh quotiënt bevint zich in het interval $[\,\lambda_{min},\lambda_{max}\,]$.

Ook is het Rayleigh quotiënt een continu voor $a \neq 0$, dus elke waarde tussen λ_{min} en λ_{max} wordt bereikt voor een bepaalde vector a, dus ook voor een bepaalde vector x.

Opgave 5

Het QR-algoritme zonder shifts is equivalent aan gelijktijdige iteratie toegepast op de eenheidsmatrix. Het QR-algoritme met of zonder shifts berekent alle eigenwaarden. Met de Rayleigh quotient iteratie kan men 1 eigenwaarde en bijhorende eigenvector bepalen. Bij gelijktijdige iteratie worden de vectoren bepaald die de eigenruimte opspannen, maar geen eigenwaarden.

Als de shifts gebruikt in het QR-algoritme goede benaderingen zijn voor een eigenwaarde, zal men sneller convergeren naar een eigenvector. Het Rayleigh quotient vormt een goede schatting van de eigenwaarde. Gebruiken we deze waarde in elke iteratie, dan bekomt men dezelfde schatting voor eigenwaarde en eigenvector als bij Rayleigh quotient iteratie die vertrekt vanuit een eenheidsvector. Past men dit toe op elke eenheidsvector en dus op een eenheidsmatrix krijgt men een combinatie van gelijktijdige iteratie en Rayleigh quotient iteratie. Het convergentiegedrag van het QR-algoritme met Rayleigh shift kan dus gezien worden als een combinatie van het convergentiegedrag van gelijktijdige iteratie en Rayleigh quotient iteratie. Via gelijktijdige iteratie bekomt men de eigenruimte en via Rayleigh quotient iteratie gaat men de schatting van eigenvector gebruiken als startvector in de Rayleigh quotient iteratie om een nieuwe schatting voor eigenwaarde en -vector te bekomen.

Opgave 6

We maken gebruik van de ongelijkheid

$$\frac{\|e_n\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n.$$

Voor $n = 10$ wordt dit

$$\frac{\|e_{10}\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{10}.$$

Gebruik makende van de gegevens $\|e_0\|_A = 1$ en $\|e_{10}\|_A = 2 \times 2^{-10}$, bekomen we

$$9 \leq \kappa.$$

We vinden dus een ondergrens voor κ.

Voor $n = 20$ wordt de ongelijkheid

$$\frac{\|e_{20}\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{20}.$$

Gebruik makende van het gegeven $\|e_0\|_A = 1$ en het berekende $9 \leq \kappa$, bekomen we

$$\|e_{20}\|_A \leq 2 \times 2^{-20}.$$

We vinden dus een bovengrens voor $\|e_{20}\|_A$.

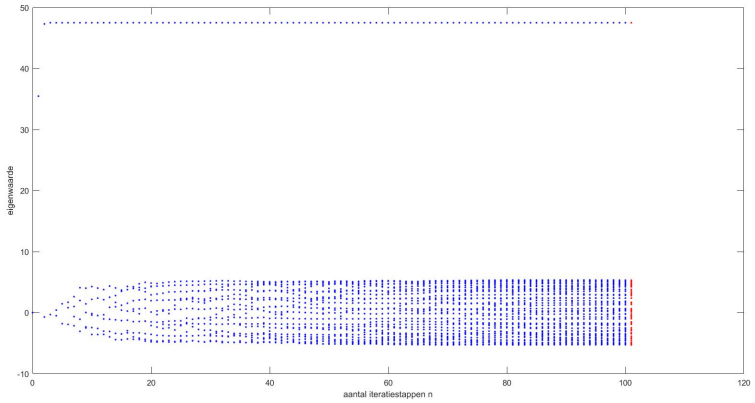
Opgave 7

Om de Ritz waarden op te slaan was er een kleine aanpassing nodig in het gegeven algoritme. In elke iteratiestap wordt een kolom van de matrix 'Ritz' aangevuld met het aantal eigenwaarden op de eerste rij en de berekende eigenwaarden op de rijen hieronder. Met behulp van volgende matlab-code stellen we een figuur op die de reële componenten van de Ritz waarden voorstelt in functie van de iteratiestap n:

```
clear;
A = sprand(1000,1000,0.1);
b = rand([1000,1]);
maxit = 100;
k=100;
[H, Q, Ritz] = arnoldi(A, b, maxit);

E = eigs(A,k);
X = zeros(1,maxit);
for it=1:maxit
    n=Ritz(1,it);
    for i = 1:n
       X_size = size(X,2);
       X(1, X_size+1) = it;
       Y(1, X_size+1) = real(Ritz(i+1,it));
    end
end
X_E = zeros(1,k)+101;
Y_E = real(E');
plot(X,Y, 'b.',X_E,Y_E,'r.');
xlabel('aantal iteratiestappen n');
ylabel('eigenwaarde');
```

Dit levert het resultaat uit figuur 2. Hierbij stellen de rode waardes rechts de werkelijke eigenvectoren voor en de blauwe de berekende Ritz waarden. Het valt op dat na 100 iteratiestappen en zelfs eerder («1000, grootte van de matrix) al een goede benadering van de eigenwaarden bereikt wordt.



Figuur 2: de Ritz waarden bij verschillende iteratiestappen

Opgave 8

Het GMRES algoritme werd geïmplementeerd als volgt:

```
function [x, itx] = NMB_gmres(A,b,maxit)

% function [x, itx, res, dx] = NMB_gmres(A,b,maxit)
%
% GMRES algoritme om het stelsel Ax=b op te lossen
%
% invoer:
% A     – matrix A uit Ax=b
% b     – vector b uit Ax=b
% maxit – maximum aantal iteratiestappen
%
% uitvoer:
% x     – oplossing van het stelsel Ax=b
% itx   – vector met x doorheen de iteraties

Q(:,1) = b/norm(b);
itx = zeros(maxit,maxit);
m = size(A,1);

for n=1:maxit
  v = A*Q(:,n);
  for j = 1:n
    H(j,n) = Q(:,j)'*v;
    v = v – H(j,n)*Q(:,j);
  end

  [Q_H, R_H] = qr(H);
  e1 = zeros(n,1);
  e1(1,1) = norm(b);
```

```
30    y = R_H\(Q_H'*e1);
31
32    itx(1:m,n) = Q*y;
33
34    H(n+1,n) = norm(v);
35    if H(n+1,n) <= 0
36        break;
37    end
38    Q(:,n+1) = v/H(n+1,n);
39
40  end
41  x=itx(:,maxit);
42  end
```

Dit geeft het convergentiegedrag zoals weergegeven in figuur 3 voor verschillende waardes voor `alpha`. Hoe groter `alpha`, hoe sneller de convergentie. Voor `alpha` gelijk aan 1 is er geen convergentie.

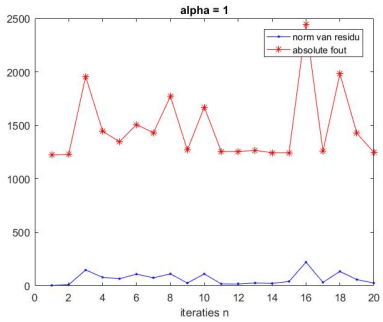
Opgave 9

De interlace eigenschap van een tridiagonale, symmetrische en reële matrix A luidt als volgt. Voor $A \in \mathbb{R}^{n\times n}$, en $A^{(1)}, A^{(2)} \dots A^{(n)}$ de principale vierkante submatrices van dimensie $1, 2 \dots n$, geldt dat de eigenwaarden van deze submatrices interlacen. Dit wil zeggen dat $\lambda_j^{(k+1)} \leq \lambda_j^{(k)} \leq \lambda_{j+1}^{(k+1)}$. Als A irreduceerbaar is (en dus geen nul heeft op een nevendiagonaal), dan worden de ongelijkheden strikte ongelijkheden. Dit laatste is belangrijk voor de bisectie-methode.

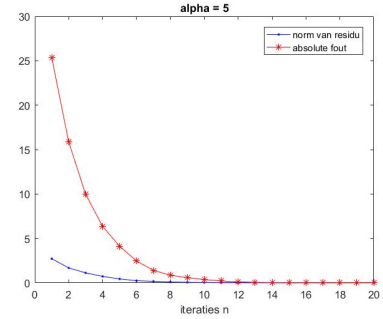
Als voorbeeld nemen we de tridiagonale, symmetrische en reële matrix

$$A = \begin{bmatrix} 1 & 5 & 0 & 0 \\ 5 & 2 & 6 & 0 \\ 0 & 6 & 3 & 7 \\ 0 & 0 & 7 & 4 \end{bmatrix}.$$

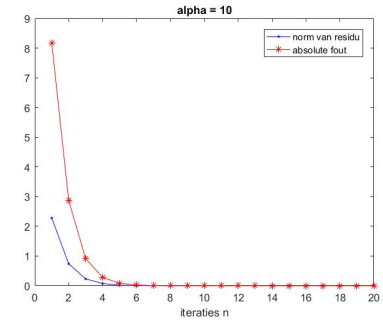
We zien duidelijk op figuur 4 dat de eigenwaarden van een principale submatrix tussen de eigenwaarden van de principale submatrix van een dimensie groter liggen.



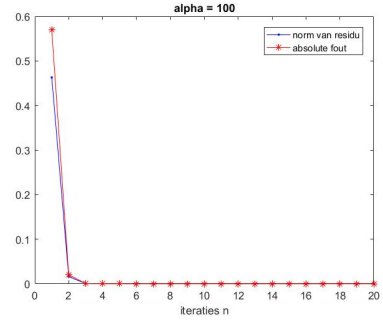
(a) alpha = 1



(b) alpha = 5

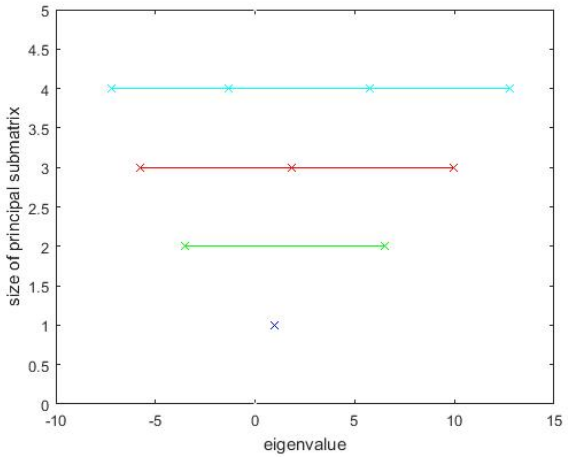


(c) alpha = 10



(d) alpha = 100

Figuur 3: convergentiegedrag voor verschillende waardes voor alpha



Figuur 4: De eigenwaarden van de opeenvolgende principale submatrices van A

Opgave 10

Een algoritme in pseudo-code dat alle eigenwaarden van een symmetrische, tridiagonale en reële matrix in het interval [a,b) berekent tot op een bepaalde tolerantie met behulp van de bisectie-methode, is te vinden in algoritme 1.

Algorithm 1 Bisectie-methode

queue = [[a,b)]
while queue niet leeg **do**
    Haal eerste interval uit de queue
    $Sturm_{links}$ = #tekenwisselingen in de Sturm-rij van A-linkergrens*I
    $Sturm_{rechts}$ = #tekenwisselingen in de Sturm-rij van A-rechtergrens*I
    #eigenwaarden = $Sturm_{rechts}$ - $Sturm_{links}$
    midden = 0.5*(rechtergrens - linkergrens)
    if #eigenwaarden == 1 **then**
        if midden < tolerantie **then**
            Voeg linkergrens+midden toe als eigenwaarde
        else
            Voeg [linkergrens,linkergrens+midden] en [linkergrens+midden,rechtergrens] vooraan bij in de queue
        end if
    else if #eigenwaarden > 1 **then**
        Voeg [linkergrens,linkergrens+midden] en [linkergrens+midden,rechtergrens] vooraan bij in de queue
    end if
end while

Deze methode werd uitgeschreven in MATLAB (zie onderstaande code) en werd toegepast op enkele symmetrische, tridiagonale en reële matrices.

```
1 function E = bisection(A,a,b,tol)
```

```
queue = [a,b];
E = [];
while size(queue,1)~=0
    l = queue(1,1);
    r = queue(1,2);
    queue = queue(2:end,:);
    [~,s_l] = sturm(A,l);
    [~,s_r] = sturm(A,r);
    nb_ev = s_r - s_l;
    mid=(r-l)/2;
    if nb_ev == 1
        if mid < tol
            E = [E;l+mid];
        else
            queue = [l l+mid;l+mid r;queue];
        end
    elseif nb_ev > 1
        queue = [l l+mid;l+mid r;queue];
    end
end
end
```

Als eerste voorbeeld nemen we de tridiagonale, symmetrische en reële matrix $A \in \mathbb{R}^{n \times n}$ met $n = 4$.

$$A = \begin{bmatrix} 1 & 5 & 0 & 0 \\ 5 & 2 & 6 & 0 \\ 0 & 6 & 3 & 7 \\ 0 & 0 & 7 & 4 \end{bmatrix}$$

Als we de eigenwaarden van deze matrix willen berekenen in het interval [-2,6) tot op een tolerantie 10^{-2}, dan vinden we we eigenwaarden [-1,3359375; 5,7734375]. Deze komen overeen met de 'exacte' eigenwaarden [-1,32970777874292; 5,77851991186833] berekend met de functie eig(A) van MATLAB. Als we beide resultaten afronden tot op twee cijfers na de komma, zien we inderdaad dat de eigenwaarden gevonden zijn tot op de gegeven absolute fout.

Een interessant testprobleem is een random symmetrische, tridiagonale en reële matrix $B \in \mathbb{R}^{n \times n}$ MATLAB met $n = 100$. De diagonalen zijn berekend met de functie rand(n) en rand($n - 1$). We zoeken naar de eigenwaarden in het interval [0,1) tot op een tolerantie 10^{-2}. De matrix waarmee dit getest is, wordt hier niet weergegeven omdat deze veel te groot is. Het interessante aan deze matrix is dat de eigenwaarden die tussen 0 en 1 gelegen zijn, meestal dichter bij elkaar liggen dan de opgegeven tolerantie. Het algoritme dat hierboven beschreven wordt, kan toch nog steeds onderscheid maken tussen de verschillende eigenwaarden. Zo vindt het bijvoorbeeld als eerste twee eigenwaarden 0,03515625 en 0,04296875 voor de eigenwaarden (berekend met de eig() functie van MATLAB) 0,0340437309188287 en 0,0441523470170203.

Een volgend interessant testprobleem wordt bekomen door eigenwaarden als grens te nemen. Theoretisch gezien mag de eigenwaarde die de rechtergrens voorstelt niet gevonden worden. Dit is echter soms toch het geval (zeker voor grote matrices). Dit komt doordat de elementen van de Sturm-rij opgesteld worden aan de hand van vorige elementen. Zo kunnen reken- en afrondingsfouten

zich voortplanten.

Ook is er tijdens het testen opgevallen dat de gevraagde nauwkeurigheid niet altijd bereikt wordt voor alle eigenwaarden. Dit komt doordat wanneer de grens van een te onderzoeken interval dicht bij een eigenwaarde komt, de Sturm-rij van $A - xI$ slecht geconditioneerd is. Het laatste element ligt dan rond 0, waardoor er snel een tekenwissel te veel of te weinig is.