Numerieke Modellering & Benadering Practicum 2

Dennis Debree Jonas Bertels

20 mei 2019

1 Evoluerende kleinste kwadraten problemen

2 Rayleigh quotiënt iteratie

In deze sectie bespreken we enkele aspecten voor het benaderen van eigen waarden bij symmetrische vierkante matrices.

2.1 Numerieke stabiliteit

Wanneer de benaderde eigenwaarde $\lambda^{(k-1)}$ dichter in de buurt komt wordt de matrix $A-\lambda^{(k-1)}I$ almaar meer singulier gezien de determinant van deze matrix dichter en dichter bij nul komt. In het geval van Rayleigh quotiënt iteration (alsook inverse iteration) vormt dit geen numerieke problemen. Dit kunnen we zo inzien: Neem een reële symmetrische matrix A met een eigenwaarde veel kleiner in absolute waarde dan de andere. Nemen we dan een vector v met componenten in de richtingen van iedere eigenvector bij de eigenwaarden van A. Los dan $Aw = v$ met een achterwaarts stabiel algoritme op naar $\tilde{w}$. Maken we gebruik van de volgende formule (zie p. 95 in 'Numerical Linear Algebra' (L. Trefethen en D. Bau, 1997)):

$$\frac{||\delta w||}{||w||} \leq \kappa(A)\frac{||\delta A||}{||A||} \tag{1}$$

Waarbij de gelijkheid enkel geldig is indien de berekende vector een veelvoud is van de singuliere vector horende bij de kleinste eigenwaarde van A. Gezien de startvector componenten heeft in alle richtingen, is de kans dat we de eigenvector bij de kleinste eigenwaarde benaderen zeer klein, dus zal de relatieve fout veel kleiner zijn dan het slechtste geval.

2.2 Link met kleinste kwadraten probleem

We tonen nu aan dat de oplossing voor het minimalisatie probleem

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

overeenkomt met het Rayleigh quotiënt van x.

Bovenstaande expressie is een kleinste kwadraten probleem waarbij $x\rho \approx Ax$ met x de matrix, ρ de onbekende vector en Ax de rechterhand zijde. Als we met die gegevens de normaalvergelijkingen uitschrijven:

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

Geeft het ons onmiddellijk de formule voor het Rayleigh quotiënt.

3 Het QR-algoritme

We gaan nu over naar een ander algoritme die alle eigenwaarden en bijhorende eigenvectoren van een reële symmetrische matrix $A \in \mathbb{R}^{m \times m}$ kan bepalen, namelijk het QR-algoritme zoals beschreven in algoritme 28.2 uit 'Numerical Linear Algebra' (L. Trefethen en D. Bau, 1997).

3.1 Enkele eigenschappen

Nu tonen we enkele interessante eigenschappen van dit algoritme, hierbij gebruiken we andere theorema's uit hetzelfde boek.

3.1.1 Behoud van eigenwaarden

We tonen nu het behoud van eigenwaarden aan in het algoritme 28.2. In het begin van het algoritme wordt de matrix A getridiagonaliseerd. Dit komt neer op een gelijkvormigheidstransformatie met orthogonale matrices dewelke volgens theorema 24.3 de eigenwaarden behoud. Ook de deflatie zorgt vanzelfsprekend niet voor verlies van eigenwaarden. We tonen nu aan dat het uitrekenen van A^k ook de eigenwaarden behoud: Gegeven de formules:

$$A^{(k-1)} - \mu^{(k)} I = Q^{(k)} R^{(k)} \tag{2}$$
$$A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I \tag{3}$$

Uit (2) bepalen we :

$$R^{(k)} = (Q^{(k)})^T (A^{(k-1)} - \mu^{(k)} I) \tag{4}$$

En dan (4) invullen in (3):

$$A^{(k)} = (Q^{(k)})^T (A^{(k-1)} - \mu^{(k)} I) Q^{(k)} + \mu^{(k)} I \tag{5}$$
$$A^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)} - \mu^{(k)} (Q^{(k)})^T Q^{(k)} + \mu^{(k)} I \tag{6}$$
$$A^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)} \tag{7}$$

(6) naar (7) gebruikt het feit dat $(Q^{(k)})^T Q^{(k)} = I$. Gezien $Q^{(k)}$ een orthogonale matrix is, is dit een gelijkvormigheidstransformatie en dus worden de eigenwaarden behouden.

3.1.2 Behoud van de tridiagonaal vorm

We tonen nu aan dat de tridiagonaal vorm behouden wordt in $A^{(k)}$ voor $k = 1, 2, \ldots$.
Gegeven is $A^{(k-1)}$ dat symmetrisch en tridiagonaal is. Een tridiagonaal matrix is automatisch ook bovenhessenberg. De gebruikte shifts behouden de tridiagonaal vorm. Zolang $\mu^{(k)}$ geen exacte eigenwaarde is, kunnen we steeds de QR-factorisatie bepalen. We tonen nu aan dat $A^{(k)} = R^{(k)}Q^{(k)}$ de tridiagonaal vorm bewaart.
In een QR-factorisatie is R een bovendriehoeks matrix. Gezien de inverse van een bovendriehoeks matrix opnieuw bovendriehoeks is, is $Q^{(k)} = A^{(k-1)}R^{(k)}$ een bovenhessenberg matrix want een bovenhessenberg matrix maal een bovendriehoeks matrix (of omgekeerd) geeft opnieuw een bovenhessenberg matrix.
Dit geeft ons dan $A^{(k)} = R^{(k)}Q^{(k)}$ wat opnieuw een bovendriehoeks maal bovenhessenberg is, dus is $A^{(k)}$ een bovenhessenberg. Uit (7) weten we dat $A^{(k)}$ via een gelijkvormigheidstransformatie berekend wordt met orthogonale matrices, wat de symmetrie behoud (A symmetrisch, Q orthogonaal):

$$(Q^T AQ)^T = Q^T A^T (Q^T)^T = Q^T A^T Q = Q^T AQ$$

Dus is $A^{(k)}$ symmetrisch en bovenhessenberg dus automatisch tridiagonaal.

3.2 Een efficiënte implementatie

Nu we deze handige eigenschappen hebben bewezen, kunnen we ze gebruiken om een efficiënt algoritme te bepalen voor het bepalen van $A(k)$ via (2) en (3). We beginnen met het bepalen van de QR-factorisatie: De matrix $A^{(k-1)} - \mu^{(k)}I$ is een tridiagonaal matrix, deze reduceren kan gemakkelijk door givens-rotaties op ieder subdiagonaal element toe te passen. We houden deze givens-rotaties bij in de matrices $T_1^{(k)} \ldots T_m^{(k)}$ zodat we de Q matrix niet expliciet moeten berekenen. Deze is namelijk:

$$Q^{(k)} = (T_1^{(k)})^T (T_2^{(k)})^T \ldots (T_m^{(k)})^T$$

Eens deze bepaald is kunnen we dan $A^{(k)}$ bepalen uit de berekende $R^{(k)}$ en de givens-rotatie matrices (zie ook algoritme 1).
Het bepalen van van de matrix $A^{(k-1)} - \mu^{(k)}I$ heeft exact m optellingen en m vermenigvuldigingen. Het bepalen van de m givens-rotaties is iets complexer maar ook een operatie van orde $O(m)$. De hieruit bepaalde $R^{(k)}$ en $T_1^{(k)} \ldots T_m^{(k)}$ kunnen dan efficiënt vermenigvuldigd worden gezien er bij de $T_i^{(k)}$ matrices steeds slechts 2 niet diagonaal elementen verschillen van nul. Hieruit volgt dat $R^{(k)}Q^{(k)}$ opnieuw orde $O(m)$ is. Dan is de optelling met $\mu^{(k)}I$ ook nog van orde $O(m)$ waardoor het volledig algoritme van orde $O(m)$ is.

Algorithm 1: Iteratie in QR-algoritme

Input: $A^{(k-1)}$ tridiagonaal en symmetrisch, $\mu^{(k)}$
Output: $A^{(k)}$
begin
  $A^{(k-1)} \leftarrow A^{(k-1)} - \mu^{(k)} I$
  for $i = 1 \dots m$ **do**
    $T_i \leftarrow$ Givens rotatie die element $A^{(k-1)}[i+1, i]$ nul maakt.
    $A^{(k-1)} \leftarrow T_i A^{(k-1)}$
  end
  $A^{(k)} \leftarrow A^{(k-1)}$
  for $i = m \dots 1$ **do**
    $A^{(k)} \leftarrow A^{(k)} (T_i)^T$
  end
  $A^{(k)} \leftarrow A^{(k)} + \mu^{(k)} I$
end

4 Alternatieve eigenwaardenalgoritmen

We zagen reeds de Rayleigh quotiënt iteratie en QR-algoritme voor het vinden van eigenwaarden. Er bestaan echter nog andere algoritmes om de eigenwaarden van een symmetrische matrix te bepalen. We bespreken hieronder zo een algoritme.

4.1 Bisectie-methode

De bisectie-methode werkt aan de hand van de interlacing eigenschap van de eigenwaarden van een tridiagonale, symmetrische matrix en de iteratieve methode om nulpunten van een functie te bepalen met de (gelijknamige) bisectie-methode. Hiermee kan je makkelijk de reële eigenwaarden zoeken binnen een bepaald interval.

4.1.1 Interlacing-eigenschap

Neem een symmetrische, tridiagonale matrix $A \in \mathbb{R}^{m \times m}$ die irreduceerbaar is (geen nullen op sub-diagonaal). Stel $A^{(1)}, \dots, A^{(m)}$ de boven linkse submatrices van dimensie $1 \dots m$. Neem nu de eigenwaarden van matrix $A^{(k)} : \lambda_1^{(k)} < \lambda_2^{(k)} < \dots < \lambda_k^{(k)}$
De interlacing eigenschap zegt dan dat de eigenwaarden van $A^{(k)}$ voldoen aan:

$$\lambda_j^{(k)} < \lambda_j^{(k-1)} < \lambda_{j+1}^{(k)} \tag{8}$$

Deze eigenschap is cruciaal in de implementatie van de bisectie-methode, dit kunnen we zo inzien: Als we de eigenwaarden van $A^{(k-1)}$ weten, kennen we automatisch de intervallen waarin we de eigenwaarden van $A^{(k)}$ moeten zoeken, dit komt neer op de nulpunten zoeken van de karakteristieke polynoom van $A^{(k)}$ binnen die gekende intervallen.
We weten namelijk dat als we de eigenwaarden van $A^{(k-1)} : \lambda_1^{(k-1)} < \lambda_2^{(k-1)} < \dots < \lambda_k^{(k-1)}$ kennen en we zoeken de eigenwaarden binnen het gesloten interval

$[a, b]$, dan:

$$\begin{aligned} a \leq & \lambda_1^{(k)} < \lambda_1^{(k-1)} \\ \lambda_1^{(k-1)} < & \lambda_2^{(k)} < \lambda_2^{(k-1)} \\ & \vdots \\ \lambda_k^{(k-1)} < & \lambda_{k+1}^{(k+1)} \leq b \end{aligned}$$

We tonen dit nog eens aan in een simpel voorbeeld: Neem de tridiagonale symmetrische matrix A:

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

Nemen we de eerste submatrix $A^{(1)}$ dan is de eigenwaarde vanzelfsprekend gelijk aan 1. Voor de tweede submatrix $A^{(2)}$ kan de karakteristieke polynoom makkelijk bepaald worden:

$$det(A^{(2)} - \lambda I) = x^2 - 4x - 1$$

Met nulpunten (dus eigenwaarden) -0.2361 en 4.2361. Dewelke voldoen aan de interlacing eigenschap. Voor de matrix $A^{(3)} = A$ is de karakteristieke polynoom nu:

$$det(A - \lambda I) = x^3 - 9x^2 + 3x + 21$$

Dus zijn de eigenwaarden -1.2902, 1.9520 en 8.3382, deze voldoen nu ook aan de interlacing eigenschap.

4.1.2 Implementatie van het algoritme

4.1.3 Vergelijking met het QR-algoritme

4.1.4 Problemen met de implementatie: recursie van het polynomiaal

Bij de implementatie van de bisectie-methode in MATLAB kwamen we tegenover enkele interessante problemen te staan. Een eerste was dat het programma exponentieel vertraagde naarmate de matrix groter werd. Rond n=25 duurde een volledige oplossing al enkele seconden en n=30 duurde zolang dat het niet echt mogelijk was om nog grotere matrices te berekenen. Deze exponentieel stijgende rekentijd klopt natuurlijk niet met de bewering dat de bisectie methode werkt volgens $O(nlog(\epsilon_{mach}))$. Na grondig de code na te kijken bleek dat doordat we de polynoom als een functie bewaren, elke keer dat de functie opgeroepen wordt, 2 andere functies uit de vorige stappen worden opgeroepen om een nieuw resultaat te geven. Omdat hun functiewaarden nergens worden bewaard leidt dit tot $O(2^n)$ operaties, waardoor het dus onmogelijk wordt om matrices groter dan dimensie 30 uit te rekenen. De oplossing voor dit probleem was de functies te vervangen door lijsten die polynomen voorstellen en deze polynomen dan samen te voegen zodat de berekening van een eigenwaarde voor een gegeven matrix weer $O(nlog(\epsilon_{mach}))$ wordt.

5 Iteratieve methoden

5.1 De Arnoldi methode