Numerieke Modellering en Benadering

Practicum 1: Eigenwaardenproblemen

De Wolf Peter
Vekemans Wout

7 mei 2014

Inhoudsopgave

Lijst van figuren

Inleiding

In dit practicum onderzoeken we methoden voor het bepalen van eigenwaarden van volle matrices. In een eerste sectie beschouwen we enkele theoretische eigenschappen van de methoden. In een tweede sectie worden de convergentie-eigenschappen van de methoden onderzocht aan de hand van MATLAB-experimenten. In de derde en laatste sectie gaan we dieper in op één van de methoden, namelijk de Jacobi-methode.

Theoretische eigenschappen

Opgave 1

Gelijktijdige iteratie is een andere naam voor het toepassen van de methode van de machten op meerdere kolommen van A tegelijk. Zo kunnen de grootste n eigenwaarden van de matrix A gevonden worden. Om met gelijktijdige iteratie de kleinste n eigenwaarden van A te berekenen is het voldoende om het algoritme toe te passen op A^{-1} in plaats van op A. Dit is makkelijk in te zien. De eigenwaarden van de inverse van een singuliere matrix A zijn gelijk aan het omgekeerde van de eigenwaarden van A.

$$Ax = \lambda x \tag{1a}$$

$$A\frac{1}{\lambda}x = x \tag{1b}$$

$$A^{-1}A\frac{1}{\lambda}x = A^{-1}x \tag{1c}$$

$$A^{-1}x = \frac{1}{\lambda}x \tag{1d}$$

Als we dit algoritme uitschrijven krijgen we volgende pseudocode:

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

Dit zal convergeren naar de n grootste eigenwaarden van A^{-1}, die zoals eerder aangetoond in (1) overeenkomen met de n kleinste eigenwaarden van A.

Opgave 2

a) De Rayleigh quotiënt iteratie convergeert zeer snel naar de juiste waarde van de eigenwaarden van A. Elke iteratiestap verdrievoudigt het aantal juiste cijfers. Elke stap dient er wel een stelsel te worden opgelost dat meer en meer singulier wordt naarmate de benadering voor de eigenwaarde juister wordt. Het betreft het stelsel

$$(A = \lambda^{(k-1)}I)w = v^{(k-1)} \tag{2}$$

Als λ dichter en dichter bij een eigenwaarde van A komt wordt $A - \lambda^{(k-1)}I$ meer en meer singulier, en zal de inverse van deze matrix, die nodig is om (2) op te lossen, ervoor zorgen dat de vector w enorm wordt opgeblazen. De vector zal echter wel in de juiste richting blijven wijzen en wordt genormeerd, waardoor dit geen problemen oplevert.

b) 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. Toon aan dat de oplossing $\rho \in \mathbb{R}$ van het minimalisatieprobleem

$$\min_{\rho\in\mathbb{R}} ||Ax - \rho x||_2 \tag{3}$$

overeenkomt met het Rayleighquotiënt van x.
Dit probleem lijkt op het standaard kleinste kwadraten probleem $Ax \approx b$. We proberen nu een ρ te vinden zodat $||Ax - \rho x||_2$ geminimaliseerd wordt. De overeenkomstige normaalvergelijkingen (cf $A^TAx = A^Tb$) worden gegeven door

$$x^Tx\rho = x^TAx \tag{4a}$$

$$\rho = \frac{x^TAx}{x^Tx} = r(x) \tag{4b}$$

waarbij $r(x)$ het Rayleigh quotiënt van x voorstelt.

Opgave 3

a) Gegeven een schatting voor de eigenvector horend bij de derde kleinste eigenwaarde van een symmetrische matrix $A \in \mathbb{R}^{m\times m}$, bepaal de bijhorende eigenwaarde.
Als een benadering v voor een eigenvector gekend is, kan er m.b.v. het

Rayleigh quotiënt μ van die vector een benadering van de bijhorende eigenwaarde worden berekend.

$$\mu = \frac{v^T A v}{v^T v} \tag{5}$$

Daarna kan er met een inverse iteratiestap een betere benadering voor de eigenvector worden berekend. Als deze stappen k keer herhaald worden spreekt men van Rayleigh quotiënt iteratie. Deze methode kent een kubische convergentie voor een symmetrische matrix.
Om het algoritme efficiënt te kunnen uitvoeren moet de matrix eerst gereduceerd worden tot Hessenberg vorm, dit vraagt $\mathcal{O}(\frac{10}{3}m^3)$ flops. Na deze reductie heeft Rayleigh quotiënt iteratie maar $\mathcal{O}(m)$ flops per iteratiestap nodig, in tegenstelling tot de $\mathcal{O}(m^3)$ flops per stap indien er niet gereduceerd wordt.

b) Gegeven een symmetrische matrix $A \in \mathbb{R}^{m \times m}$, bepaal de eigenwaarde het dichts gelegen bij een getal α.
Om een schatting van een eigenvector te krijgen, maken we gebruik van 1 iteratiestap van inverse iteratie. Hiervoor lossen we het stelsel (2) op, met α als waarde voor λ, en v een willekeurige vector met $||v||_2 = 1$.
De vector w die we dan krijgen kunnen we gebruiken als startvector voor Rayleigh quotiënt iteratie. Dit heeft kubische convergentie en is dus het snelst van alle beschouwde algoritmes. Zoals hierboven vermeld is er eerst reductie tot Hessenberg vorm nodig om het aantal flops per iteratiestap te verminderen.

Convergentie-experimenten

Voor al deze experimenten wordt gebruik gemaakt van een volle, reële symmetrische matrix A. We laden hiervoor de gegeven matrix $mat1.txt$ in in MATLAB.

Opgave 4

Het uitvoeren van het *spy*-commando laat zien dat er geen enkel $non-zero$ element in de matrix zit. De uitvoering van het QR-algoritme op een matrix met die afmetingen (nog steeds relatief klein) zou zeer veel werk vragen.
Na het reduceren tot Hessenberg vorm zien we dat alle elementen onder de eerste benedendiagonaal nul zijn geworden. Verdere analyse leert ons dat ook de elementen boven de eerste bovendiagonaal allemaal van grootteorde ϵ_{mach} zijn. Als we de Hessenberg vorm van de matrix afronden tot op 15

decimalen verkijgen we een tridiagonale matrix.(zie figuur 1) Dit komt doordat de originele matrix symmetrisch was. De reductie naar Hessenberg vorm zorgt ervoor dat het algoritme niet op een volledige matrix moet inwerken. Het uitvoeren van het QR-algoritme vraagt nu slechts $\mathcal{O}(n^2)$ flops.

(a) zonder afronden (b) met afronden

Figuur 1: Structuur van de Hessenberg matrix

Opgave 5

Het toepassen van de drie verschillende versies van het QR algoritme geeft ons figuur 2. Het is duidelijjk dat er twee kubisch convergerende methodes zijn, en 1 lineaire.
Als maat voor de convergentie gebruiken we de waarde van een element net onder de diagonaal A. Dit doen we omdat de matrix convergeert naar een diagonaalmatrix. Hoe dichter de waarde van dat element bij nul ligt, hoe dichter de matrix een diagonaalmatrix benadert. We zien op de grafiek duidelijk dat het residu bij het algoritme zonder shifts lineair convergeert naar 0. De algoritmes met shifts (Rayleigh en Wilkinson) convergeren kubisch en zijn dus veel sneller dan het algoritme zonder shifts. Het is zelfs zo dat de methoden met shift al een absolute fout kleiner dan 10^{-16} hebben na een tweetal stappen.

Figuur 2: Convergentie van verschillende QR algoritmes

Voor het algoritme zonder shifts stemt dit volledig overeen met de theorie. Theorema 28.4 (p. 218 in het handboek) geeft duidelijk aan dat bij het gebruik van QR zonder shifts $A^{(k)}$ lineair convergeert naar een diagonaalmatrix met de eigenwaarden van A als elementen. Deze lineaire convergentie kan sterk verbeterd worden door het gebruik van shifts.
Als we bijvoorbeeld het Rayleigh quotiënt gebruiken als shift in elke iteratiestap krijgen we kubische convergentie naar de eigenwaarde die hoort bij de eigenvector die benaderd wordt door de laatste kolom van A. Dan zijn de berekenden benaderingen μ en q_m in elke stap gelijk aan diegenen die berekend worden door de Rayleigh quotiënt iteratie die gestart wordt met de m^{de} eenheidsvector. Door dit verband is het duidelijk dat het QR algoritme met Rayleigh shift in het beste geval kubisch convergeert. De Rayleigh iteratie

doet dit immers ook. Dit is duidelijk te zien op de figuur.
Om een resultaat te vinden dat minder afhankelijk is van beginvoorwaarden kunnen we ook Wilkinson shift gebruiken. Hiervoor nemen we een submatrix B uit A en gebruiken we de eigenwaarde van B die het dichtste bij a_m ligt als shift. Dit algoritme garandeert zelfs in het slechtste geval nog kwadratische convergentie, in tegenstelling tot het algoritme met Rayleigh shifts, dat soms helemaal niet convergeert.

$$B = \begin{bmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{bmatrix} \tag{6}$$

Iteratie	**unshifted**	**Wilkinson**	**Rayleigh**
1	7.6359E-01	1.8360E-02	1.8347E-02
2	2.1011E-01	9.8332E-07	1.2342E-05
3	8.0838E-02	1.1383E-18	9.5580E-15
4	3.3388E-02	0	0
5	1.4090E-02	0	0

Tabel 1: Residu van de 3 verschillende methoden, voor de eerste vijf stappen

Opgave 6

De Rayleigh quotiënt iteratie en de Rayleigh shift methode zouden allebei kubisch moeten convergeren. In figuur 3 is de convergentie van beide methoden weergegeven. We zien duidelijk dat er een gelijkaardige convergentiesnelheid is. Dit komt doordat de Rayleigh shift methode gebruik maakt van het Rayleigh quotiënt. Het QR algoritme met Rayleigh shifts is namelijk een methode waarbij er snelle convergentie is in de laatste kolom van $Q^{(k)}$. Als we (7) toepassen op de laatste kolom van de matrix krijgen we een goede schatting voor de eigenwaarde horende bij die laatste kolom.

$$\mu^{(k)} = \frac{(q_m^{(k)})^T A q_m^{(k)}}{(q_m^{(k)})^T q_m^{(k)}} = (q_m^{(k)})^T A q_m^{(k)} \tag{7}$$

Nu is dit getal gelijk aan $A_{mm}^{(k)}$ en kan dit getal zo als shift gebruikt worden voor de volgende iteratiestap. Zoals eerder gezegd toont figuur 3 duidelijk dat beide methodes ongeveer even snel convergeren naar een exacte waarde voor de eigenwaarde.

Figuur 3: Vergelijking convergentie Rayleigh shift en Rayleigh quotiënt

De convergentie van gelijktijdige iteratie en QR zonder shifts wordt geïllustreerd in figuur 4. Als residu gebruiken we een elementje net onder de diagonaal van A. Aangezien de matrix moet convergeren naar een diagonaalmatrix geeft dit een duidelijk beeld van de convergentie. Hoe sneller het elementje naar 0 nadert, hoe sneller de convergentie. Het is duidelijk dat deze methodes ongeveer even snel convergeren. Dit kan eenvoudig worden aangetoond. Het algoritme voor gelijktijdige iteratie is als volgt:

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

Theorema 28.3 (p. 216) in het handboek toont aan dat gelijktijdige iteratie en unshifted QR dezelfde matrices $\underline{R}^{(k)}, \underline{Q}^{(k)} en \underline{A}^{(k)}$ genereren. Het is ook eenvoudig in te zien dat dit de matrices zijn die gedefinieerd worden door de k^{de} macht van de QR factorisatie van A.

Volgens theorema 28.4 (p. 218) convergeert het QR algoritme zonder shifts lineair naar een diagonaalmatrix. Aangezien gelijktijdige iteratie dezelfde matrices produceert moet deze methode een gelijkaardige convergentiesnelheid hebben. Uit figuur 4 wordt duidelijk dat matrix A inderdaar voor allebei de methodes lineair convergeert naar het resultaat.

Figuur 4: Vergelijking convergentie van unshifted QR en gelijktijdig iteratie

Opgave 7

We hebben een random ijle matrix $A \in \mathbb{R}^{1000 \times 1000}$ gegenereerd. Als we hierop de Arnoldi iteratie toepassen en de Ritz waarden berekenen voor elke iteratiestap krijgen we figuur 5. We zien duidelijk dat er zeer snelle, geometrische convergentie is naar de extreme eigenwaarde 24.3868 die we berekend hebben met het commando $eigs(A)$. De andere eigenwaarden convergeren niet zo snel.

Figuur 5: Convergentie van Arnoldi-Ritz waarden

Alternatieve eigenwaardenalgoritmen

Opgave 8

De Jacobi methode voor het vinden van eigenwaarden van een symmetrische matrix A steunt op de vermenigvuldiging

$$J^T \begin{bmatrix} a & d \\ d & b \end{bmatrix} J \tag{8}$$

waarin geldt

$$J = \begin{bmatrix} c & s \\ \text{-s} & c \end{bmatrix} \tag{9}$$

Hieronder zullen we de waarde van s en c berekenen zodat de niet-diagonaalelementen na de vermenigvuldiging (8) gelijk zijn aan 0.
Aangezien het gaat om een rotatiematrix, moet de determinant van J gelijk zijn aan 1.

$$c^2 + s^2 = 1 \tag{10}$$

Daarom kozen we voor volgende waarden voor s en c

$$\begin{cases} c = \cos(\theta) \\ s = \sin(\theta) \end{cases} \tag{11}$$

Dit geeft ons volgende uitdrukking voor de waarde van de niet-diagonaalelementen

$$\frac{1}{2}\sin(2\theta)(a-b) + d\cos(2\theta) = 0 \tag{12}$$

Verder uitwerken geeft ons

$$\theta = \frac{1}{2}\arctan(\frac{2d}{b-a}) \tag{13}$$

Als $a = b$ stellen we θ gelijk aan $\frac{\pi}{4}$, omdat bij die hoek de boogtangens oneindig is.

Opgave 9

Tijdens een Jacobi iteratie zal de diagonaal van de matrix A overlopen worden, tot aan het voorlaatste diagonaalelement. Per diagonaalelement zal er in de rij gezocht worden naar het in absolute waarde grootste element verschillend van het diagonaalelement. Dit element zal geselecteerd worden omdat het de grootste impact heeft op het niet diagonaal zijn van de matrix. Daarom gaan we dit element verkleinen. Om dit te doen kiezen we als tweede diagonaalelement het element in dezelfde kolom als het grootste element. Met behulp van deze drie waarden en formule (13) berekenen we θ. De J-matrix verkrijgen we door op een eenheidsmatrix met dezelfde dimensies als A de overeenstemmende diagonaalelementen te vervangen door $\cos(\theta)$ en het element op de plaats van het grootste element en het spiegelbeeld van dit element te vervangen door respectievelijk $\sin(\theta)$ en $-\sin(\theta)$. Vervolgens vermenigvuldigt men matrix A langs links met J^T om de rijen aan te passen en langs rechts met J om de kolommen aan te passen.
Om een matrix met de bijhorende eigenvectoren te verkrijgen begint men met matrix V te initialiseren als een eenheidsmatrix. Bij elke vermenigvuldiging van A met J moet men V langs rechts vermenigvuldigen met J.

input(A,tol) A = symmetrische matrix, tol = bovengrens voor de fout op een eigenwaarde
n = size(A)
V = identitymatrix(n)
while maximum error >tol **do**
  for elk diagonaalelement behalve het laatste **do**
    max = kies het in grootste niet diagonaalelement van deze rij
    rij van diagonaalelement = n, m = kolom van max
    definieer a, b en d met $a = A_{n,n} b = A_{m,m}$ en $d = A_{n,m}$
    bepaal θ m.b.v (13)
    construeer Jacobimatrix J
    $A = J^T A J$
    $V = V J$
  end for
end while

Opgave 10

Als we onze implementatie van de Jacobi methode uitvoeren met matrix $mat1.txt$ als input en machineprecisie als tolerantie, krijgen we figuur 6. We zien duidelijk dat de convergentie sneller dan lineair is, en eerder kwadratisch

lijkt. Dit komt overeen met de theorie.

Figuur 6: Convergentie van Jacobi methode

Voor een symmetrische tridiagonale matrix $A \in \mathbb{R}^{m \times m}$ zijn er slechts $\mathcal{O}(m)$ elementen die gelijk moeten worden aan 0. Voor elke van deze elementen moet θ worden berekend en moeten er 3 matrixvermenigvuldigingen worden uitgevoerd. Na elke stap daalt de som van de kwadraten van de niet-diagonaalelementen met een constante factor. De convergentie kan dus gegarandeerd worden na $\mathcal{O}(m^2 \log(\epsilon_{mach}))$ stappen. Bij een volle matrix zouden er $\mathcal{O}(m^2)$ niet-nul elementen, en zouden er m keer meer bewerkingen nodig zijn.

Implementatie van de Jacobimethode

```
1  function [V,D,errormat] = jacobitol(A,tol)
2
3  % A =de matrix waarvan de eigenwaarden moeten bepaald
       worden
4  % tol = de toegelaten tolerantie op de maximale fout
       van de eigenwaarden.
5  [n,p] = size(A);
6  errormat = [];
7  if n~=p,
8    disp('A is geen vierkante matrix')
9    return
10 end
11 if n<2
12   disp('A moet minstens dimensie 2 hebben')
13   return
14 end
15 error = tol +1;
16 V = eye(n);
17 while(error > tol)
18 for index = 1:(n-1);
19     %find largest off-diagonal element in row with
           number index
```

```
20      place = index+1;
21      for elementnumber = index+2:n;
22          if (elementnumber > n)
23              elementnumber_to_check = n;
24          else
25              elementnumber_to_check = elementnumber;
26          end
27
28          if (abs(A(index, elementnumber_to_check)) > abs
                (A(index, place)))
29              place = elementnumber_to_check;
30          end
31      end
32      a = A(index,index);
33      b = A(place, place);
34      d = A(index, place);
35
36      theta = 0.5* atan(2*d/(b-a));
37      if ( b == a)
38          theta = pi/4;
39      end
40      s = sin(theta);
41      c = cos(theta);
42      spiegelmatrix = eye(n);
43      spiegelmatrix(index,index) = c;
44      spiegelmatrix(index,place) = s;
45      spiegelmatrix(place,index) = -s;
46      spiegelmatrix(place,place) = c;
47      A = transpose(spiegelmatrix)* A * spiegelmatrix;
48      V = V* spiegelmatrix;
49  %find the maximum off-diagonal element
50  %start at element(1,2)
51  error = abs(A(1,2));
52  end
53  for rownumber = 1:n;
54      for columnnumber = 1:n;
55          if(rownumber ~= columnnumber)
56              if abs(A(rownumber, columnnumber)) > error
57                  error = abs(A(rownumber,columnnumber));
58              end
59          end
```

```
60        end
61 end
62 errormat=[errormat  error];
63 end
64 disp('error');
65 disp(error);
66 D = A;
```