# Numerieke Modellering en Benadering
# Practicum 1: Eigenwaardenproblemen

Martijn Boussé
Dario Incalza
*Faculteit Ingenieurswetenschappen*
*Katholieke Universiteit Leuven*

woensdag 24 april 2013

# Inhoudsopgave

# Lijst van figuren

# Inleiding

In het kader van het vak Numerieke Modellering en Benadering worden in dit practicum methoden voor het bepalen van eigenwaarden van volle matrices onderzocht. In een eerste gedeelte worden enkele theoretische eigenschappen van de methoden beschouwd. Vervolgens worden de convergentie-eigenschappen van deze methoden in het tweede deel nader onderzocht aan de hand van enkele MATLAB-experimenten. Ten slotte wordt in een derde gedeelte één van de alternatieve eigenwaardenalgoritmen, met name de bisectie-methode, in meer detail bekeken.

# Theoretische eigenschappen

## Opgave 1

**a)** Om alle eigenwaarden van een symmetrische matrix $A \in \mathbb{R}^{m\times m}$ te vinden gebruiken we het QR-algoritme. Om het algoritme praktisch bruikbaar te maken dienen er enkele aanpassingen te gebeuren aan het basis algoritme. Eén van de belangrijkste aanpassing is het gebruiken van shifts. Het volledig algoritme wordt gegeven als:

$(Q^{(0)})^T A^{(0)} Q^{(0)}$
**for** $k = 1, 2, ...$ **do**
  Kies een shift $\mu^{(k)}$
  $Q^{(k)} R^{(k)} = A^{(k-1)} - \mu^{(k)} I$
  $A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I$
  **if** Een niet-diagonaal element $A^{(k)}_{j,j+1}$ dicht genoeg bij nul is **then**
    kies $A_{j,j+1} = A_{j+1,j} = 0$ en zo bekom je,
    $\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} = A^{(k)}$
    Pas het QR-algoritme toe op $A_1$ en $A_2$
  **end if**
**end for**

Alvorens dit algoritme toegepast wordt, dient er opgemerkt te worden dat $A$ gereduceerd wordt tot een tridiagonale vorm. Door deze reductie toe te passen worden er nullen toegevoegd zowel in de rijen als in de kolommen, op deze manier worden overbodige arithmetische operaties vermeden. Dit bevordert aanzienlijk de rekenkost dat nodig is om het algoritme uit te voeren.

Men kan het Rayleigh quotiënt toe passen op de laatste kolom van $Q^{(k)}$, we krijgen dan:

$$\mu^{(k)} = \frac{(q_m^{(k)})^T A q_m^{(k)}}{(q_m^{(k)})^T q_m^{(k)}} = (q_m^{(k)})^T A q_m^{(k)} \tag{1}$$

Door dit getal, $\mu^{(k)}$, te kiezen als shift in elke stap zal $q_m^{(k)}$ kubisch gaan convergeren naar een eigenvector. Naast de *Rayleigh shift* kan men ook de *Wilkinson shift* toepassen. Het is éénvoudig aan te tonen dat *kubische convergentie* enkel in de, min of meer, algemene gevallen bereikt wordt. Er kan aangetoond worden dat de Wilkinson shift, minstens, *kubische convergentie*

bereikt in het slechtste geval. Beschouw matrix B waarbij Rayleigh shift niet zal werken en Wilkinson Shift wel,

$$B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2}$$

Het algoritme zal falen voor B, omdat de eigenwaarden van B symmetrisch rond de oorsprong liggen, $-1$ en 1. De gebruikte schatter kan dan geen richting kiezen om naar toe te convergeren. Men gebruikt dan de Wilkinson Shift om de symmetrie te breken, waardoor een convergentie naar, in dit geval, -1 gevonden wordt. Het aantal flops nodig door het QR-algoritme met shifts bedraagt, bij benadering, $\frac{4}{3}m^3$ flops.

**b)** Om de eigenwaarden te bepalen die het dichts gelegen zijn bij een bepaald getal $\beta$, van een symmetrische matrix $A \in \mathbb{R}^{m\times m}$, beschouwen we de *Rayleigh quotiënt iteratie*. De Rayleigh quotiënt van een vector $x \in \mathbb{R}^m$ is de scalar:

$$r(x) = \frac{x^T Ax}{x^T x} \tag{3}$$

We kunnen dit geometrisch interpreteren als, de eigenvectors van A zijn de stationaire punten van de functie $r(x)$ en eveneens zijn de eigenwaarden van A de waarden van $r(x)$ geëvalueerd in deze stationaire punten.

Nu we het Rayleigh quotiënt gedefinieërd hebben, kunnen we overgaan tot het *Rayleigh quotiënt iteratie algoritme*. Deze wordt gegeven als volgt:

$v^{(0)}$ =een bepaalde vector waarvan de lengte gelijk is aan 1
$\lambda^{(0)} = (v^{(0)})^T A v^{(0)}$, dit is het overeenkomende Rayleigh quotiënt
**for** $k = 1, 2, ...$ **do**
  Los $(A - \lambda^{(k-1)} I)w = v^{(k-1)}$ op naar $w$
  $v^{(k)} = \frac{w}{||w||}$
  $\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$
**end for**

Dit algoritme kan intuïtief beschouwd worden door te beredeneren dat we constant de schattingen van de eigenwaarden beter gaan proberen uit te voeren. Hierdoor zal de convergentie van de inverse iteratie stijgen in elke stap. Elke iteratie verdrievoudigd de nauwkeurigheid. Meer nog, er kan bewezen worden dat de convergentie kubisch is. Voor het bewijs verwijzen we naar theorema 27.3 op pagina 208 in het boek Numerical Linear Algebra. Het aantal flops nodig voor Rayleigh Quotient Iteratie bedraagt $O(m^3)$, indien we de matrix eerst in zijn tridiagonale vorm zetten spreken we nog over $O(m)$ flops.

## Opgave 2

**a)** Gegeven een matrix $A \in \mathbb{R}^{m\times m}$ en een vector $x \in \mathbb{R}^{m\times 1}$, waarbij $x$ een benadering is voor een eigenvector van $A$. Welke scalar $\rho$ gedraagt zich het *meeste* als een eigenwaarde voor $x$? Dit kan geformuleerd worden als een $m \times 1$ kleinste kwadraten probleem van de vorm

$x\rho \approx Ax$, waarbij $Ax$ het rechterlid voorstelt, $x$ de matrix en $\rho$ de onbekende vector. Beschouw de volgende doelfunctie:

$$f(\rho) = \frac{1}{2}||Ax - x\rho||_2^2 \tag{4}$$

en volgend optimalisatieprobleem zonder beperkingen:

$$\min_{\rho \in \mathbb{R}} \quad f(\rho) \tag{5}$$

Aangezien $f(\rho)$ convex is, wordt het minimum gevonden als een stationair punt van de doelfunctie:

$$\nabla f(\rho) = 0 \Leftrightarrow \frac{1}{2}\nabla \left(x^T A^T Ax - 2\rho x^T Ax + \rho^2 x^T x\right) = 0 \Leftrightarrow \frac{1}{2}\left(-2x^T Ax + 2\rho x^T x\right) = 0 \tag{6}$$

Dit levert exact de definitie van de Rayleigh quotiënt van $x$:

$$r(x) = \rho = \frac{x^T Ax}{x^T x} \tag{7}$$

**b)** De Rayleigh quotiënt methode lost in elke iteratie het stelsel $(A - \lambda^{(k-1)}I)w = v^{(k-1)}$ op. Dit stelsel wordt steeds meer en meer singulier naarmate de benaderende eigenwaarde $\lambda^{(k-1)}$ dichter in de buurt van de exacte eigenwaarde komt. Merk echter op dat de term $-\lambda^{(k-1)}I$ de gezochte eigenwaarde zeer klein maakt, waardoor bij het oplossen van het stelsel de richting zeer sterk zal benadrukt worden. De vector $w$ zal daardoor ook zeer groot worden, maar nog steeds in de juiste richting wijzen.

## Opgave 3

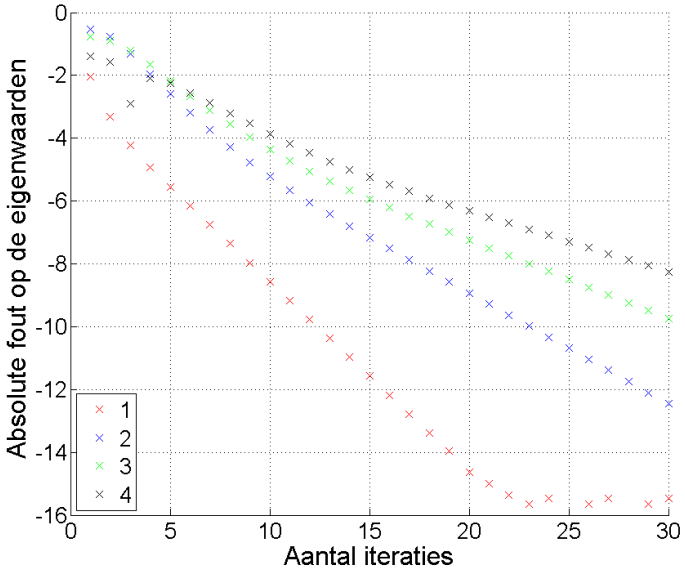
Gelijktijdige iteratie past de methode van de machten toe op meerdere kolommen tegelijk. Op die manier kunnen de $n$ grootste eigenwaarden van een matrix $A$ bepaald worden. Indien $A$ niet singulier is kunnen de $n$ kleinste eigenwaarden gevonden worden door de methode van de machten met de inverse van de matrix $A$ toe te passen. Beschouw de volgende pseudo-code:

Initialiseer $\hat{Q}^{(0)} \in \mathbb{R}^{m \times n}$ met $n$ orthonormale kolommen.
**for** $k = 1, 2, ...$ **do**
  $Z = A^{-1}\hat{Q}^{k-1}$
  $\hat{Q}^k \hat{R}^k = Z$
**end for**

Om in te zien dat dit algoritme werkt kunnen we analoog aan pagina 213 in Numerical Linear Algebra werken. Neem dezelfde veronderstellingen en symbolen aan, dan kan nu $v_j^k$ geschreven worden als:

$$v_j^k = a_{1j}\lambda_1^{-k}q_1 + ... + a_{mj}\lambda_m^{-k}q_m \tag{8}$$

Vergelijking 8 toont aan dat nu de kleinste eigenwaarden de grootste waarden worden en dat bijgevolg de $n$ kleinste eigenwaarden uit het algoritme volgen. De daaropvolgende afleidingen zijn volledig analoog. Figuur 1 toont een eenvoudig MATLAB-experiment, waarbij de vier kleinste eigenwaarden van een matrix $A$ met spectrum $\Lambda = \{1, 2, ..., 10\}$ worden bepaald.




**Figuur 1**: Convergentiegedrag van inverse gelijktijdige iteratie voor een eenvoudige matrix $A$ met spectrum $\Lambda = \{1, 2, ..., 10\}$. De vier kleinste eigenwaarden $\lambda = \{1, 2, 3, 4\}$ worden bepaald. De nauwkeurigheid op de kleinste eigenwaarde bereikt het eerst de machineprecisie.

# Convergentie-experimenten

In dit deel wordt een volle, reële, symmetrische matrix $A$ beschouwd. In MATLAB wordt daarvoor de matrix *mat1.txt* ingeladen met het commando *load*.

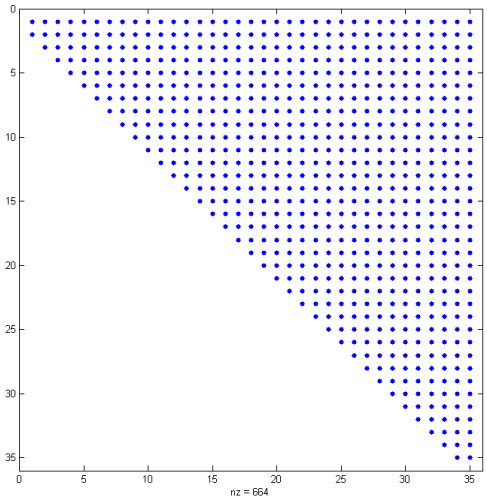
## Opgave 4

De gegeven matrix $A$ wordt gereduceerd naar Hessenberg vorm met behulp van het commando *Hess*. Aan de hand van het commando *spy* wordt in Figuur 2 de structuur van de bekomen matrix getoond. Merk echter op dat de resulterende matrix tridiagonaal is aangezien de originele matrix symmetrisch was. Inspecteer hiervoor de elementen van de matrix in MATLAB.

Methoden voor het bepalen van eigenwaarden zullen doorgaans werken in twee fasen. In een eerste fase gebruikt men een directe methode om de volle, symmetrische matrix $A \in \mathbb{R}^{m \times m}$ te structuren tot een tridiagonale matrix $H$. Deze directe reductie kost $\mathcal{O}(m^3)$ flops. In een tweede fase genereert men een oneindige sequentie van tridiagonale matrices die convergeren naar een diagonaalmatrix. Convergentie wordt meestal reeds bereikt in $\mathcal{O}(m)$ iteraties en per iteratie kost dit slechts $\mathcal{O}(m)$ flops. Bijgevolg kost deze tweede fase $\mathcal{O}(m^2)$ flops. Dit verklaart het belang van de eerste fase! Zonder de directe reductie wordt in de tweede fase telkens op een volle matrix ingewerkt, wat een verhoging van het aantal flops per iteratie inhoudt en bijgevolg het totaal aantal flops.

## Opgave 5

Wanneer we de drie methoden toe passen op de gegeven matrix, bekomen we Figuur 3. We zien hier duidelijk dat het QR-algoritme zonder shifts een lineaire convergentie heeft en de QR-algoritmen die Rayleigh of Wilkinson shifts gebruiken een kubische convergentie hebben. Alvorens de convergentie te onderzoeken lichten we even toe waarom we residus gebruiken. We




**Figuur 2**: Structuur van de reële, symmetrische matrix A na reductie. Inspecteer de elementen van de matrix om in te zien dat de structuur wel degelijk tridiagonaal is.

definiëren het residu als het element dat vlak boven een diagonaal element staat. Deze zal geleidelijk aan naar nul convergeren. We beschouwen de convergentie van deze residus om de convergentie van de verschillende algoritmen te bepalen. We zullen de huidige residus in elke stap evalueren ten opzichte van de vorige residus.

Het QR-algoritme zonder shifts is conform met de theorie. Indien we theorema 28.4 beschouwen op pagina 218 in het boek Numerical Linear Algebra, komen we tot de vastelling dat dit algoritme een lineare convergentie heeft. Dit op voorwaarde dat de matrix waarop het algoritme toegepast wordt een reële symmetrische matrix is.

Stel dat we het getal, dat besproken is in Vergelijking 1, gebruiken als shift in elke iteratie stap dan bekomen we het QR-algoritme met Rayleigh shifts. In dat geval geld er dat de eigenwaarde en eigenvector schattingen, $\mu^{(k)}$ en $q_m^{(k)}$, identiek zijn aan deze die berekend worden door de Rayleigh quotiënt iteratie dat gestart wordt met $e_m$. Hieruit leiden we af dat het QR-algoritme dan een kubische convergentie heeft, omdat $q_m^{(k)}$ kubisch convergeert naar een eigenvector. Dit komt overeen met het resultaat op de figuur.

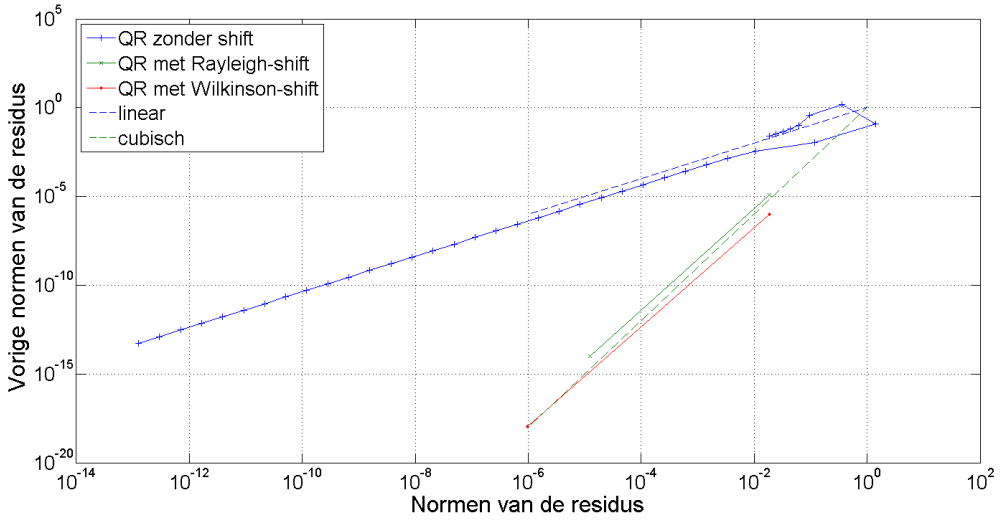
Het QR-algoritme met Rayleigh shifts is zeer gevoelig aan zijn begincondities. Niet in alle gevallen hebben we convergentie, daarom wordt Wilkinson shift gebruikt bij het QR-algoritme. We definiëren de Wilkinson shift als een eigenwaarde van een matrix $B$ dat het dichtst gelegen is bij $a_m$. Waarbij voor $B$ geldt:

$$B = \begin{bmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{bmatrix} \tag{9}$$

Gelijkaardig aan de Rayleigh quotiënt shift bereikt ook de Wilkinson shift kubische convergentie. Meer nog, zelfs in het slechtste geval bereikt het gegarandeerd kubische convergentie. Dit wordt ook bevestigd op Figuur 3.

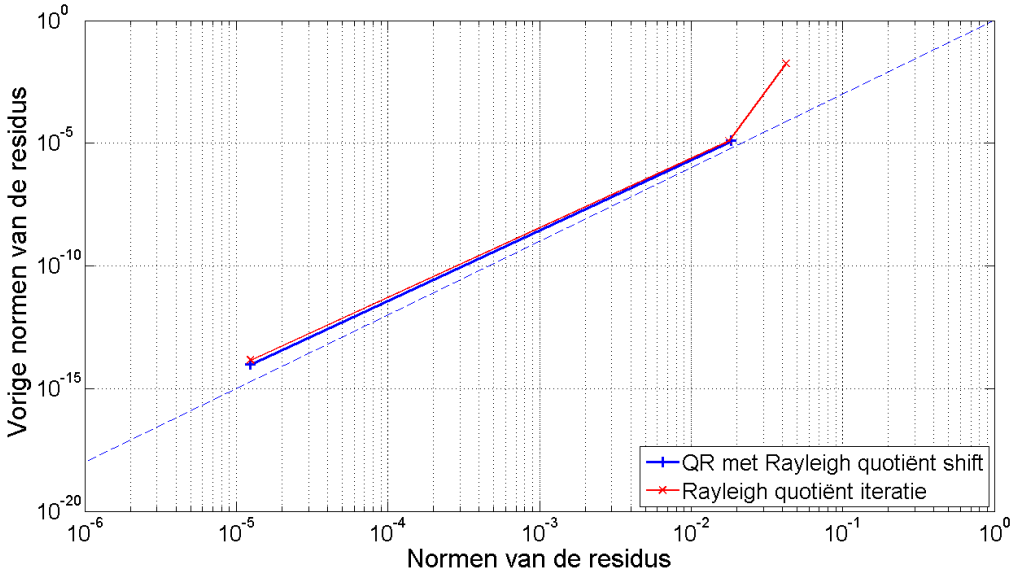
## Opgave 6

Figuur 4 illustreert het convergentiegedrag van het QR-algoritme met Rayleigh shift en Rayleigh quotiënt iteratie. Toepassing van theorema 27.3, op pagina 208 in Numerical Linear Algebra,




**Figuur 3**: Convergentiegedrag van het QR-algoritme zonder shift, met Rayleigh shift en met Wilkinson shift.

leert ons dat Rayleigh quotiënt iteratie een kubische convergentie kent. Verder is er ook een verband aan te tonen tussen Rayleigh quotiënt shift en Rayleigh quotiënt iteratie.

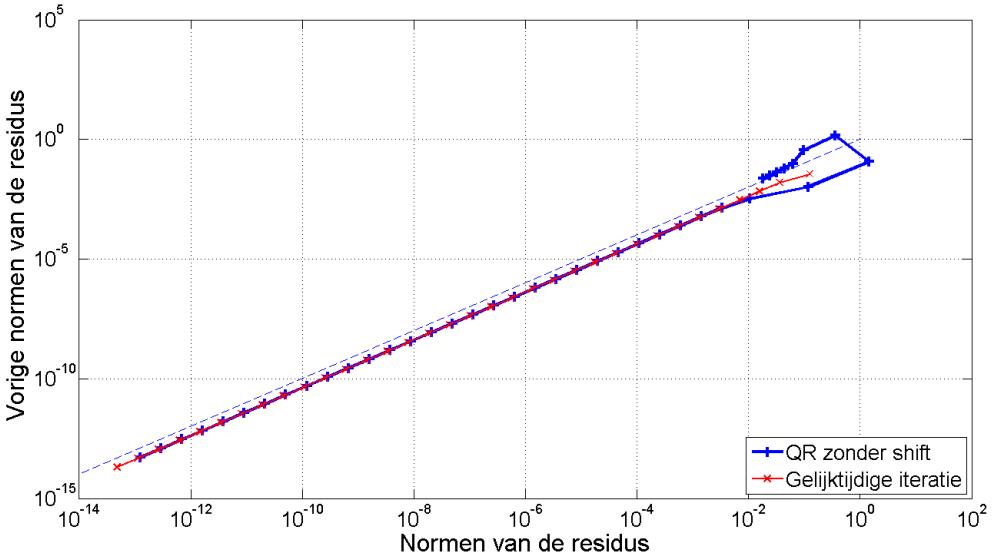



**Figuur 4**: Vergelijking van het convergentiegedrag van het QR-algoritme met Rayleigh shift ten opzichte van Rayleigh quotiënt iteratie.

Wanneer we het QR algoritme met shifts beschouwen, hebben we een manier nodig om snelle convergentie te hebben in de laatste kolom $\underline{Q}^{(k)}$. We kunnen hiervoor het Rayleigh quotiënt gebruiken.

Indien we nu het bovenbeschreven verband beschouwen samen met bovenvernoemde theorema 27.3, komen we tot de vastelling dat beide algoritmen een kubische convergentie moeten vertonen, wat bevestigd wordt door Figuur 4.

Het convergentiegedrag van het QR-algoritme zonder shift en gelijktijdige iteratie wordt getoond in Figuur 5. Beschouwen we verder theorema 28.3, pagina 216 in het boek Numerical Linear Algebra, dan zien we een verband tussen het QR-algoritme zonder shift en het algoritme dat gelijktijdige iteratie gebruikt. Indien men het algoritme voor gelijktijdige iteratie




**Figuur 5**: Vergelijking van het convergentiegedrag van het QR-algoritme zonder shift ten opzichte van gelijktijdige iteratie.

beschouwt:

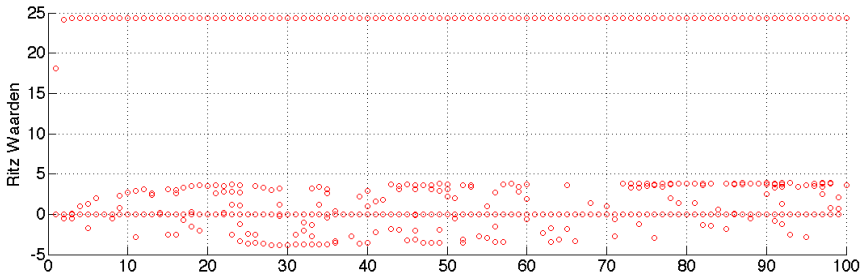
Kies $\hat{Q}^{(0)} \in \mathbb{R}^{m \times n}$ met orthonormale kolommen
**for** $k = 1, 2, \ldots$ **do**
$Z = A\hat{Q}^{(k-1)}$
$\hat{Q}^{(k)}\hat{R}^{(k)} = Z$
**end for**

Dan kan het verband aangetoond worden volgens voorgenoemde theorema als volgt. Beide algoritmen genereren dezelfde matrices $\underline{R}^{(k)}, \underline{Q}^{(k)} en \underline{A}^{(k)}$. Meer nog, dit zijn de matrices gevonden door het uitvoeren van een QR-factorisatie van de k-de macht op de matrix A.

Volgens theorema 28.4, pagina 218 in het boek Numerical Linear Algebra, is de convergentie van een QR-algoritme zonder shifts lineair. Samen met het bovengenoemde verband en theorema 28.4 stellen we vast dat beide algoritmen lineair convergeren. Figuur 5 bevestigt de redenering.

## Opgave 7

We genereren een ijle $1000 \times 1000$ matrix. Wanneer we Arnoldi toepassen op deze matrix en vervolgens de Ritz waarden berekenen in elke stap dan verkrijgen we Figuur 6.




**Figuur 6**: Illustratie van de convergentie van de Arnoldi-Ritz waarden.

Wanneer we de eigenwaarden van de matrix berekenen met het commando $eigs(A)$ verkrijgen we als extreme eigenwaarde 24.3530, de figuur convergeert inderdaad naar deze eigenwaarde toe. Er dient opgemerkt dat Arnoldi niet altijd convergeert naar alle eigenwaarden, enkel naar de extremen. Wanneer we Figuur 6 beschouwen zien we dat de Arnoldi iteratie veel sneller convergeert voor de extreme eigenwaarde dan voor de eigenwaarden die aanzienlijk kleiner zijn dan de extreme eigenwaarde. We zien hier duidelijk een geometrische convergentie naar de extreme eigenwaarde.

# Alternatieve eigenwaardenalgoritmen

## Opgave 8

Beschouw een symmetrische matrix $A \in \mathbb{R}^{m\times m}$ met $A^{(1)}, A^{(2)}, ..., A^{(m)}$ de linksboven vierkante submatrices van dimensies $1, 2, ..., m$. Neem verder aan dat $A$ tridiagonaal is en dat de elementen van de nevendiagonalen verschillend zijn van nul:

$$A = \begin{pmatrix} a_1 & b_1 & 0 & 0 & \\ b_1 & a_2 & b_2 & 0 & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{m-1} \\ & & & b_{m-1} & a_m \end{pmatrix} \tag{10}$$

De eigenwaarden van de submatrix $A^{(k)}$ zijn allen verschillend en worden genoteerd als: $\lambda_1^{(k)} < \lambda_2^{(k)} < ... < \lambda_k^{(k)}$. De eigenwaarden van de verschillende submatrices *interlacen* strikt als volgt:

$$\lambda_j^{(k+1)} < \lambda_j^k < \lambda_{j+1}^{(k+1)} \tag{11}$$

Voor $k = 1, 2, ..., m-1$ en $j = 1, 2, ..., k-1$. Het is net deze eigenschap die het mogelijk maakt om het aantal eigenwaarden van een matrix in een interval te bepalen.

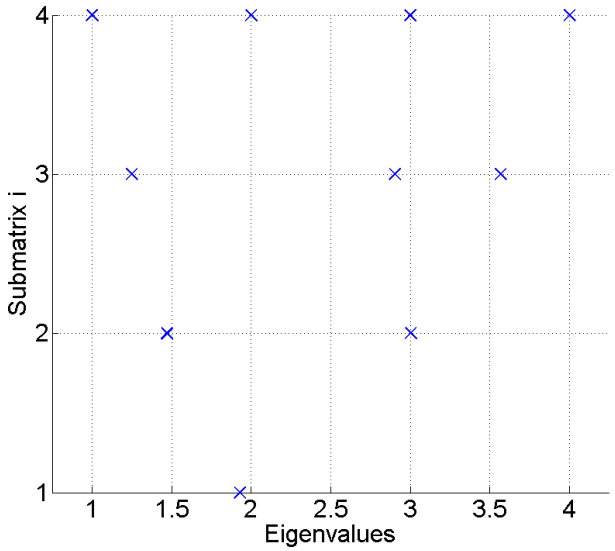
Beschouw bij wijze van voorbeeld de symmetrische matrix $A \in \mathbb{R}^{4\times 4}$:

$$A = PLP^T \tag{12}$$

met $P$ een willekeurige orthogonale matrix, geconstrueerd met de commando's *orth* en *rand*, en $L$ gedefinieerd als volgt:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix} \tag{13}$$

Figuur 7 illustreert de *interlacing* voor de eigenwaarden van de overeenkomstige submatrices.




**Figuur 7**: Illustratie van de *interlacing*-eigenschap voor een eenvoudige reële, symmetrische matrix $A$ met spectrum $\Lambda = \{1, 2, 3, 4\}$.

## Opgave 9

Dit gedeelte beschrijft in pseudo-code het algoritme van de bisectie-methode om alle eigenwaarden van een symmetrische matrix $A \in \mathbb{R}^{m \times m}$ in een interval $[a, b)$ te bepalen. Voor alle duidelijkheid wordt het algoritme beschreven door twee kleinere algoritmes. Een eerste pseudo-code beschrijft een manier om de tekenwissels te tellen in de Sturm-sequentie, een tweede beschrijft de klassieke bisectie-methode met behulp van het vorige.

Uit de *interlacing*-eigenschap, beschreven in de vorige opgave, volgt de Sturm sequentie (veronderstel dezelfde aannames als in de vorige opgave):

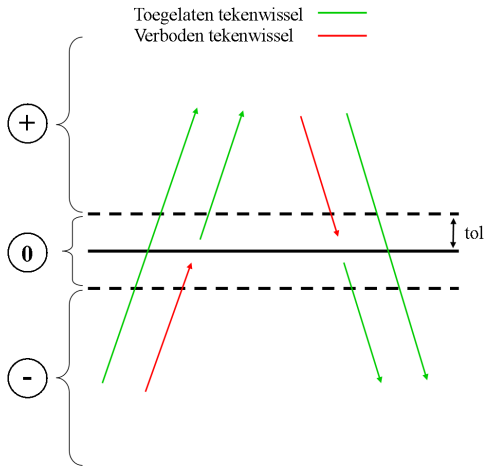
$$p^{(k)}(x) = (a_k - x)p^{(k-1)}(x) - b_{k-1}^2 p^{(k-2)}(x) \tag{14}$$

Definieer $p^{(0)} = 1$ en $p^{(1)} = a_k - x$ opdat dit geldig is voor $k = 2, 3, ..., m$. Indien Vergelijking 14 meermaals wordt toegepast op verschillende waarden voor $x$ en daarbij de tekenwissels in de sequentie worden geteld, kan hierbij het aantal eigenwaarden binnen een interval bepaald worden. Beschouw nu de volgende pseudo-code:

Initialiseer de Sturm sequentie.
$p^{(0)} = 1, p^{(1)} = a_k - x$
Initialiseer de teller.
$t = 0$
**if** $p^{(0)}p^{(1)} < 0$ **then**
  $t = 1$
**end if**
**for** $k = 2, 3, ..., n$ **do**
  Bereken het volgende element uit de Sturm sequentie.
  $p^{(k)} = (a_k - x)p^{(k-1)} - b_{k-1}^2 p^{(k-2)}$
  Verhoog de teller indien een tekenwissel is opgetreden.
  **if** $(p^{(k-1)}p^{(k)} < 0)$ **then**
    $t++$

**end if**
**end for**

Voor de implementatie in MATLAB komen er nog enkele kleinigheden bij zoals de allocatie van vectoren en matrices, andere nummering van elementen, etc. Belangrijker is de definitie van een tekenwissel, zo wordt natuurlijk een wissel van plus naar min en omgekeerd meegenomen. Daarenboven is ook een wissel van nul naar min of plus een tekenwissel, maar *niet* van plus of min naar nul. Dit kan opgelost worden door een absolute tolerantie te definiëren. Het concept wordt getoond in Figuur 8. Zodoende kloppen de grenzen voor het interval; gesloten voor $a$ en open voor $b$. Voor meer details wordt verwezen naar de implementatie, achteraan dit document toegevoegd.




**Figuur 8**: Illustratie van de definitie van een tekenwissel. Deze figuur wordt het best samen bekeken met de volledige implementatie, achteraan dit document toegevoegd.

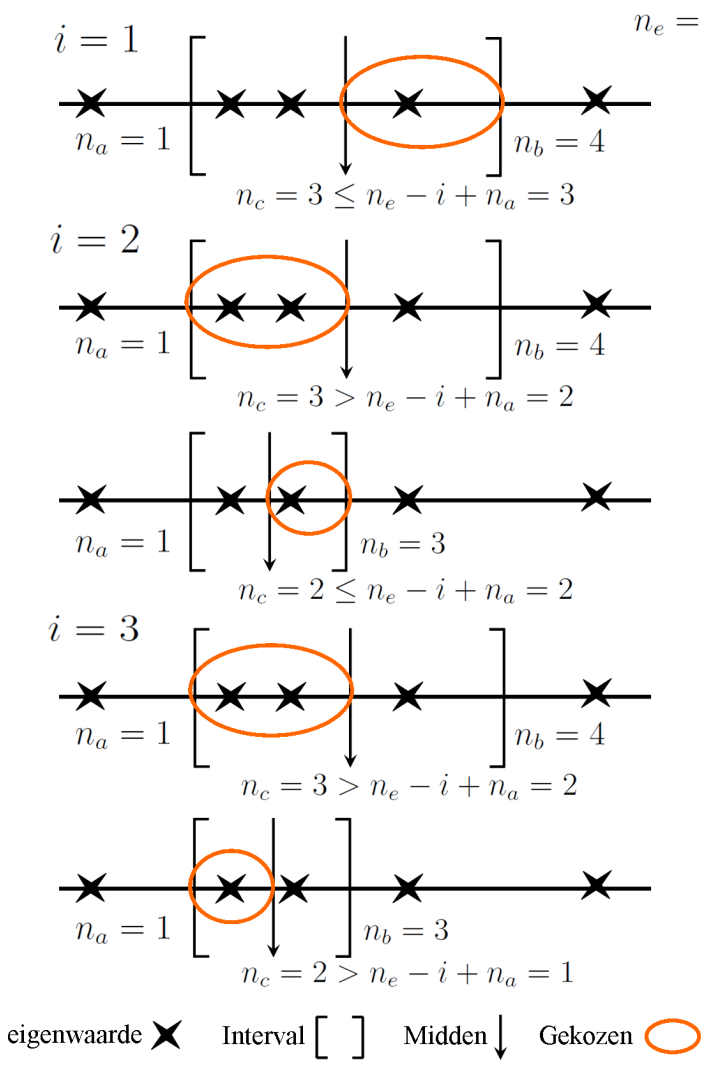
De volgende pseudo-code is, met behulp van het vorige, eerder triviaal:

Bepaal het aantal eigenwaarden $n_a$ en $n_b$ links van $a$ en $b$
Zo ook het aantal eigenwaarden $n_e$ in het interval $[a, b)$.
$n_e = n_b - n_a$
Bepaal alle eigenwaarden
**for** $i = 1, 2, ..., n_e$ **do**
  $lower = a, upper = b, c_i = (lower + upper)/2$
  Klassieke bisectie-methode
  **while** $(|upper - lower| > tol)$ **do**
    Bepaal het aantal eigenwaarden $n_c$ links van $c_i$
    **if** $(n_c \leq n_e - i + n_a)$ **then**
      $lower = c_i$
    **else**
      $upper = c_i$
    **end if**
    $c_i = (upper + lower)/2$
  **end while**
**end for**

Ook hier zijn er enkele verschillen met de implementatie in MATLAB zoals: allocatie van vectoren en matrices, een maximaal aantal iteraties voor de bisectie, etc. In de implementatie

worden ook de bijbehorende eigenvectoren berekend door middel van één stap van inverse iteratie.

Figuur 9 toont een eenvoudig voorbeeld van de werking van het algoritme. Hierbij wordt duidelijk dat alle eigenwaarden in het interval worden bereikt; verdere stappen zijn triviaal.

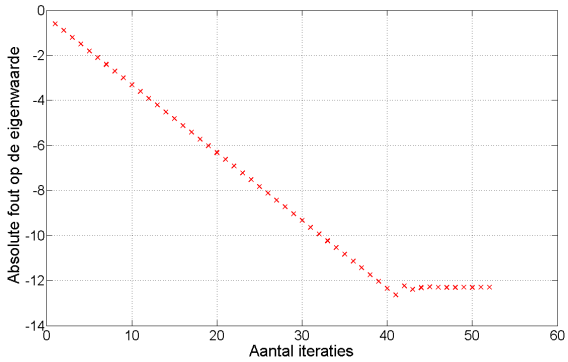



**Figuur 9**: Illustratie van de werking van de bisectie-methode aan de hand van een eenvoudig voorbeeld.

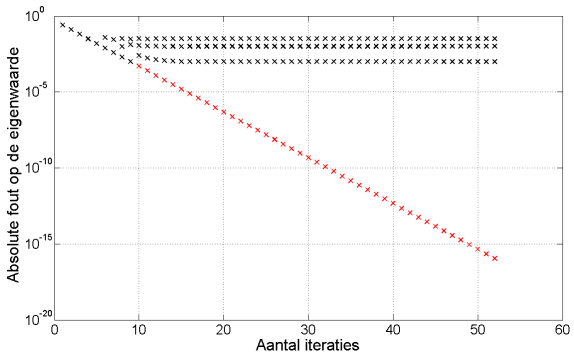
Om de correcte werking van het algoritme aan te tonen berekenen we de eigenwaarde gelegen in het interval $[-1, 2)$ van een matrix met spectrum $\Lambda = \{-100, 1, 100, 200\}$. Figuur 10a toont dat de eigenwaarde $\lambda = 1$ binnen 40 iteraties wordt gevonden tot op hoge nauwkeurigheid. Figuur 10b illustreert het convergentiegedrag voor een matrix met een eigenwaarde $\lambda = 1$ met een multipliciteit van vier. De eigenwaarde wordt slechts één keer gevonden tot op hoge nauwkeurigheid.

## Opgave 10

Figuur 11 illustreert de rekentijd van het QR-algoritme met Rayleigh quotiënt shift en de bisectie-methode voor het bepalen van zeven eigenwaarden en bijbehorende eigenvectoren voor een aantal matrices $A \in \mathbb{R}^{m \times m}$ van verschillende groottes met $m = 20, 40, 80, 160, 320, \ldots$.



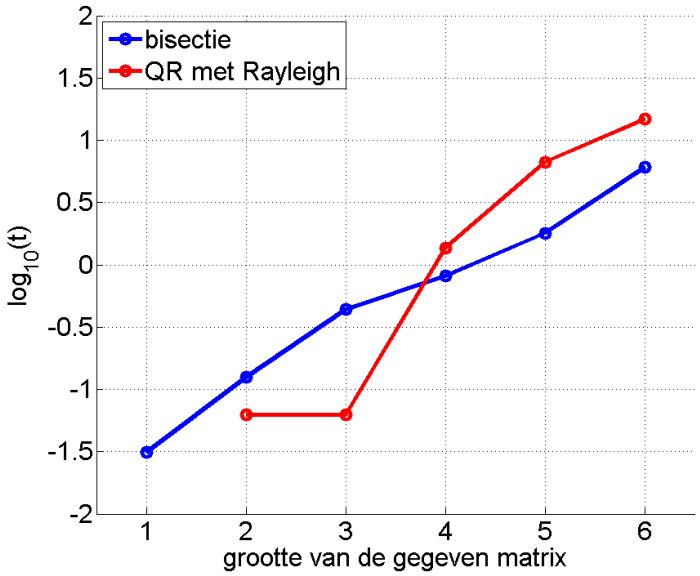

(a) Convergentiegedrag van de bisectie-methode voor het berekenen van de eigenwaarde gelegen in het interval $[-1, 2)$ voor een matrix met spectrum $\Lambda = \{-100, 1, 100, 200\}$.




(b) Convergentiegedrag van de bisectie-methode voor het berekenen van de eigenwaarde gelegen in het interval $[-1, 2)$ voor een matrix met eigenwaarde $\lambda = 1$ met multipliciteit vier.

**Figuur 10**: Convergentiegedrag van de bisectie-methode voor enkele testproblemen.

Voor een klein aantal eigenwaarden doet de bisectie-methode het beter, in de zin van lagere rekentijden, dan het QR-algoritme met Rayleight quotiënt shift. Alhoewel de bisectie-methode slechts lineaire convergentie heeft (net zoals het klassieke bisectie-algoritme), is de kost voor een evalutatie van de Sturm sequentie slechts $\mathcal{O}(m)$. Dit is een duidelijke verbetering ten opzichte van het QR-algoritme met $\mathcal{O}(m^2)$ flops.




**Figuur 11**: Vergelijking van de rekentijd van het QR-algoritme met Rayleigh quotiënt shift en de bisectie-methode voor het bepalen van een aantal eigenwaarden en bijbehorende eigenvectoren voor matrices van verschillende groottes $A \in \mathbb{R}^{m \times m}$ met $(m = 20, 40, 80, 160, 320, ...)$.

# Implementatie van de bisectie-methode

```
function [E,V] = bisection(A,a,b,tol)
maxIter = 500;
H = hess(A);
na = p(a,H);
nb = p(b,H);
ne = nb - na;
c = zeros(ne,1);
n = length(A);
V = zeros(n,ne);
for i = 1:ne
    k = 0;
    lower = a;
    upper = b;
    c(i) = (lower + upper)/2;
    while (( k <= maxIter ) && ( abs(upper-lower) > tol))
        if (p(c(i),H) <= ne-i+na)
            lower = c(i);
        else
            upper = c(i);
        end
        c(i) = (upper + lower)/2;
        k = k+1;
    end
    [~,V(1:n,i)] = inviter(A, c(i), ones(n,1)./sqrt(n), 10);
end
E = c;
```

```
function [s] = p(x,H)
    tol = 1e-6;
    n = size(H,1);
    f = zeros(n+1,1);
    f(1,1) = 1; f(2,1) = H(1,1)-x;
    t = 0;
    if (f(1)*f(2) < 0)
        t = 1;
    end
    for i = 3:n+1
        f(i,1) = (H(i-1,i-1)-x)*f(i-1,1)-(H((i-1)-1,i-1)^2)*f(i-2,1)
            ;
        if ((f(i-1,1)*f(i,1)<0) && (abs(f(i-1,1))>tol) ...
                    && (abs(f(i,1))>tol))
            t = t+1;
        elseif ((abs(f(i-1,1))<tol) && ((abs(f(i,1))>tol)))
            t = t+1;
        end
    end
    s = t;
end
```