

Numerik MA0008: Zusammenfassung

Jonas Treplin

February 16, 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 $S_i := K(a_{ii}, \sum_{j=1, j \neq i}^n |a_{ij}|)$. 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 Rückwärtssubstitution. Diese sind aus dem Endschrift des Lösen von Gleichungssystemen mit dem Gauß-Algorithmus bekannt. Glücklicherweise kann jede invertierbare Matrix (fast eindeutig)

Algorithm 1 Vorwärtssubstitution (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 Rückwärtssubstitution (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 $j = \arg \max_{j \geq i} |a_{ji}^i|$ 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
       $u_{jk} \leftarrow u_{jk} - a_{ji}a_{ki}$ 
    end for
  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.

```
 $L, U \leftarrow \text{LU\_Zerlegung}(A)$   
 $D = (u_{ii})$  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** $j \in (n, \dots, i+1)$ **do**

$$G := \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & c & s & \\ & & -s & c & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix} \leftarrow \text{Givens}\left(\begin{bmatrix} A_{j-1,i} \\ A_{j,i} \end{bmatrix}\right)$$

 $Q \leftarrow GQ$ $A \leftarrow GA$ **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 $v := a + \text{sign}(a_1) \|a\| e_1$ 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**

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

 $Q \leftarrow HQ$ $A \leftarrow HA$ **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 Wilkinson-Matrix ist definiert als:

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

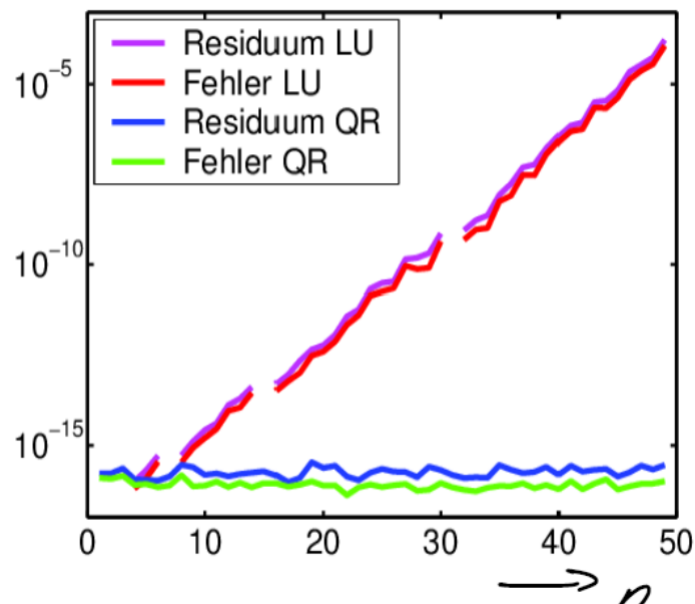


Figure 1: Fehler beim Lösen von $W_n x = b_n$

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 $T = (A_1, A_2, \dots, A_n)$ mit Störung $\tilde{T} = (A_1 + E_1, \dots, A_n + E_n)$:

- 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** ist 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:

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

Man unterscheidet zwischen:

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

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

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

Theorem 7 (Kondition einer C^1 -Funktion) Sei $f \in C^1(D)$, $D \subset \mathbb{R}^n$ mit $f(x) \neq 0$, dann gilt für die Kondition:

$$\kappa(f, x) = \frac{\sum_{i=1}^n |\partial_i f(x)|}{|f(x)|}$$

Definition 7 (Vorwärts- und Rückwärtsfehler) Sei \hat{f} eine numerische Näherung der Funktion f , dann ist an x :

1. Der **Vorwärtsfehler:**

$$[f(x) - \hat{f}(x)]$$

2. Der **Rückwärtsfehler:**

$$\inf_{\Delta x} \{[\Delta x] | f(x + \Delta x) = \hat{f}(x)\}$$

Definition 8 (Stabilität) Ein Algorithmus \hat{f} zur Näherung von f heißt **stabil** an x bezüglich des Fehlermaßes $[\cdot]$ falls ein \tilde{x} existiert mit:

$$[x - \tilde{x}] = O(\epsilon_m)$$

$$[\hat{f}(x) - f(\tilde{x})] = O(\epsilon_m)$$

Dabei sei ϵ_m das Maschinenepsilon.

Definition 9 (Rückwärtsstabilität) Eine Näherung \hat{f} zu f heißt **rückwärtsstabil** an x bezüglich des Fehlermaßes $[\cdot]$ falls ein $\tilde{x} = x + \Delta x$ existiert mit:

$$[x - \tilde{x}] = O(\epsilon_m)$$

$$\hat{f}(x) = f(\tilde{x})$$

Theorem 8 (Fehler rückwärtsstabiler Algorithmen) Der Fehler eines Rückwärtsstabilen Algorithmus \hat{f} hängt nur von der Kondition von f ab:

$$[\hat{f}] = [f(\tilde{x}) - f(x)] \leq c\kappa(f, x)\epsilon_m$$

Theorem 9 (Rückwärtsfehler beim Lösen linearer Gleichungssysteme)

Sei $Ax = b$ ein zu lösendes Gleichungssystem. Definiere

$$w(\tilde{x}) = \inf_E \left\{ \frac{\|E\|_2}{\|A\|_2} : (A + E)\tilde{x} = b \right\}$$

als den relativen Rückwärtsfehler bezüglich der Matrix A . Dann gilt:

$$w(\tilde{x}) = \frac{\|b - A\tilde{x}\|_2}{\|A\|_2 \|\tilde{x}\|_2}$$

Theorem 10 (Stabilität der Matrixmultiplikation) Sei $MN = A$ und \tilde{M}, \tilde{N} fehlerbehaftet mit relativen Fehler kleiner als (δ) , dann ist:

$$\|\tilde{M}\tilde{N} - A\| \leq (2\delta + \delta^2)\|M\|\|N\|$$

. Daraus folgt, dass die QR-Zerlegung stabil bezüglich des relativen Fehlermaßes ist:

$$\frac{\|\tilde{Q}\tilde{R} - A\|}{\|A\|} \leq \frac{1\|A\|}{\|A\|} = 1$$

Ähnliches gilt auch für die Cholesky-Zerlegung, jedoch nicht für die LU-Zerlegung

Algorithm 7 Nachiteration

Require: Gleichungssystem $Ax = b$

Zerlege $LU = PA$

Berechne $LUx^{(0)} = Pb$ durch Vorwärts-/Rückwärtssubstitution

for $i = 1, \dots, k$ **do**

 Berechne $LU\Delta x^{(i)} = P(b - Ax^{(i-1)})$

$x^{(i)} \leftarrow x^{(i-1)} + \Delta x^{(i)}$

end for

4 Ausgleichsrechnung

Definition 10 (Lineares Ausgleichsproblem) Sei $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$ mit $m \leq n$. Gesucht ist $x \in \mathbb{R}^m$. Sodass

$$\|Ax - b\|_2$$

minimiert wird. Es wird im Allgemeinen angenommen, dass $\text{Rang}(A) = m$.

Definition 11 (Normalengleichung) Gegeben ein Ausgleichsproblem A, b , nennt man:

$$A^T Ax = A^T b$$

die Normalengleichung.

Theorem 11 (Lösung des Linearen Ausgleichproblems mittels Normalengleichung)

Die Lösung der Normalengleichung ist das eindeutige gesuchte Minimum des Ausgleichsproblems.

Algorithm 8 Lösen des Ausgleichproblems mittels Normalengleichung

Require: Ausgleichsproblem $A \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^n$

$L \leftarrow \text{Cholesky}(A^T A)$

Löse $LL^T x = A^T b$ durch Vorwärts und Rückwärtssubstitution.

Algorithm 9 Lösen des Ausgleichproblems mittels QR-Methode

Require: Ausgleichsproblem $A \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^n$

Zerlege $A = Q\hat{R}$ mit $\hat{R} = \begin{bmatrix} R \\ 0 \end{bmatrix}$ wobei $R \in \mathbb{R}^{m \times m}$ obere Dreiecksmatrix sei.

Löse $Rx = (Q^T b)_1$ wobei $(\cdot)_1$ die ersten m Elemente seien.

Die Stabilität des Algorithmus hängt von der Stabilität der Matrixmultiplikation $A^T A$ ab. Falls $\kappa(A^T A)$ groß ist und $\|Ax - b\|$ klein treten hier Stabilitätsprobleme auf. Um diesen entgegen zu wirken, kann man die Orthogonalisierungsmethode verwenden.

Theorem 12 (Aufwand der Lösungsmethoden) *Der Aufwand beträgt:*

1. Normalengleichung: $nm^2 + \frac{m^3}{3}$.

2. QR-Methode: $2nm^2 - 2\frac{m^3}{3}$

Für $n \gg m$ ist also die QR-Methode doppelt so teuer wie der Ansatz der Normalengleichung.

Theorem 13 (Kondition der Normalengleichung) Sei $A \in \mathbb{R}^{n \times m}$ und R wie in der Zerlegung für die QR-Methode. Es gilt:

$$\kappa(A^T A) = \kappa(R)^2$$

5 Eigenwertapproximation

Eine einfache Idee um einen einzelnen Eigenwert mit Eigenwert einer Matrix $A \in \mathbb{R}^{n \times n}$ zu bestimmen, ist die Vektoriteration. Diese beruht darauf, dass Eigenräume hoffentlich anziehende Fixpunkte sind.

Algorithm 10 Vektoriteration

Require: $A \in \mathbb{R}^{n \times n}$ und Startvektor $x^{(0)} \in \mathbb{R}^n$.

for $i = 1, 2, 3, \dots$ **do**

$y^{(i)} \leftarrow Ax^{(i-1)}$

$\lambda^{(i-1)} \leftarrow (x^{(i-1)})^T y^{(i)}$

$x^{(i)} \leftarrow \frac{y^{(i)}}{\|y^{(i)}\|}$

end for

Theorem 14 (Konvergenz der Vektoriteration) Sei $A \in \mathbb{R}^{n \times n}$ mit EW $|\lambda_n| \leq \dots \leq |\lambda_2| < |\lambda_1|$ und EV $(v_i)_{i \in [n]}$. Dann gilt:

$$|\lambda^{(i)} - \lambda_1| \leq C_1(x^{(0)}) \left| \frac{\lambda_2}{\lambda_1} \right|^{2i}$$

$$\| \text{sign}(\lambda_1)^i x^{(i)} - \text{sign}(\beta_1) v_1 \|_2 \leq C_2(x^{(0)}) \left| \frac{\lambda_2}{\lambda_1} \right|^i$$

Wobei $\beta_1 := v_1^T x^{(0)}$ und C_1, C_2 Konstanten sind die vom Startwert abhängen

Durch diese Methode lässt sich nur ein einzelner Eigenvektor und auch nur der zum Betragsmäßig größten Eigenwert bestimmen. Um andere Eigenvektor zu berechnen, nutzen wir, dass der Betragsmäßig größte Eigenwert von $(A - \mu I)^{-1}$ der Kerwehr des nächsten Eigenwerts an μ ist.

Algorithm 11 Inverse Vektoriteration

Require: $A \in \mathbb{R}^{n \times n}$ und Startvektor $x^{(0)} \in \mathbb{R}^n$, sowie shift $\mu \in \mathbb{R}$.

for $i = 1, 2, \dots$ **do**
 $(A - \mu I)\omega^{(i)} = x^{(i-1)}$
 $\eta \leftarrow \|\omega^{(i)}\|_2$
 $x^{(i)} \leftarrow \omega^{(i)} / \eta$
 $\rho \leftarrow x^{(i)T} x^{(i-1)} / \eta$
 $\lambda^{(i)} \leftarrow \mu + \rho$
end for

Der Aufwand hängt hier davon ab, wie oft der Shiftparameter μ neu berechnet wird, dann ist jedes Mal eine LU-Zerlegung mit $O(n^3)$ Schritten nötig. Ansonsten kostet jeder Schritt $O(n^2)$ Operation hauptsächlich für Vorwärts und Rückwärtssubstitution.

Um alle Eigenwerte gleichzeitig zu bestimmen kann iterativ eine Schur-Zerlegung berechnet werden. Dies geschieht mit der QR-Iteration.

Algorithm 12 QR-Iteration ohne Shift

Require: $A \in \mathbb{R}^{n \times n}$

$A_0 \leftarrow Q_0^T A Q_0$ mit $Q \in U(n)$.
for $i = 1, 2, 3, \dots$ **do**
Bestimme $Q_i R_i = A_{i-1}$
 $A_i \leftarrow R_i Q_i$
end for

Der Algorithmus beruht auf der Tatsache, dass $RQ \sim A$ also die gleichen EW wie A hat.

Theorem 15 (Konvergenz der QR-Iteration) Sei $A \in \mathbb{R}^n$ symmetrisch und mit EW $|\lambda_1| > |\lambda_2| > \dots > |\lambda_m| > 0$. Sei $A = V \Lambda V^T$ diagonalisiert mit V orthogonal. Besitzt $(Q_0^T V)^{-1}$ eine normierte LU-Zerlegung, dann gilt:

1. $\lim_{k \rightarrow \infty} |(Q_k)_{ij}| = \delta_{ij}$
2. $\lim_{k \rightarrow \infty} |(R_k)_{ij}| = \delta_{ij} |\lambda_i|$
3. $\lim_{k \rightarrow \infty} (A_k)_{ij} = \delta_{ij} \lambda_i$

Der Beweis kann auf schwächere Bedingungen ausgeweitet werden zum Beispiel sind mehrfache Eigenwerte auch erlaubt. Häufig kann die QR-Iteration auch auf nicht-symmetrische Matrizen angewendet werden, wenn sie in diesem Fall

konvergiert dann allerdings nicht mehr gegen eine Diagonal-, sondern obere Dreiecksmatrix.

Der erste Schritt die Transformation mit Q_0 transformiert die Matrix in eine obere Hessenberg Matrix. Für eine Obere Hessenberg-Matrix ist die QR-Zerlegung in $O(n^2)$ berechenbar. Da auch ein einzelner Iterationschritt die Hessenberg Form erhält beschleunigt dies das Verfahren enorm.

Algorithm 13 Berechnung von Q_0

Require: $A \in \mathbb{R}^{n \times n}$

$Q_0 \leftarrow Id$

for $i = 1, \dots, n - 1$ **do**

for $j = n, i + 2$ **do**

$$G := \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & c & s & & \\ & & -s & c & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix} \leftarrow \text{Givens} \left(\begin{bmatrix} A_{j-1,i} \\ A_{j,i} \end{bmatrix} \right)$$

$A \leftarrow GA$

$Q_0 \leftarrow GQ_0$

end for

end for

Weiterhin kann ein Shiftparameter eingeführt werden um das Verfahren zu beschleunigen idealerweise liegt dieser Shiftparameter möglichst nah an einem Eigenwert der Matrix A .

Algorithm 14 QR-Iteration mit Shift

Require: $A \in \mathbb{R}^{n \times n}$

$A_0 \leftarrow Q_0^T A Q_0$ mit $Q \in U(n)$.

for $i = 1, 2, 3, \dots$ **do**

$\mu_i \leftarrow \text{Shift}(A_i)$

 Bestimme $Q_i R_i = A_{i-1} - \mu_i I$

$A_i \leftarrow R_i Q_i + \mu_i I$

end for

Dabei können verschiedene Shiftstrategien angewandt werden:

1. **Rayleighquotienten-Shift:** $\mu_i := (A_i)_{n,n}$
2. **Wilkinson-Shift:** Bestimme die Eigenwerte $\hat{\lambda}_1, \hat{\lambda}_2$ von $\begin{bmatrix} (A_i)_{n-1,n-1} & (A_i)_{n-1,n} \\ (A_i)_{n,n-1} & (A_i)_{n,n} \end{bmatrix}$.
Wähle $\mu_i = \arg \max_{\mu \in \{\hat{\lambda}_1, \hat{\lambda}_2\}} |\mu - (A_i)_{n,n}|$
3. **Random-Shift:** Wähle zufällig einen Shift aus.

Der Rayleighshift produziert bei $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ als shift 0 und landet somit in

einem Deadend. Ähnlich schlägt der Wilkinson-Shift bei $A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ fehl.

In diesen Situationen kommt der Random-Shift zum Einsatz.

Weiterhin kann das Verfahren beschleunigt werden, indem man, sobald $|(A_i)_{n,n-1}| \leq \text{TOL}$ nur noch mit $(A_i)_{1:m-1,1:m-1}$ weiterrechnet. Diese Strategie heißt **Deflation**.

Um aus den berechneten Eigenwerten Eigenvektoren zu erhalten, kann man für wenige Eigenwerte die Inverse Vektoriteration mit Shift verwenden. Alternativ lassen sich die aus der QR-Zerlegung berechneten Q_i verwenden.

Theorem 16 (Eigenvektoren einer oberen Dreiecksmatrix) Sei $R \in \mathbb{R}^n$ obere Dreiecksmatrix. Definiere:

$$R^{(i)} := (R)_{1:i-1,1:i-1}$$

$$\omega := (R)_{1:i-1,i}$$

Dann hat der i -te Eigenvektor von R die Form:

$$v_i = \begin{pmatrix} (R^{(i)})^{-1}\omega \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Theorem 17 (Eigenvektoren der QR-Zerlegung) Seien $(Q_i)_{i \in [k]}$ die berechneten Q_i einer QR-Iteration auf A mit Q_0 der Umformungsmatrix für die Hessenberg-Form. Dann ist

$$A_k = Q_k^T \dots Q_1^T Q_0^T A Q_0 Q_1 \dots Q_k$$

Wir definieren $\hat{Q}_k := Q_0 Q_1 \dots Q_k$. Bei genügend hohem k sollte A_k fast gleich einer oberen Dreiecksmatrix \hat{R}_k sein. Deren Eigenvektoren lassen sich berechnen nach dem obigen Satz. Dann sind die $\hat{Q}_k v_i$ die Eigenvektoren von A .

Theorem 18 (Kondition des Eigenwertproblems für beliebige Matrizen) Sei $A \in \mathbb{R}^{n \times n}$. Dann existiert eine Konstante C_A sodass für jede Störung ΔA :

$$\forall \mu \in \sigma(A + \Delta A) \exists \lambda_\mu \in \sigma(A) : |\mu - \lambda_\mu| \leq C_A \max\{\|\Delta A\|_2, \|\Delta A\|_2^{\frac{1}{n}}\}$$

Theorem 19 (Bauer-Fike, Kondition des Eigenwertproblems für diagonalisierbare Matrizen) Sei $A \in \mathbb{R}^{n \times n}$ reell diagonalisierbar durch $T^{-1}DT$, dann existiert für jede Störung ΔA und jedes $\mu \in \sigma(A + \Delta A)$ ein $\lambda_\mu \in \sigma(A)$ mit

$$|\mu - \lambda_\mu| \leq \|T\|_p \|T^{-1}\|_p \|\Delta A\|_p$$

Für alle $p \in [0, \infty]$.

6 Interpolation

Definition 12 (Interpolation) Gegeben Stützpunkte $(x_1, f_1), \dots, (x_n, f_n)$ finde f in einer bestimmten Klasse von Funktionen, sodass $f(x_i) = f_i \quad \forall i$

Die bekanntesten Formen von Interpolation sind:

1. **Polynominterpolation:** $f \in \mathbb{P}^n$
2. **Spline-Interpolation:** f ist stückweise polynomiell und gesamt C^l .
3. **Rationale Interpolation:** $f \in R_{k,l} \{ \frac{\sum_{i=0}^k a_i x^i}{\sum_{i=0}^l b_i x^i} \mid a_i, b_i \in \mathbb{R} \}$
4. **Trigonometrische Interpolation:** $f \in T_n := \{ b_0 + \sum_{k=1}^n a_k \sin(2\pi kx) + \sum_{k=1}^n b_k \sin(2\pi kx) \}$

Definition 13 (Lagrange-Polynom) Zu den $n+1$ verschiedenen Stützstellen x_0, \dots, x_n sind die Lagrange-Polynome von Grad n definiert als:

$$L_i(x) := \prod_{k=0, k \neq i}^n \frac{x - x_k}{x_i - x_k}$$

Theorem 20 (Polynominterpolation) Das Interpolationspolynom π_n zu den Stützstellen x_0, \dots, x_n ist eindeutig definiert als:

$$\pi_n(x) = \sum_{i=0}^n f_i L_i(x)$$

Theorem 21 Die Lagrange Polynome erfüllen die Partition der 1:

$$\sum_{i=0}^n L_i(x) = 1$$

Definition 14 (Vandermonde-Matrix) Die Matrix $V_n \in \mathbb{R}^{(n+1) \times (n+1)}$ mit Einträgen:

$$(V_n)_{i,j} = x_{i-1}^{j-1}$$

Heißt Vandermonde-Matrix zu den Stützstellen x_0, \dots, x_n .

Theorem 22 Es gilt:

$$\det V_n = \prod_{0 \leq i < j \leq n} (x_j - x_i)$$

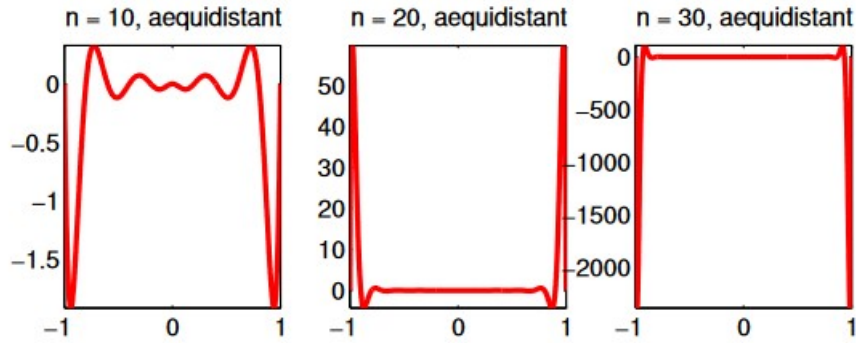
Theorem 23 Das Interpolationspolynom ist auch durch $\sum_{k=0}^n a_k x^k$ gegeben

$$\text{wobei } \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} =: a = V_n \begin{pmatrix} f_0 \\ \vdots \\ f_n \end{pmatrix}$$

Definition 15 (Knotenpolynom) Zu den $n+1$ Stützstellen x_0, \dots, x_n definiert man das Knotenpolynom

$$\omega_{n+1}(x) = \prod_{i=0}^n (x - x_i)$$

Figure 2: Interpolations Fehler für π_n



Theorem 24 Falls der Interpoland f aus C^{n+1} kommt ist der Interpolationsfehler beschränkt durch:

$$\|f - \pi_n\| \leq \frac{\|\omega_{n+1}\|}{(n+1)!} \|f^{(n+1)}\|$$

Theorem 25 Es gilt für jede beliebige Wahl von Punkten:

$$\|\omega\| \geq 2^{-n}$$

Bei äquidistanten Stützstellen treten beim Interpolieren von $f(x) = \frac{1}{1+25x^2}$ große Fehler in den Randbereichen auf. Generell lässt sich keine generell optimale Knotenfolge finden. Allerdings kann $\|\omega_n\|$ minimiert werden. Dies geschieht durch die Chebyshev-Punkte:

Definition 16 Die Chebyshev-Polynome sind definiert als:

$$T_n(X) := \cos(n \arccos x)$$

Sie sind orthogonal bezüglich der Gewichtsfunktion $\frac{1}{\sqrt{1-x^2}}$. Sie besitzen die Nullstellen:

$$x_i^{(n+1)} = \cos\left(\frac{2i+1}{2n+2}\pi\right)$$

Diese werden auch als Chebyshev-Punkte bezeichnet.

Theorem 26 (Dreiterm-Rekursion der Chebyshev-Polynome) $T_0 = 1, T_1(x), T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$

Theorem 27 Für das Knotenpolynom $\omega_{n+1, \text{cheb}}$ zu den Chebyshevpunkten gilt:

$$\|\omega_{n+1, \text{cheb}}\| = 2^{-n}$$

Die theoretischen Überlegungen zur Bestimmung des Interpolationspolynom sind numerisch nicht stabil.

6.1 Auswertungsschemata

Algorithm 15 Auswertungsschema von Aitken-Neville

Require: $n + 1$ Stützstellen $(x_0, f_0), \dots, (x_n, f_n)$, Auswertungsstelle y .

Setze $P_i(y) \leftarrow f_i$ für $i = 0, \dots, n$

for $i = 1, \dots, n$ **do**

for $j = 0, \dots, n - i$ **do**

$$P_{j \dots j+i}(y) \leftarrow \frac{(y - x_j)P_{j+1 \dots j+i}(y) - (y - x_{j+i})P_{j \dots j+i-1}(y)}{x_{j+i} - x_j}$$

end for

end for

Figure 3: Darstellung des Aitken-Neville-Schema

Aitken-Neville-Schema für $n = 2$, $n = 3$, $n = 4$ und $n = 5$:

	$k = 0$	1	2	3	4	5
x_0	$f_0 = P_0(x)$					
x_1	$f_1 = P_1(x)$	$P_{01}(x)$				
x_2	$f_2 = P_2(x)$	$P_{12}(x)$	$P_{012}(x)$			
x_3	$f_3 = P_3(x)$	$P_{23}(x)$	$P_{123}(x)$	$P_{0123}(x)$		
x_4	$f_4 = P_4(x)$	$P_{34}(x)$	$P_{234}(x)$	$P_{1234}(x)$	$P_{01234}(x)$	
x_5	$f_5 = P_5(x)$	$P_{45}(x)$	$P_{345}(x)$	$P_{2345}(x)$	$P_{12345}(x)$	$P_{012345}(x)$

Theorem 28 (Aufwand des Aitken-Neville-Verfahrens) Für l verschiedene Funktionen, m Auswertungsstellen und $n + 1$ Stützstellen beträgt der Aufwand für das Aitken-Neville-Schema $O(lmn^2)$.

Das Aitken-Neville Schema ist stabil aber langsam, wenn mehrere Stellen ausgewertet werden sollen. Um dies zu beschleunigen kann das

Definition 17 (Newton-Darstellung eines Polynoms) Die Darstellung eines Polynoms in der Form:

$$\pi_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \dots + c_n(x - x_0) \dots (x - x_{n-1})$$

Heißt **Newton'sche Darstellung**.

Algorithm 16 Newton'sche Dividierte Differenzen

Require: $n + 1$ Stützstellen $(x_0, f_0), \dots, (x_n, f_n)$

Setze $f[x_i] := f_i$ für $i = 0, \dots, n$

for $i = 1, \dots, n$ **do**

for $j = 0, \dots, n - i$ **do**

$$f[x_j \dots x_{j+i}] \leftarrow \frac{f[x_{j+1} \dots x_{j+i}] - f[x_j \dots x_{j+i-1}]}{x_{j+i} - x_j}$$

end for

end for

$c_i \leftarrow f[x_0 \dots x_i]$

Algorithm 17 Horner-Schema

Require: Polynom in Newton-Darstellung mit Koeffizienten c_0, \dots, c_n , Auswertungsstelle y .

```
p ← cn
for k = n - 1, ..., 0 do
    p ← p(y - xk) + ck
end for
```

Theorem 29 (Aufwand des Horner Schema) Für l verschiedene Funktionen, m Auswertungsstellen und $n + 1$ Stützstellen beträgt der Aufwand für das Aitken-Neville-Schema $O(l(mn + n^2))$.

Allerdings ist das Horner-Schema **nicht stabil** für große n . Der Zugang über die baryzentrische Darstellung bietet dem Abhilfe und noch größere Effizienz.

Definition 18 (baryzentrische Gewichte) Wir definieren die **baryzentrischen Gewichte** zu den Stützstellen x_0, \dots, x_n als:

$$\lambda_i := \frac{1}{\prod_{k=0, k \neq i}^n (x_i - x_k)} = \frac{1}{\omega'_{n+1}(x_i)}$$

Theorem 30 (baryzentrische Darstellung des Lagrange-Polynoms) Für die Stützstellen $(x_0, f_0), \dots, (x_n, f_n)$ hat das Interpolationspolynom die Form:

$$\pi_n(x) = \omega_{n+1}(x) \sum_{i=1}^n \frac{\lambda_i f_i}{x - x_i} = \frac{\sum_{i=1}^n \frac{\lambda_i f_i}{x - x_i}}{\sum_{i=1}^n \frac{\lambda_i}{x - x_i}}$$

Theorem 31 (Formen der baryzentrischen Gewichte) Die Gewichte haben folgende Form (abhängig von der Wahl der Stützstellen):

- Äquidistant: $(x_i = -1 + ih, h = 2/n)$

$$\lambda_i = \frac{(-1)^{n-i}}{h^n (n-i)! i!}$$

- Chebyshev: $(x_i = \cos(\frac{2i+1}{2n+2}\pi))$

$$\lambda_i = (-1)^i \frac{2^n}{n+1} \sin\left(\frac{2i+1}{2n+2}\right)$$

Theorem 32 Die Berechnung von l Funktionen an m Auswertungsstellen kostet $O(n^2 + lmn)$ Schritte. Außerdem ist der Algorithmus stabil.

6.2 Interpolationsfehler

Definition 19 (Lebesgue-Konstante) Wir definieren die **Lebesgue-Konstante** auf dem Intervall $[-1, 1]$ als:

$$\Lambda_n := \left\| \sum_{i=0}^n |L_i(\cdot)| \right\|_{\infty}$$

Theorem 33 Sei π_n das Interpolationspolynom für f und $\tilde{\pi}_n$ das für \tilde{f} dann gilt:

$$\|\pi_n - \tilde{\pi}_n\| \leq \Lambda_n \|f - \tilde{f}\|$$

Definition 20 (Stetigkeitsmodul) Der **Stetigkeitsmodul** von f ist definiert als:

$$\omega(f, \delta) := \sup_{|x-y|<\delta} |f(x) - f(y)|$$

Theorem 34 Für Lipschitz-stetige f mit Lipschitzkonstante L gilt: $\omega(f, \delta) \leq L\delta$.

Theorem 35 (Interpolationsfehlerschranke durch Stetigkeitsmodul) Es gilt folgende Abschätzung für den Fehler der besten Approximation:

$$\inf_{q \in \mathbb{P}^n} \|f - q\| \leq (1 + \frac{\pi^2}{2}) \omega(f, 1/n)$$

Theorem 36 (Interpolationsfehlerschranke durch Bestapproximation) Sei p^* die beste polynomielle Approximation an f , dann gilt folgende Fehlerabschätzung für das Interpolationspolynom π_n :

$$\|f - \pi_n\| \leq (1 + \Lambda_n) \|f - p^*\|$$

Theorem 37 (Wachstumsschranken der Lebesgue-Konstanten) Die Lebesgue-Konstanten sind durch folgende Schranken begrenzt.

- Für Chebychev-Punkte gilt:

$$\frac{2}{\pi} \ln(n+1) \leq \Lambda_n \leq \frac{2}{\pi} \ln(n+1) + 1$$

- Für äquidistante Punkte gilt:

$$Ce^{n/2} \leq \Lambda_n \sim \frac{2^{n+1}}{en \log n}$$

Theorem 38 (De la Vallée Poussin) Sei $f \in C$ und $p \in \mathbb{P}^n$ außerdem seien x_0, \dots, x_{n+1} sodass: $(f(x_i) - p(x_i))(f(x_{i+1}) - p(x_{i+1})) < 0$. Dann gilt für die Bestapproximation p^* :

$$\|p^* - f\| \geq \min |f(x_i) - p(x_i)|$$

Theorem 39 (Oszillationstheorem) Sei $f \in C$ und $p \in \mathbb{P}^n$ zusätzlich seien x_0, \dots, x_{n+1} , sodass:

- $f(x_i) - p(x_i) = -(f(x_{i+1}) - p(x_{i+1}))$
- $|f(x_0) - p(x_0)| = \|f - p\|$

Dann ist p die Bestapproximation von f in \mathbb{P}^n . Man nennt p auch **MinMax-Polynom**.

6.3 Spline-Interpolation

Definition 21 (Spline) Sei $\Delta = \{a = x_0 < x_1 < \dots < x_n = b\}$ ein Gitter auf $[a, b]$. Dann sind die Splineräume von Grad k definiert als:

$$S_{\Delta,k} := \{v \in C^{k-1}([a, b]) : v|_{[x_i, x_{i+1}]} \in \mathbb{P}^k\}$$

$$S_{\Delta,0} := \{v|_{[x_i, x_{i+1}]} \in \mathbb{P}^0\}$$

Die Spline-Räume sind Vektorräume mit Dimension $n + k$.

Definition 22 (Unterarten des kubischen Splines) Um einen kubischen Spline $s \in S_{\Delta,3}$ zu $f \in C^2$ zu berechnen benötigt man weitere Nebenbedingungen: Dafür gibt es mehrere Möglichkeiten:

- Vollständiger Spline: $s'(x_0) = f'(x_0)$ und $s'(x_n) = f'(x_n)$
- Natürlicher Spline: $s''(x_0) = s''(x_n) = 0$
- Periodischer Spline: $s'(x_0) = s'(x_n)$ und $s''(x_0) = s''(x_n)$

Definition 23 (Momentendarstellung eines kubischen Splines) Für $s \in S_{\Delta,3}$ gilt $s'' \in S_{\Delta,1}$. Seien die N_i^2 Basisfunktionen von $S_{\Delta,1}$. Dann kann s'' dargestellt werden als:

$$s''(x) = \sum_{i=0}^n m_i N_i^2(x)$$

Die m_i heißen **Momente**.

Definition 24 Aus den Momenten kann s berechnet werden als:

$$s(x)|_{[x_{i-1}, x_i]} = c_i d_i \left(x - \frac{x_{i-1} + x_i}{2}\right) + m_i \frac{(x - x_i)^3}{6h_i} - m_{i-1} \frac{(x - x_{i-1})^3}{6h_i}$$

Dabei sei $h_i = x_i - x_{i-1}$ und

$$c_i = \frac{f_i + f_{i-1}}{2} - \frac{h_i^2}{12}(m_i + m_{i-1})$$

$$d_i = \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(m_i - m_{i-1})$$

Die Momente lassen sich folgendermaßen ausrechnen: