Numerik MA0008: Zusammenfassung

Jonas Treplin

February 14, 2023

1 Grundlagen

Theorem 1 (Satz von Gerschgorin) *Sei $(a_{ij}) = A \in \mathbb{R}^{n\times n}$ dann sind die Eigenwerte von A enthalten in $\bigcup_{i=1}^{n} S_i \subset \mathbb{C}$, dabei sind die $S_i := K(a_{ii}, \sum_{j=1, i\neq j}^{n})$. Wobei mindestens ein Eigenwert jeder Zusammenhangskomponente zugeordnet ist.*

2 Matrixfaktorisierung

Theorem 2 *Die Permutationsmatrizen, die unitären Matrizen, die invertierbaren Matrizen und die unteren/oberen Dreiecksmatrizen bilden jeweils unter Matrixmultiplikation eine Gruppe. Insbesondere sind ihre Inverse von der selben Klasse von Matrizen.*

Gleichungssysteme für die unitären Matrizen (und damit auch Permutationsmatrizen) lassen sich einfach durch adjungieren lösen. Für untere und obere Dreiecksmatrizen existieren Vorwärts- und Rückwärtssubsitution. Diese sind aus dem Endschritt des Lösens von Gleichungssystemen mit dem Gauß-Algorithmus bekannt. Glücklicherweise kann jede invertierbare Matrix (fast eindeutig)

Algorithm 1 Vorwärtssubsitution (Lösen einer unteren Dreiecksmatrix)

Require: $(l_{ij}) = L \in \mathbb{R}^{n\times n}$ Untere Dreiecksmatrix, $b \in \mathbb{R}^n$.
  for $i \in 1 : n$ **do**
    $x_i \leftarrow \frac{1}{l_{ii}}(b_i - \sum_{j=1}^{i-1} l_{ij} * x_j)$
  end for

Algorithm 2 Rückwärtsssubsitution (Lösen einer oberen Dreiecksmatrix)

Require: $(u_{ij}) = U \in \mathbb{R}^{n\times n}$ Obere Dreiecksmatrix, $b \in \mathbb{R}^n$.
  for $i \in n : 1$ **do**
    $x_i \leftarrow \frac{1}{u_{ii}}(b_i - \sum_{j=i+1}^{n} u_{ij} * x_j)$
  end for

in solche Matrizen zerlegt werden. Dies geschieht Wahlweise durch eine LU-Zerlegung oder eine QR-Zerlegung. Mit Pivots erreicht man, dass jede invertierbare Matrix $A \in GL(n)$ zerlegt werden kann, sodass $PA = LU$. Dazu wählt man in jedem Schritt i die Zeile $j = \arg\max_{j\geq i} |a^i_{ji}|$ und vertauscht diese mit der i-ten Zeile.

Algorithm 3 LU-Zerlegung ohne Pivots

Require: $(a_{ij}) = A \in GL(n)$
  for $i \in 1 : n$ **do**
    $l_{ii} \leftarrow 1$
    for $j \in i + 1 : n$ **do**
      $l_{ji} \leftarrow -\frac{a_{ji}}{a_{ii}}$
      for $k \in i : n$ **do**
        $u_{jk} \leftarrow u_{jk} - a_{ji}a_{ji}$
      end for
    end for
  end for

Theorem 3 (LU-ZErlegung mit Pivots) *Sei $A \in GL(n)$. Dann existieren eine eindeutige Permutationsmatrix P, sowie untere (obere) Dreiecksmatrix L (U, sodass $PA = LU$. Dabei ist L normiert also $l_{ii} = 1$.*

Theorem 4 (Cholesky-Zerlegung) *Sei A symmetrisch positiv definit dann lässt sich eine nicht normierte untere Dreiecksmatrix $\tilde{L}$ finden, sodass $A = \tilde{L}\tilde{L}^T$.*

Algorithm 4 Berechnung der Cholesky Zerlegung

Require: A s.p.d.
  $L, U \leftarrow$ LU_Zerlegung(A)
  $D = (u_{ii})$ Diagonalmatrix.
  $\tilde{L} = \sqrt{D}L$.

Eine weitere Möglichkeit ist die der QR Zerlegung.

Definition 1 (Givensrotation) *Für ein $a \in \mathbb{R}^2$ sei $Q = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$. Wobei $c = \frac{1}{\sqrt{1+\tau^2}}$ und $s = c\tau$ mit $\tau = \frac{v_2}{v_1}$ wenn $|v_1| \geq |v_2|$ und $s = \frac{1}{\sqrt{1+\tau^2}}$ und $c = s\tau$ mit $\tau = \frac{v_1}{v_2}$ wenn $|v_1| < |v_2|$.*

Diese Fallunterscheidung ist so gewählt, dass $||Q|| \leq 1$, damit sich Rundungsfehler nicht akkumulieren.

Theorem 5 (Givens-Rotation) *Es gilt $Qa = \xi e_1$.*

Algorithm 5 QR-Zerlegung mit Givens-Rotationen

Require: $A \in \mathbb{R}^{n \times m}$
  $Q \leftarrow I_n$
  for $i \in [n]$ **do**
    for $j \in (n, ..., i+1)$ **do**

$$G := \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & s & & & \\ & & -s & c & & & \\ & & & & \ddots & & \\ & & & & & 1 & \end{bmatrix} \leftarrow Givens(\begin{Bmatrix} A_{ji-1} \\ A_{ji} \end{Bmatrix})$$

      $Q \leftarrow GQ$
      $A \leftarrow GA$
    end for
  end for
  $Q \leftarrow Q*$

Definition 2 (Householder Spiegelung) *Die Householder Spiegelung für einen Vektor $a \in \mathbb{R}^n$ ist:*

$$Q := Id - \frac{2}{v^T v} v v^T$$

Wobei $v := a + sign(a_1)||a||e_1$ Sie erfüllt ebenfalls $Qa = \alpha e_1$

Algorithm 6 QR-Zerlegung mit Householder Rotationen

Require:
Require: $A \in \mathbb{R}^{n \times m}$
  $Q \leftarrow I_n$
  for $i \in [n]$ **do**

$$H \leftarrow \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & \text{Householder}((a_i, ..., a_n)) & \end{bmatrix}$$

    $Q \leftarrow HQ$
    $A \leftarrow HA$
  end for
  $Q \leftarrow Q*$

Die QR-Zerlegung ist der LU-Zerlegung hinsichtlich numerischer Stabilität überlegen, besonders bei Betrachtung der Wilkinson-Matrix:

Definition 3 (Wilkinson-Matrix) *Die Wilinson-Matrix ist definiert als:*

$$W_n := \begin{bmatrix} 1 & 0 & ... & 0 & 1 \\ -1 & \ddots & ... & \vdots & \vdots \\ \vdots & & & \ddots & \vdots \\ -1 & ... & ... & \dots & 1 \end{bmatrix}$$

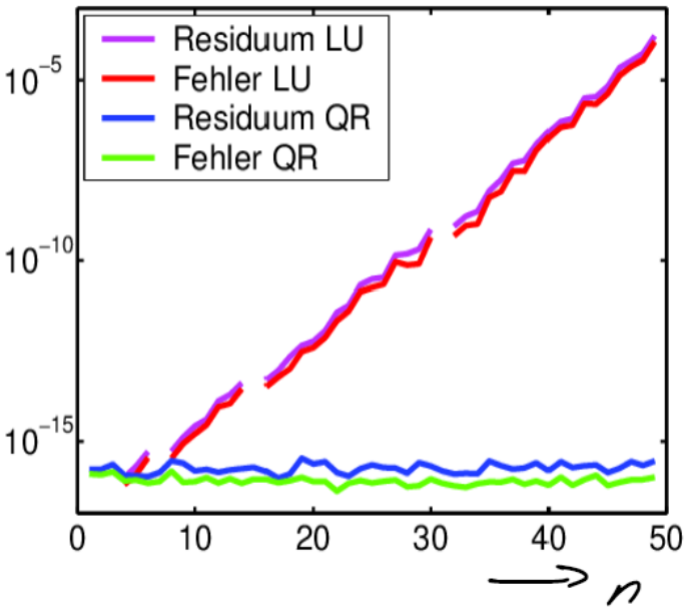

Figure 1: Fehler beim Lösen von $W_n x = b_n$

Ein besonders instabiler Lösungsvektor ist $b_n = \begin{bmatrix} 0 \\ \frac{1}{n} \\ \vdots \\ \frac{n-2}{n} \\ 1 \end{bmatrix}$

3 Fehlerrechnung

Definition 4 (Fehlermaße) *Wir definieren für eine Tupel* $T = (A_1, A_2, ..., A_n)$ *mit Störung* $\tilde{T} = (A_+ E_1, ..., A_n + E_n)$*:*

- *das absolute Fehlermaß:*

$$[[E]]_{abs} := \max ||E_i||$$

- *das relative Fehlermaß:*

$$[[E]]_{rel} := \max \frac{||E_i||}{||A_i||}$$

Definition 5 (Maschinenepsilon) *Das* ***Maschinen-***ϵ *is der relative Fehler, der bei Addition und Multiplikation von SKalaren auftritt. Er liegt für IEEE double-precision bei ca.* 10^{-16}

Definition 6 (Kondition) *Die Kondition einer Abbildung f im Punkt x ist definiert als:*

$$\kappa(f,x) = \limsup_{y \to x} \frac{[[f(y) - f(x)]]}{||y - x||}$$

Man unterscheidet zwischen:

- ***gut konditionierten** Problemen: $\kappa(f,x)$ $O(1)$*
- ***schlecht konditionierten** Problemen: $\kappa(f,x) >> 1$*
- ***schlecht gestellten** Problemen: $\kappa(f,x) = \infty$*

Theorem 6 *DIe Kondition einer linearen Gleichung $Ax = b$ hängt nur von A ab und ist:*

$$\kappa(A) = ||A||||A^{-1}||$$

4 Appendix: Matlab Implementationen

```
function x=backwardsub(U, b)
        n = length(b);
        x = zeros(n, 1);
        for i=n:-1:1
                x(i) = (b(i) - U(i, i+1:n)*x(i+1:n))/U(i, i);
        end
end
function x=forwardsub(L, b)
        x = zeros(1, length(b));
        x(1) = b(1)/L(1, 1);
        for i=2:length(b)
                x(i) = (b(i) - L(i, 1:i-1)*x(1:i-1)')/L(i,i);
        end
end
function [L, U]=lu_decomp(A)
        n = size(A, 1);
        for i=1:n
                if A(i, i) == 0
                        error("Alle Pivotelemente muessen ungleich 0 sein")
                end
                A(i+1:n,i) =  A(i+1:n,i)/A(i, i);
                A(i+1:n,i+1:n) = A(i+1:n,i+1:n) - A(i+1:n,i)*A(i, i+1:n);
        end
        U = triu(A);
        L = tril(A);
        for i=1:n
                L(i, i)=1;
        end
end
function [L, U, p]=lu_decomp_pivot(A)
        n = size(A, 1);
        p = 1:n;
```

```
        for i=1:n
            % pivot search
            [~, j] = max(abs(A(i:n,i))); % search for the argmax in A^k
            j = j + i-1; % correct the index for the global matrix A
            p([i j]) = p([j i]); A([i j],:) = A([j i],:);
            % elimination step
            A(i+1:n,i) = A(i+1:n,i)/A(i, i);
            A(i+1:n,i+1:n) = A(i+1:n,i+1:n) - A(i+1:n,i)*A(i, i+1:n);
            end
        U = triu(A);
        L = tril(A);
        for i=1:n
            L(i, i)=1;
        end
end
```