Numerik II

(Prof. Arnold – WS 2005/2006)
(by Georg Kuschk)

TODO : Def. L_1-Norm
gram-schmidtsches orth-verf
bulge-chasing verfahren
Horner-Schema zu Bem 9.18 ?

Wiederholung : Komplexe Zahlen

$z = a + ib = r \cdot e^{ij}$, $r =$ Betrag von z , $j = \arg z$

(d.h. $a = r\cos j$ und $b = r\sin j$)

Rechenregeln :

$$(a_1, b_1) \pm (a_2, b_2) = (a_1 \pm a_2, b_1 \pm b_2)$$
$$(a_1, b_1) \cdot (a_2, b_2) = (a_1 a_2 - b_1 b_2 \, , \, a_1 b_2 + a_2 b_1)$$

Dreiecksungleichung :

$\forall z_1, z_2 \in \mathbb{C}$ gilt : $|z_1 + z_2| \le |z_1| + |z_2|$

Eulersche Beziehung :

$$e^{ix} = \cos x + i \sin x$$

!! In $\mathbb{C}$ gibt es keine Ordnungsrelationen . **!!**

6.5 Kondition des Eigenwertproblems

geg.: $A \in C^{n \times n}$

ges.: $l \in C$, $x \in R^n \setminus 0^n$ mit $Ax = lx$

$$k_{abs} = \| l'(A) \| = \frac{\|x_0\| \cdot \|y_0\|}{|\langle x_0, y_0 \rangle|} = \frac{1}{|\cos(\forall (x_0, y_0))|}$$

mit $x_0 \in C^n$ Eigenvektor der Matrix $A \in C^{n \times n}$ und y_0 konjugierter Eigenvektor $\in C^n$

$$k_{abs} = \frac{\|A\|}{|l|} \| l'(A) \|$$???

Bemerkung 6.6 : Normale Matrizen

$A \in C^{n \times n}$ heißt **normal** , wenn $AA^* = A^* A$ $\left(A^* = \overline{A}^T\right)$

Unter Verwendung der Schurschen Normalform zeigt man , dass A genau dann normal ist ,
wenn gilt : $A = U^* \Lambda U$
, mit einer unitären Matrix $U \in C^{n \times n}$, $U^* U = I$, $\Lambda = diag \; l_i$ mit $l_i \in C$

Ist $x_0 \in C^n$ Eigenvektor von A zum einfachen Eigenwert l_0 , so ist $U^* \Lambda U x_0 = l_0 x_0$.

Folgerung 6.7 :

Das Eigenwertproblem für einfache Eigenwerte normaler Matrizen ist gut konditioniert mit $k_{abs} = 1$.

Beispiel 6.8 : schlecht konditionierte EWP

a)

Sei $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ und $B = \begin{pmatrix} 0 & 1 \\ d & 0 \end{pmatrix}$ mit $|d| << 1$

Die Eigenwerte sind $l_{1,2}(A) = 0$ und $l_{1,2}(B) = \pm\sqrt{d}$.

$$\left(0 = \det(lI - B) = \begin{vmatrix} l & -1 \\ -d & l \end{vmatrix} \right)$$

$$k_{abs} \geq \frac{|l_1(B) - l_1(A)|}{\|B - A\|_2} = \frac{\sqrt{d}}{d} = \frac{1}{\sqrt{d}} \to \infty \qquad \text{für } d \to 0$$

b) (Wilkinson)

Sei $P(l) = \prod_{i=1}^{20} (l - i) \in \Pi_{20}$ ($l_i = i$ Nullstellen des Polynoms)

Diese Nullstellen sind gegenüber kleinen Störungen der Polynomkoeffizienten extrem empfindlich.

So hat z.B. $\tilde{P}(l) = P(l) - 2^{-23} l^{19}$ die Nullstellen

$l \approx 16.73 \pm 2.8i$ und $l \approx 19.50 \pm 1.9i$

6.2 Vektoriteration („power method“ / „von Mises“)

Algorithmus :

geg.: $A \in R^{n \times n}$ und Anfangsnäherung $y^{(0)} \in R^n$

Setze $x^{(0)} := \frac{y^{(0)}}{\|y^{(0)}\|_2}$, $k := 0$

Schritt k :

$$y^{(k+1)} := A x^{(k)}$$

$$x^{(k+1)} := \frac{y^{(k+1)}}{\|y^{(k+1)}\|_2}$$

$$n^{(k+1)} := \left(x^{(k)}\right)^T y^{(k+1)} \quad \text{Eigenwert}$$

$$k := k + 1$$

Eigenschaften :

Mittels vollständiger Induktion folgt : $x^{(k)} = \frac{A^k x^{(0)}}{\left\| A^k x^{(0)} \right\|_2}$

(konvergiert gegen den betragsgrößten Eigenvektor – siehe Information Retrieval HITS-Algorithmus)

Sei A diagonalisierbar mit Eigenwerten l_i $\quad |l_1| \geq |l_2| \geq ... \geq |l_n|$

und den Eigenvektoren z_i $\quad (i = 1,...,n)$.

Dann ist $x^{(0)} = \sum_{i=1}^{n} a_i z_i$ $\quad$ mit gewissen $a_i \in C$.

Ist l_0 dominanter Eigenwert von A, d.h. $|l_1| > |l_i|$ $\quad (i = 2,...n)$, so ist insbesondere $l_1 \in R$ und es gilt

$$x^{(k)} = \frac{a_1 l_1^{\,k} z_1 + \sum_{i=2}^{n} a_i l_i^{\,k} z_i}{\left\| a_1 l_1^{\,k} z_1 + \sum_{i=2}^{n} a_i l_i^{\,k} z_i \right\|_2} \qquad \left(A^k \cdot \sum_i a_i z_i = \sum_i a_i \left(A^k z_i \right) = \sum_i a_i l_i^{\,k} \right)$$

$$= \operatorname{sgn}\left(a_1 l_1^{\,k} \right) \frac{z_1 + r^{(k)}}{\left\| z_1 + r^{(k)} \right\|_2} \qquad \text{mit } r^{(k)} := \sum_{i=2}^{n} \frac{a_i}{a_1} \left(\frac{l_i}{l_1} \right)^k z_i \to 0 \text{ für } k \to \infty$$

, wobei $a_1 \neq 0$ vorausgesetzt wird.

Ergebnis :

$x_\infty := \lim_{k \to \infty} x^{(k)}$ ist Eigenvektor zum Eigenwert l_1

und $\lim_{k \to \infty} n^{(k)} = x_\infty^T \underline{A x_\infty} = x_\infty^T l_1 x_\infty = l_1 \left\| x_\infty \right\|_2^2 = l_1$

D.h. , die Vektoriteration berechnet näherungsweise den betragsmäßig größten Eigenwert von A und eines zugehörigen Eigenvektors.

Die Normierung der $x^{(k)}$ verhindert einen Überlauf.

Satz 6.10 : Rayleigh-Quotient

Unter den Voraussetzungen von Bem. 6.9 gilt für einen symmetrische Matrix $A \in R^{n \times n}$:

$$n^{(k+1)} = l_1 \left(1 + O\left(\left(\frac{l_2}{l_1} \right)^{2k} \right) \right)$$

, d.h. die Rayleigh-Quotienten $n^{(k+1)} = \frac{\left(x^{(k)} \right)^T A x^{(k)}}{\left(x^{(k)} \right)^T x^{(k)}}$ konvergieren quadratisch gegen l_1.

Bemerkung 6.11 Inverse Vektoriteration (Wielandt)

geg.: $A \in R^{n \times n}$, $l \in C$ sei Eigenwert von A

$B \in R^{n \times n}$, $B := A - \overline{l} I$, $l - \overline{l}$ Eigenwert von B („shift“ der Matrix A)

A regulär $\Rightarrow$ $C := A^{-1}$, l^{-1} Eigenwert von C

Die formale Anwendung der Vektoriteration auf B^{-1} führt zu einer Approximation des betragskleinsten Eigenwertes einer gegebenen Matrix.

praktisch :

$B := A - \overline{l} I$ mit einer Näherung $\overline{l}$ eines Eigenwertes l_j von A

Eigenwerte von B : $l_i = \overline{l}$ $(i = 1,...,n)$ „shift“

Eigenwerte von B^{-1} : $\frac{1}{l_i - \overline{l}}$ $(i = 1,...,n)$

A, B und B^{-1} haben die gleichen Eigenvektoren.

Für $\overline{l} \approx l_j$ ist $\left| \frac{1}{l_j - \overline{l}} \right| >> \left| \frac{1}{l_i - \overline{l}} \right|$ $\forall i \neq j$

$\Rightarrow$ schnelle Konvergenz der Vektoriteration

Implementierung :

$y^{(k+1)} := B^{-1} x^{(k)}$ $\quad | \cdot B$

$(A - \overline{l} I) y^{(k+1)} = x^{(k)}$

$$\underbrace{\overline{n}^{(k+1)}}_{\approx \text{Eigenwert von } A} := \overline{l} + \frac{1}{\underbrace{n^{(k+1)}}_{\approx \text{Eigenwert von } B^{-1}}}$$

Aufwand :

- Pro Iterationsschritt Lösung eines Gleichungssystems (kubisch)

1x LR-Zerlegung (=LU-Zerlegung) von $A - \overline{l} I$ $\quad O\left(\frac{n^3}{3}\right)$

und pro Iterationsschritt

1x Vorwärtssubstitution $Lz = x^{(k)}$

1x Rückwärtssubstitution $Uy = z$ (bzw. $Ry = z$)

jeweils $O(n^2)$

Problem :

$A - \overline{l} I$ ist sehr schlecht konditioniert für sehr gute Näherungen der Eigenwerte ,
da $A - l_j I$ singulär ist.

Allerdings nicht die Richtung (bleibt genau) , sondern nur der Betrag.
Dieser wird jedoch im weiteren Verlauf der Berechnung nicht benötigt.

Konsequenz : LR-Zerlegung mit Spaltenpivotisierung für $A - \bar{l} I$
Verschwindet ein Pivotelement , so wird es durch eps ersetzt.

Modifikation :

Ersetze $\bar{l}$ für $k > 0$ durch die (bessere) Approximation $\bar{n}^{(k)}$ des Eigenwertes l_j .

$$\left(A - \bar{n}^{(k)} I\right) y^{(k+1)} = x^{(k)} \quad , \quad \bar{n}^{(k+1)} := \bar{n}^{(k)} + \frac{1}{n^{(k+1)}}$$

Im allgemeinen unattraktiv , wegen des hohen Zusatzaufwands zur neuen LR-Zerlegung von $A - \bar{n}^{(k)} I$ in **jedem** Iterationsschritt.

Idee :

Vor Beginn der Iteration die Matrix A durch Ähnlichkeitstransformationen auf einfachere Gestalt bringen.
(Ähnlichkeitstransformationen lassen die Eigenwerte unverändert.)

Bemerkung 6.12 : Transformation auf Hessenberg-Form

Transformationsschritt :

Bestimme zu gegebener Matrix $A = \left(a_{ij}\right) \in R^{n \times n}$ ($A = QR$ siehe Householder)

einen Vektor $\tilde{v}_1 \in R^{n-1}$ so , dass

$$\left(I_{n-1} - 2 \frac{\tilde{v}_1 \tilde{v}_1^T}{\tilde{v}_1^T \tilde{v}_1} \right) \cdot \begin{pmatrix} a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix} = a \cdot \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Mit $v_1 = \begin{pmatrix} 0 \\ \tilde{v}_1 \end{pmatrix} \in R^n$ und $Q_1 := I - 2 \frac{v_1 v_1^T}{v_1^T v_1}$ ist $A^{(2)} := Q_1 A Q_1^T$,

denn die Rechts-Multiplikation mit Q_1^T lässt wegen $v_{1,1} = 0$ die erste Spalte unverändert und transformiert nur die restlichen Spalten.

Die Matrizen $A^{(2)}$ und $A^{(1)} := A$ sind zueinander ähnlich und haben insbesondere dieselben Eigenwerte.

allgemein :

Analog zur QR-Zerlegung von A bestimmt man sukzessive (nacheinander) $n-2$ Spiegelungsvektoren

$v_k = \begin{pmatrix} s_k \\ \tilde{v}_k \end{pmatrix} \in R^n$ mit $\tilde{v}_k \in R^{n-k}$ und die zugehörigen Spiegelungsmatrizen

$Q_k := I - 2 \frac{v_k v_k^T}{v_k^T v_k} \in R^{n \times n}$ so , dass mit $A^{(k+1)} := Q_k A^{(k)} Q_k^T$ die

Matrix $\tilde{A} := A^{(n-2)}$ Hessenberg-Form besitzt.

Mittels vollständiger Induktion folgt , dass $\tilde{A}$ und A zueinander ähnlich sind :
$\tilde{A} := Q A Q^T$ mit $Q = Q_{n-2} Q_{n-1} \ldots Q_2 Q_1$.

Ist $\tilde{x}_i$ Eigenvektor von $\tilde{A}$ zum Eigenwert l_i , so ist $x_i := Q^T \tilde{x}_i = Q_1^T Q_2^T ... Q_{n-2}^T \tilde{x}_i$ Eigenvektor von A zum Eigenwert l_i , denn

$$\tilde{A}\tilde{x}_i = l_i \tilde{x}_i \quad \Leftrightarrow \qquad QAQ^T \tilde{x}_i = l_i \tilde{x}_i \qquad | \cdot Q^T$$
$$A\underbrace{Q^T \tilde{x}_i}_{x_i} = l_i \underbrace{Q^T \tilde{x}_i}_{x_i}$$

Hat $\tilde{A}$ Hessenberg-Form , so auch $\tilde{A} - \bar{l} I$ für jedes $\bar{l} \in C$.
Die LR-Zerlegung von $\tilde{A} - \bar{l} I$ erfordert $O(n^2)$ Rechenoperationen.
Aufwand für die Hessenberg-Form : $O\left(\frac{4}{3} n^3\right)$.

Spezialfall : Ist A symmetrisch , so ist $\tilde{A}$ Tridiagonalmatrix.

6.3 QR-Algorithmus

geg.: $A \in R^{n \times n}$, setze $A_0 := A$

Algorithmus :

- Berechne die QR-Zerlegung von $A_k = Q_k R_k$
- $A_{k+1} = R_k Q_k$

Wegen $A_{k+1} = R_k Q_k = Q_k^T A_k Q_k$ folgt mittels vollständiger Induktion , dass alle A_k ähnlich zu A sind.

Nachteil : pro Iterationsschritt QR-Zerlegung und Matrix-Matrix-Multiplikation

Alternative : Transformiere A zunächst auf Hessenberg-Form $\tilde{A}$ und wende den QR-Algorithmus auf $\tilde{A}$ an.
Vorteil : QR-Zerlegung mit $n-1$ Givens-Rotationen $\Omega_{n-1,n}, ..., \Omega_{1,2}$

$$\Omega_{1,2} \tilde{A} = \Omega_{1,2} \begin{pmatrix} * & * & & \cdots & * \\ * & * & & & \vdots \\ 0 & * & & & \\ \vdots & & \ddots & & * \\ 0 & \cdots & 0 & * & * \end{pmatrix} = \begin{pmatrix} * & * & & \cdots & * \\ 0 & * & & & \vdots \\ 0 & * & & & \\ \vdots & & \ddots & & * \\ 0 & \cdots & 0 & * & * \end{pmatrix}$$

(Nur Veränderung der ersten zwei Zeilen.)

usw. , d.h. $\Omega_{n-1,n} \cdot ... \cdot \Omega_{1,2} \cdot \tilde{A} = R_0$, $\quad Q_0^T = \Omega_{n-1,n} \cdot ... \cdot \Omega_{1,2}$

$\tilde{A}_1 = R_0 Q_0 = R_0 {\Omega_{1,2}}^T ... \Omega_{n-1,n}$ (ebenfalls Hessenberg-Form)

Ergebnisse :

- Anwendung des QR-Algorithmus auf Hessenberg-Matrix liefert eine Folge von Hessenberg-Matrizen.
- Rechenaufwand : pro Iterationsschritt $O(n^2)$ für $2 \cdot (n-1)$ Givens-Rotationen

Ist A symmetrisch und tridiagonal , so reduziert sich der Rechenaufwand pro Iterationsschritt auf $O(n)$.

Satz 6.14 : Konvergenz des QR-Algorithmus (siehe 5.7 in Numerik1-Skript)

geg.: $A \in R^{n \times n}$ symmetrisch und mit den Eigenwerten $l_1,...,l_n$, wobei $|l_1| > |l_2| > ... > |l_n|$.

Für die Matrizen Q_k, R_k und A_k gilt :

a) $\lim_{k\to\infty} Q_k = I$

b) $\lim_{k\to\infty} Q_k = \Lambda = diag(l_1,..., l_n) = \lim_{k\to\infty} A_k$

c) $a_{ij}^{(k)} = O\left(\left|\frac{l_i}{l_j}\right|^k\right)$ für $i > j$

Beweis zu a) :

Es gilt $A^k = \underbrace{Q_1...Q_k}_{P_k}\underbrace{R_k...R_1}_{S_k}$, denn für $k=1$ ist dies trivial und für $k>1$ folgt aus

$A^k = P_k S_k$ wegen $A_{k+1} = Q_{k+1}R_{k+1} = Q_k^T ...\underbrace{Q_1^T A Q_1}_{=A_2}...Q_k = P_k^{-1} A P_k$ auch

$A^{k+1} = AA^k = \underbrace{AP_k}_{P_k A_{k+1}} S_k = P_k Q_{k+1} R_{k+1} S_k = P_{k+1} S_{k+1}$

P_k ist orthogonal , S_k ist obere Dreiecksmatrix.

$\Rightarrow$ $A^k = P_k S_k$ ist (eine) QR-Zerlegung von A^k .

Bemerkung 6.15 : Allgemeine Konvergenzaussagen

a) Der QR-Algorithmus konvergiert auch für mehrfache Eigenwerte.

b) Falls unterschiedliche Eigenwerte denselben Betragswert haben , so konvergieren die A_k nicht gegen Λ , sondern es bleiben 2×2 Blöcke entlang der Hauptdiagonalen stehen (Jordan-Blöcke?). Man kann die Eigenwerte trotzdem aus diesen Blöcken ausrechnen.

Bemerkung 6.16 : Verbesserung des QR-Algorithmus durch Shift-Technik

a) Modifikation :

$A_k - \tilde{l}_k I = Q_k R_k$

QR-Zerlegung für $\tilde{l}_k \approx l_i$

$A_{k+1} = R_k Q_k + \tilde{l}_k I$

Es gilt $A_{k+1} = Q_k^T \left(A_k - \tilde{l}_k I\right) Q_k + \tilde{l}_k I = Q_k^T A_k Q_k$

$\Rightarrow$ A_{k+1} und A_k sind einander ähnlich.

b) Konvergenz

Analog zu Satz 6.14 zeigt man unter Verwendung von $\left(A - \tilde{l}_k I\right) \cdot ... \cdot \left(A - \tilde{l}_1 I\right) = Q_1 ... Q_k R_k ... R_1$

$a_{ij}^{(k)} = O\left(\left|\frac{l_i - \tilde{l}_1}{l_j - \tilde{l}_1}\right| \cdot ... \cdot \left|\frac{l_i - \tilde{l}_k}{l_j - \tilde{l}_k}\right|\right) \quad \forall i > j$

Falls $\tilde{l}_l \approx l_i \quad (l \geq 1)$, so ergibt sich eine Konvergenzbeschleunigung.

c) Implementierung : Expliziter Shift (nach Wilkinson)

Ziel : $a_{n,n-1}^{(k+1)} \approx 0 \quad \Rightarrow \quad a_{n,n}^{(k+1)}$ ist eine sehr gute Näherung für Eigenwert von A

Wähle $\tilde{l}_k$ als denjenigen Eigenwert von $\begin{pmatrix} a_{n-1,n-1}^{(k)} & a_{n-1,n}^{(k)} \\ a_{n,n-1}^{(k)} & a_{n,n}^{(k)} \end{pmatrix} \in R^{2\times 2}$, der näher an $a_{n,n}^{(k)}$ liegt.

$\Rightarrow$ Konvergenz von $a_{n,n-1}^{(k)} \to 0$ und $a_{n,n}^{(k)} \to l_i(A)$ quadratisch und häufig sogar kubisch.

Berechnung des Eigenwertes $l_i(A)$ in der Regel mit $O(n)$ Rechenoperationen.

Erweiterung auf nichtsymmetrische Matrizen mit konjugiert komplexen Eigenwerten :
Implizite Shift-Technik (nach Francis) – ermöglicht rein reelle Berechnungen

d) nächste Eigenwerte und Abbruchkriterium

Deflation : Ist $\max_{1\le j<n} \left| a_{n,j}^{(k)} \right| = O(eps)$, so ist $a_{n,n}^{(k)}$ eine gute Näherung für einen Eigenwert von A.

Der QR-Algorithmus wird mit $A = A_k(1:(n-1),1:(n-1)) \in R^{(n-1)\times(n-1)}$ fortgesetzt.
(D.h. letzte Zeile und letzte Spalte weglassen.)
Falls $n = 2$, so werden die letzten beiden Eigenwerte explizit berechnet.

e) Gesamtaufwand für symmetrische Matrizen

- $O\left(\frac{4}{3}n^3\right)$ Rechenoperationen für Transformation auf Tridiagonal-Form
- $n \cdot O(n) = O(n^2)$ Rechenoperationen für QR-Algorithmus

f) Berechnung der Eigenvektoren

$A = U\Lambda U^T \quad \Rightarrow \quad AU = U\Lambda$

$\Rightarrow$ Spaltenvektoren von U sind die Eigenvektoren von A

$PAP^T = \tilde{A}$ (Transformation auf Tridiagonalgestalt)

$\tilde{A} = A_0 = Q_0 Q_1 ... Q_k A_{k+1} Q_k^T ... Q_1^T Q_0^T$

$\Rightarrow \quad U = P^T Q_0 Q_1 ... Q_k$

<u>Beispiel 6.17 : QR-Algorithmus mit Shift-Technik</u>

$$A = \begin{pmatrix} 12 & 1 & & \cdots & 0 \\ 1 & 9 & 1 & & \vdots \\ & 1 & 6 & 1 & \\ \vdots & & 1 & 3 & 1 \\ 0 & \cdots & & 1 & 0 \end{pmatrix}$$

k	$a_{5,4}^{(k)}$	$a_{5,5}^{(k)}$	$\tilde{l}_k$
1	-4.5 E-3	0	-0.302776
2	1.1 E-10	-0.316869	-0.316876
3		-0.316876	-0.316876
4	9.1 E-23	-0.316876	-0.316876

$\Rightarrow l_5 \approx -0.316876$

Fortsetzung mit der Matrix $A_4(1:4,1:4)$:

$\Rightarrow a_{4,3}^{(6)} = -1.1\,E-18 \qquad \Rightarrow l_4 \approx 2.98386$

Fortsetzung mit der Matrix $A_6(1:3,1:3)$:

$\Rightarrow a_{3,2}^{(8)} = 1.8\,E-20 \qquad \Rightarrow l_3 \approx 6.0$

Explizite Berechnung der beiden letzten Eigenwerte aus $A_8(1:2,1:2)$:

$l_2 \approx 9.01614$, $\qquad l_1 \approx 12.316876$

Zum Vergleich der Konvergenzgeschwindigkeiten : QR-Algorithmus ohne Shift-Technik

$a_{2,1}^{(12)} = 0.0337458$

$a_{3,2}^{(12)} = 0.011408$

$a_{3,3}^{(12)} = -0.316876$

Konvergenzgeschwindigkeit entspricht wegen $\left|\frac{l_5}{l_4}\right| \approx 0.1$ den Abschätzungen von Satz 6.14.

6.4 Lancosz-Methoden

Bemerkung 6.18 : Eigenwertprobleme als Extremwertaufgaben

Zu einer symmetrischen Matrix $A \in R^{n\times n}$ und $x \in R^n \setminus \{0\}$ bezeichnet

$m(x) := \frac{\langle x, Ax\rangle}{\langle x, x\rangle}$ den **Rayleigh-Quotienten**.

Es gilt : $\min_{x\neq 0} m(x) = l_{\min}(A)$ und $\max_{x\neq 0} m(x) = l_{\max}(A)$ mit dem kleinsten bzw. größtem Eigenwert von A.

Denn A ist diagonalisierbar : $A = U\Lambda U^T$ mit $\Lambda = diag(l_1,...,l_n)$, so dass wegen $U^T AU = \Lambda$ für $y := U^T x$ folgt :

$\langle y, y\rangle = \langle U^T x, U^T x\rangle = x^T UU^T x = \langle x, x\rangle$ und

$$\langle x, Ax\rangle = \langle Uy, AUy\rangle = \langle y, U^T AUy\rangle = \langle y, \Lambda y\rangle = \sum_{i=1}^{n} l_i y_i^2$$

Idee :

Approximiere $l_{\min}(A) = \min\{m(x) : x \in R^n, x \neq 0\}$ durch $\min\{m(x) : x \in V_k, x \neq 0\}$ mit einem niedrigdimensionalen Unterraum $V_k \subset R^n$. Analog $l_{\max}(A)$.

Ansatz :

Krylovraum $V_k = V_k(x) = K_k(x, A) = span\{x, Ax,..., A^{k-1}x\}$

Bemerkung 6.19 : Lanczos-Algorithmus

$A = A^T \in R^{n\times n}$

Bestimmen einer Orthonormalbasis $v_1,...,v_l$ von $V_l(x) = K_l(x,A)$ mittels Drei-Term-Rekursion.
(Effizienter als z.B. Gram-Schmidtsches -Verfahren)

$v_0 := 0$, $v_1 := \frac{x}{\|x\|_2}$

for $k = 1:(l-1)$

$a_k := \langle v_k, Av_k \rangle$

$w_{k+1} := Av_k - a_k v_k - b_k v_{k-1}$ (b_1 nicht nötig , da $v_0 = 0$)

$b_{k+1} := \|w_{k+1}\|_2$

$v_{k+1} := \frac{w_{k+1}}{b_{k+1}}$, falls $b_{k+1} \neq 0$

Abbruch , falls $w_{k+1} = 0$

endfor

In Matrixschreibweise : $AQ_k = Q_k T_k$ mit

$Q_k = [v_1, v_2, ..., v_k] \in R^{n\times k}$ und $T_k = \begin{pmatrix} a_1 & b_2 & & & & \\ b_2 & a_2 & b_3 & & & \\ & b_3 & a_3 & b_4 & & \\ & & \ddots & \ddots & \ddots & \\ & & & b_{k-1} & a_{k-1} & b_k \\ & & & & b_k & a_k \end{pmatrix}$

Es gilt : $T_k = Q_k^T A Q_k$, jedoch sind A und T_k einander **nicht ähnlich**. $(k < n)$

zu zeigen :

a) $A^k x \in span\{v_1,...,v_{k+1}\}$

b) $\langle w_{k+1}, v_j \rangle = 0 \qquad (j = 1,...,k)$

Beweis :

zu a) Folgt aus $w_{k+1} = g \cdot A^k x + r_k$ mit einem $g \neq 0$ und einem $r_k \in K_k(x,A)$
Nachweis mittels vollständiger Induktion

zu b)

$$\langle w_{k+1}, v_j \rangle = \underbrace{\langle Av_k, v_j \rangle}_{= \langle v_k, Av_j \rangle - \text{ wegen } A = A^T} - a_k \underbrace{\langle v_k, v_j \rangle}_{=0 \text{ für } j<k} - b_k \underbrace{\langle v_{k-1}, v_j \rangle}_{=0 \text{ für } j \neq k-1}$$

$Av_j \in K_{j+1}(x,A) = span\{v_1, v_2, ..., v_{j+1}\}$ $\Rightarrow$ Skalarprodukt $= 0$ für $j < k-1$

d.h. , $\langle w_{k+1}, v_j \rangle = 0$ für $j = 1,2,...,k-2$

Fall $j = k-1$:

$$\langle w_{k+1}, v_{k-1}\rangle = \left\langle v_k, \underbrace{w_k + a_{k-1}v_{k-1} + b_{k-1}v_{k-2}}_{\text{siehe obige Def. von } w_k}\right\rangle - b_k \underbrace{\langle v_{k-1}, v_{k-1}\rangle}_{=1}$$
$$= \langle v_k, Av_{k-1}\rangle - b_k$$
$$= \langle v_k, w_k\rangle - b_k$$
$$= 0 \quad \text{, da } w_k = b_k v_k \text{ und } \langle v_k, v_k\rangle = 1$$

Fall $j = k$:

$$\langle w_{k+1}, v_k\rangle = \langle v_k, Av_k\rangle - a_k\langle v_k, v_k\rangle - b_k \cdot 0 = \langle v_k, A_{v_k}\rangle - a_k = 0$$

Anwendung :

Zu $u \in R^k$ betrachte $y = Q_k u \in K_k(x, A) \subset R^n$, dann ist

$\langle y, y\rangle = \langle Q_k u, Q_k u\rangle = \langle u, Q_k^T Q_k u\rangle = \langle u, u\rangle$ und

$\langle y, Ay\rangle = \langle Q_k u, AQ_k u\rangle = \langle u, Q_k^T AQ_k u\rangle = \langle u, T_k u\rangle$

Also ist

$$l_{\min}^{(k)} = \min_{\substack{y \in V_k(x) \\ y \neq 0}} \frac{\langle y, Ay\rangle}{\langle y, y\rangle} = \min_{\substack{u \in R^k \\ u \neq 0}} \frac{\langle u, T_k u\rangle}{\langle u, u\rangle} = l_{\min}(T_k)$$

Analog ist $l_{\max}^{(k)} = l_{\max}(T_k)$.

Wegen $V_k \subset V_{k+1}$ folgt weiterhin $l_{\min}^{(k+1)} \leq l_{\min}^{(k)}$ und $l_{\max}^{(k+1)} \geq l_{\max}^{(k)}$.

Satz 6.20 : Eigenwertberechnung mittels Lanczos-Verfahren

Sei A eine symmetrische Matrix mit den Eigenwerten $l_1 \leq ... \leq l_n$ und zugehörigen orthonormalen Eigenvektoren $h_1, ..., h_n$.

Aus dem Lanczos-Verfahren zum Startwert $x \neq 0$ ergibt sich eine ONB $v_1, ..., v_k$ von $V_k(x) = K_k(x, A)$, sowie die Tridiagonalmatrix $T_k \in R^{k \times k}$ mit den Eigenwerten $m_1 \leq ... \leq m_k$.

Dann gilt : $$l_n \geq m_k \geq l_n - \frac{(l_n - l_1)\tan^2(\angle(v_1, h_n))}{T_{k-1}^2\left(1 + 2\frac{l_n - l_{n-1}}{l_{n-1} - l_1}\right)}$$ mit dem Tschebyscheff-Polynom T_{k-1}.

6.5 Singulärwertzerlegung

Satz 6.21

Sei $A \in R^{m \times n}$. Dann gibt es zwei orthogonale Matrizen $U \in R^{m \times m}$ und $V \in R^{n \times n}$, so dass $U^T AV = \Sigma = diag(s_1, ..., s_p)$ mit $p = \min(m, n)$ und $s_1 \geq ... \geq s_p \geq 0$.

Diese Zerlegung $U^T AV = \Sigma$ heißt Singulärwertzerlegung von A und die s_i sind die Singulärwerte von A.

Beweis mittels vollständiger Induktion

zu zeigen :

Es gibt orthogonale Matrizen U_1, V_1 mit $U_1^T AV_1 = \begin{pmatrix} s_1 & 0 \\ 0 & B \end{pmatrix}$, wobei $B \in R^{(m-1)\times(n-1)}$ ist.

Sei $s_1 := \|A\|_2 = \max_{\|v\|_2=1} \|Av\|_2$, dann gibt es Vektoren $v_1 \in R^n$, $u_1 \in R^m$ mit

$\|v_1\|_2 = 1$, $u_1 := \frac{1}{s_1 A v_1}$, $\|u_1\|_2 = 1$

und $\{v_1\}, \{u_1\}$ können zu orthogonalen Matrizen $V_1 = [v_1, ..., v_n], U_1 = [u_1, ..., u_m]$ erweitert werden.

$$A_1 := U_1^T AV_1 = U_1^T [Av_1, Av_2, ..., Av_n] = [s \cdot u_1, Av_2, ..., Av_n] = \begin{pmatrix} s_1 & w^T \\ 0 & B \end{pmatrix}$$

(wegen $U_1^T (s_1 u_1) = s_1$ und $U_1^T u_j = 0$ für $j = 2, ..., m$)

mit einem $w \in R^{n-1}$ und einer Matrix $B \in R^{(m-1)\times(n-1)}$.

Wegen $\|C\|_2 = \sqrt{l_{\max}(C^T C)}$ gilt

$$\|A_1\|_2^2 = l_{\max}\left(A_1^T A_1\right) = l_{\max}\left(V_1^T A^T U_1 U_1^T AV_1\right) = l_{\max}\left(V_1^T (A^T A) V_1\right) = l_{\max}\left(A^T A\right)$$

(wegen Ähnlichkeit der Matrizen)

$$= \|A\|_2^2 = s_1^2$$

$\Rightarrow$

$$s_1^2 = \|A_1\|_2^2 \geq \frac{\left\| A_1 \cdot \begin{pmatrix} s_1 \\ w \end{pmatrix} \right\|_2^2}{\left\| \begin{pmatrix} s_1 \\ w \end{pmatrix} \right\|_2^2}$$ (Verträglichkeit von Matrix- und Vektornorm)

$$= \frac{\left\| \begin{pmatrix} s_1^2 + w^T w \\ \vdots \end{pmatrix} \right\|_2^2}{s_1^2 + w^T w} \geq \frac{\left(s_1^2 + w^T w\right)^2}{s_1^2 + w^T w} = s_1^2 + w^T w$$

$\Rightarrow \quad w = 0$, $U_1^T AV_1 = \begin{pmatrix} s_1 & 0 \\ 0 & B \end{pmatrix}$

Bermerkung 6.22 : Eigenschaften der Singulärwertzerlegung

Für die in Matrix 6.21 konstruierte Matrixzerlegung $U^T AV = \Sigma$ gilt :

a) $Av_i = s_i u_i \quad , \quad A^T u_i = s_i v_i \qquad (i = 1,...,p)$ mit
den Spaltenvektoren $u_1,...,u_m$ von U und $v_1,...,v_n$ von V.

b) Es ist $s_1 \geq s_2 \geq ... \geq s_r > s_{r+1} = ... = s_p = 0 \quad \Leftrightarrow \quad rang(A) = r$
$\ker(A) = span\{v_{r+1},...,v_n\}$
$im(A) = span\{u_1,...,u_r\}$

c) $s_1 = \|A\|_2$

d) $\|A\|_F^2 = \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^{\ 2} = \|\Sigma\|_F^2 = \sum_{i=1}^{p} s_i^{\ 2}$

e) $k_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2 = \frac{s_1}{s_n}$ für reguläre $A \in R^{n\times n}$

f) $A^T A$ hat die Eigenwerte $s_1^{\ 2},...,s_p^{\ 2}$ zu den Eigenvektoren $v_1,...,v_p$
$A^T A$ hat die Eigenwerte $s_1^{\ 2},...,s_p^{\ 2}$ zu den Eigenvektoren $u_1,...,u_p$

<u>weitere Eigenschaften :</u>

- Gegeben sei eine Matrix $A \in R^{m\times n}$ und orthogonale Matrizen $P \in R^{m\times m}$, $Q \in R^{n\times n}$.
  Dann haben A und $B := PAQ$ die gleichen Singulärwerte.
- Sei $U^T AV = \Sigma = diag(s_1,...,s_r,0,...,0)$ die Singulärwertzerlegung einer Matrix $A \in R^{m\times n}$
  mit $rang(A) = r$.
  Dann ist die Pseudoinverse $A^+ \in R^{n\times m}$ gegeben durch $A^+ = V\Sigma^+ U^T$
  mit $\Sigma^+ = diag\left(\frac{1}{s_1},...,\frac{1}{s_r},0,...,0\right)$

(Penrose-Axiome $AA^+A = A$, $x = A^+ b$)

<u>Bemerkung : Bi-Diagonalisierung rechteckiger Matrizen</u>

geg. : $A \in R^{m\times n}$, $m \geq n$

ges. : orthogonale Matrizen $P \in R^{m\times m}$, $Q \in R^{n\times n}$ mit

$$PAQ = \begin{pmatrix} B \in R^{n\times n} \\ ... \\ 0_{(m-n)\times n} \end{pmatrix} = \begin{pmatrix} * & * & & \\ & \ddots & \ddots & \\ & & \ddots & * \\ & & & * \\ .. & .. & .. & .. \\ 0 & \cdots & & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & & 0 \end{pmatrix}$$

, B = Bi-Diagonalmatrix

<u>Vorgehensweise :</u>

Wähle P_1 als Householder-Spiegelung , welche die erste Spalte von A in ein Vielfaches des ersten Einheitsvektors transformiert.

$$P_1 A = \begin{pmatrix} * & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix}$$

Wähle Q_1 als Householder-Spiegelung so , dass $P_1 A Q_1 = \begin{pmatrix} * & * & 0 & \dots & 0 \\ 0 & * & * & \dots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & \dots & * \end{pmatrix}$

(Die erste Komponente des Spiegelungsvektors in Q_1 wird $= 0$ gesetzt.)

$\Rightarrow$ 1. Spalte von $P_1 A$ bleibt unverändert

Mittels vollständiger Induktion erhält man die Bi-Diagonalgestalt.

$$PAQ = \begin{pmatrix} B \\ 0 \end{pmatrix} = \underbrace{P_{n-1} \dots P_1}_{=:P} A \underbrace{Q_1 \dots Q_{n-2}}_{=:Q}$$

Singulärwertzerlegung (bidiagonaler Matrizen) - Algorithmus :

siehe „chasing“ / “bulge chasing“.

MATLAB-Befehl : svd (single value decomposition)

Idee : Anwendung des QR-Algorithmus auf $B^T B$ ohne explizite Berechnung von $B^T B$
(Anwendung von Givens-Drehungen)

TODO : genaue Beschreibung

Ergebnis :

Bi-Diagonalmatrix $B_1 = \tilde{\Omega}_{n-1,n} \dots \tilde{\Omega}_{1,2} B \Omega_{1,2}^T \dots \Omega_{n-1,n}^T$

Die Transformation $B^T B \to B_1^T B_1$ entspricht dem ersten Schritt des QR-Algorithmus angewendet auf $B^T B$, jedoch ohne explizite Berechnung von $B^T B$.

Nach Satz 6.14 konvergiert $B_k^T B_k$ gegen $\Lambda = diag\left(s_1^2, \dots, s_n^2\right) = \Sigma^T \Sigma$

$$B_k \underset{k \to \infty}{\to} \Sigma$$

Aufwand : für quadratische Matrizen $(m = n)$

- $\frac{4}{3} n^3 + O(n^2)$ Rechenoperationen für Reduktion auf Bi-Diagonalgestalt
- $O(n^2)$ Rechenoperationen für den modifizierten QR-Algorithmus

7. Interpolation

Problemstellung :

Gegeben seien diskrete Funktionswerte und ggf. Ableitungen einer Funktion $f : R \to R$ an den Stützstellen t_i $(i = 0,1,\dots,n)$

Und zwar $f(t_i),\ f'(t_i),...,\ f^{(c_i)}(t_i)$ mit einem $c_i \geq 0$ (=Stützwerte)

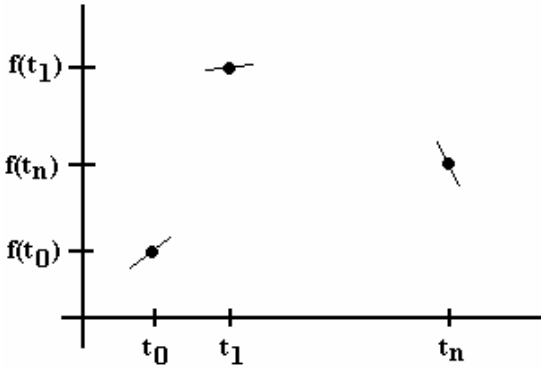


Abb : Beispiel für die zwei Fälle $c_i = 0$ und $c_i = 1$ (im ersten Fall z.B. quadratisches Polynom)

Gesucht : Eine Approximation von f durch eine mit wenig Aufwand und numerisch stabil auswertbare Funktion j , wie z.B.

- Polynome
- stückweise Polynome (bei sehr vielen Stützstellen Aufteilung)
- trigonometrische Funktionen
- Exponentialfunktionen
- rationale Funktionen

Interpolationsbedingungen : $j^{(j)}(t_i) = f^{(j)}(t_i)$, $\begin{pmatrix} j = 0,1,...,c_i \\ i = 0,1,...,n \end{pmatrix}$

7.1 Klassische Polynominterpolation

Satz 7.2 : Interpolationspolynome

Zu beliebigen $n+1$ Stützpunkten (t_i, f_i) , $(i = 0,1,...,n)$ mit $t_i \neq t_j \quad \forall i \neq j$ gibt es genau ein Polynom höchstens n-ten Grades

$P(t) \in \Pi_n = \{p(t) : p(t) = a_0 + a_1 t + ... + a_n t^n \quad , \quad a_i \in R\}$ mit

$P(t_i) = f_i \quad , (i = 0,1,...,n)$

(Zusammenhang siehe Vandermonde Determinante)

Beweis :

a) Eindeutigkeit

Gilt für $P_1, P_2 \in \Pi_n \qquad P_1(t_i) = f_i = P_2(t_i)$ für $i = 0,1,...,n$, so hat das Polynom $P_1 - P_2 \in \Pi_n$ wegen $P_1(t_i) - P_2(t_i) = 0$ mindestens $n+1$ verschiedene Nullstellen.

$\Rightarrow \quad P_1 - P_2 =$ Nullpolynom

$\Rightarrow \quad P_1 = P_2$

b) Existenz

Konstruiere Lagrange-Polynome $L_i \in \Pi_n \quad (i = 0,1,...,n)$

mit $L_i(t_k) = \begin{cases} 1 & für\ i = k \\ 0 & sonst \end{cases} = d_{ik}$

$$\Rightarrow P(t) = \sum_{i=0}^{n} f_i L_i(t)$$

Lagrangesche Darstellung des Interpolationspolynoms :

$$L_i(t) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t-t_j}{t_i-t_j} = \frac{(t-t_0)(t-t_1)...(t-t_{i-1})(t-t_{i+1})...(t-t_n)}{(t_i-t_0)(t_i-t_1)...(t_i-t_{i-1})(t_i-t_{i+1})...(t_i-t_n)}$$

$$= \frac{w(t)}{(t-t_i)\cdot w'(t_i)} \qquad \text{mit dem Knotenpolynom}$$

$$w(t) := \prod_{j=0}^{n}(t-t_j) = (t-t_0)(t-t_1)...(t-t_n) \qquad \text{q.e.d.}$$

Hinweis :

$\{L_0, L_1, ..., L_n\}$ ist Orthonormalbasis bezüglich $\langle P, Q\rangle := \sum_{i=0}^{n} P(t_i)\cdot Q(t_i)$

Bemerkung 7.3 : Interpolationspolynom

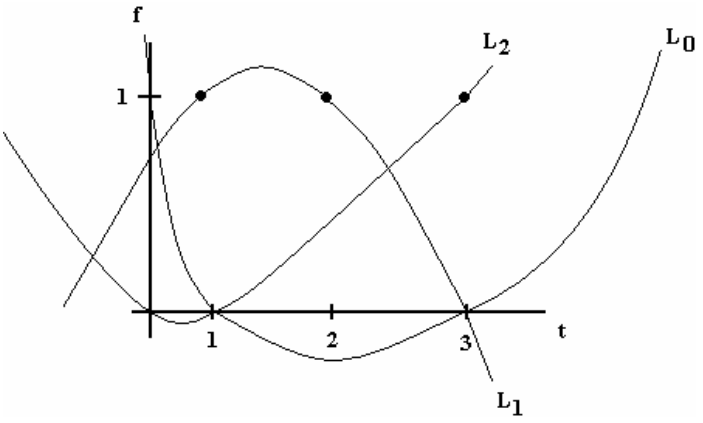
a) Das (eindeutig bestimmte) Polynom P aus Satz 7.2 heißt Interpolationspolynom zu den Stützpunkten (t_i, f_i).

b) Als Abbildung $P : \underbrace{C[a,b]}_{\text{Menge der stetigen Fkt}} \to \Pi_n$ mit $a \le t_i \le b$ besitzt die klassische Polynominterpolation die Kondition $k_{abs} = \max_{t\in[a,b]} \sum_{i=0}^{n} |L_i(t)|$.

Diese sogenannte Lebesgue-Konstante hängt von der Lage der Stützstellen $t_0, ..., t_n$ in $[a,b]$ ab und wird besonders klein für die Tschebyscheff-Knoten.

Beispiel 7.4 : Lagrange-Polynome

geg,: $(0,1), (1,3), (3,2)$



$$L_0(t) = \frac{(t-1)(t-3)}{(-1)(-3)} \quad , \qquad L_1(t) = \frac{t\cdot(t-3)}{1\cdot(-2)} \quad , \qquad L_2(t) = \frac{t\cdot(t-1)}{3\cdot 2}$$

$$P(t) = L_0(t) + 3\cdot L_1(t) + 2\cdot L_2(t)$$

Bemerkung 7.5 : Algorithmus von Aitken-Neville

geg.: (t_i, f_i) $,(i = 0,1,...,n)$

ges.: $P(t)$ effizient

Bezeichnungen :

- Zu $k+1 \le n+1$ seien paarweise Indizes i_j definiert mit $j = 0,1,...,k$ und $i_j \in \{0,1,...,n\}$.
- $P_{i_0}, P_{i_1},..., P_{i_k} \in \Pi_k$ seien die Interpolationspolynome mit $P_{i_0},..., P_{i_k}(t_i) = f_i$ für $i = i_0,..., i_k$

Eigenschaften :

a) $P_i(t) = f_i$

b) $P_{i_0},..., P_{i_k}(t) = \dfrac{(t - t_{i_0})P_{i_1},..., P_{i_k}(t) - (t - t_{i_k})P_{i_0},..., P_{i_{k-1}}(t)}{t_{i_k} - t_{i_0}}$,

denn beide Ausdrücke nehmen für $t = t_{i_j}$ den Wert f_{i_j} an und sind Polynome aus Π_k .

Rechenschema : für n=3

	$k=0$	$k=1$	$k=2$	$k=3$
t_0	$f_0 \equiv P_0(t)$	$P_{01}(t) = ...$		
t_1	$f_1 \equiv P_1(t)$		$P_{012}(t) = ...$	$P_{0123}(t) = P(t)$
t_2	$f_2 \equiv P_2(t)$	$P_{12}(t) = ...$	$P_{123}(t) = ...$	
t_3	$f_3 \equiv P_3(t)$	$P_{23}(t) = ...$		

Beispiel : Stützpunkte $(0,1)$, $(1,3)$, $(3,2)$

	$k=0$	$k=1$	$k=2$
0	1	$P_{01}(2) = \dfrac{(2-0)\cdot 3 - (2-1)\cdot 1}{1-0} = 5$	
1	3		$P_{012}(2) = \dfrac{(2-0)\cdot \frac{5}{2} - (2-3)\cdot 5}{3-0} = \dfrac{10}{3}$
2	2	$P_{12}(2) = \dfrac{(2-1)\cdot 2 - (2-3)\cdot 3}{3-1} = \dfrac{5}{2}$	

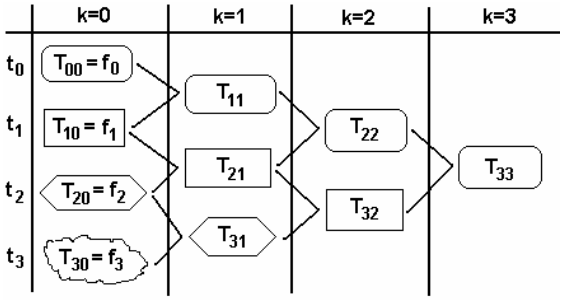
kurz :

$$P_{i,i+1,\ldots,i+k} =: T_{i+k,k} \qquad \Rightarrow T_{i,0} = f_i$$

$$P_{i-1,\ldots,i} =: T_{i,k} \qquad T_{i,k} = \frac{(t - t_{i-k})T_{i,k-1} - (t - t_i)T_{i-1,k-1}}{t_i - t_{i-k}}$$

$$T_{i,k} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\frac{t - t_{i-k}}{t - t_i} - 1} \qquad ,1 \le k \le i \qquad ,i \ge 0$$

Speicherschema für k=3 :



(Gleiche Formen belegen gleichen Speicherplatz.)

Implementierung :

Bezeichnung : $T_j := T_{i,k}$ mit $j = i - k$

for $i = 0 : n$
  $T_i := f_i$
  for $j = i - 1 : -1 : 0$

$$T_j := T_{j+1} + \frac{T_{j+1} - T_j}{\left(\frac{t - t_j}{t - t_i} - 1\right)}$$

  endfor
endfor

Bemerkung 7.6 : Dividierte Differenzen

Idee : $P_{i_0,i_1,\ldots,i_{k-1}}$ und $P_{i_0,i_1,\ldots,i_k}$ haben jeweils denselben Funktionswert in $t_i, t_{i_1},\ldots,t_{i_{k-1}}$.

$\Rightarrow P_{i_0,i_1,\ldots,i_k}(t) = P_{i_0,i_1,\ldots,i_{k-1}}(t) + (t - t_{i_0})(t - t_{i_1})\ldots(t - t_{i_{k-1}})\, f_{i_0,i_1,\ldots,i_k}$ mit einem $f_{i_0,i_1,\ldots,i_k} \in R$.

$$f_{i_0,i_1,\ldots,i_k} = \frac{f_{i_1,i_1,\ldots,i_k} - f_{i_0,i_1,\ldots,i_{k-1}}}{t_{i_k} - t_{i_0}}$$

Eigenschaften :

$f_{i_0,i_1,\ldots,i_k}$ ist invariant gegenüber Permutationen der Indizes $i_0, i_1,\ldots,i_k$, da $P_{i_0,i_1,\ldots,i_k}(t)$ durch die Interpolationsbedingungen $P(t_{i_j}) = f_{i_j}$ $,(j = 0,1,\ldots,k)$

eindeutig und von der Reihenfolge der t_{i_j} unabhängig festgelegt ist.

Steigungsschema :

$$f_{i,i+1,\ldots,i+k} = \frac{f_{i+1,\ldots,i+k} - f_{i,\ldots,i+k-1}}{t_{i+k} - t_i}$$

	$k=0$	$k=1$	$k=2$
t_0	f_0	$f_{01} = \frac{f_1 - f_0}{t_1 - t_0}$	
t_1	f_1		$f_{012} = \frac{f_{12} - f_{01}}{t_2 - t_0}$
t_2	f_2	$f_{12} = \frac{f_2 - f_1}{t_2 - t_1}$	

Beispiel : Stützpunkte $(0,1)$, $(1,3)$, $(3,2)$

0	1	$\frac{3-1}{1-0} = 2$	
1	3		$\frac{-\frac{1}{2} - 2}{3-0} = -\frac{5}{6}$
3	2	$\frac{2-3}{3-1} = -\frac{1}{2}$	

Implementierung :

for $i = n:-1:0$
  $a_i := f_i$
  for $j = i+1:n$
    $a_j = \frac{a_j - a_{j-1}}{t_j - t_i}$
  endfor
endfor

Bemerkung 7.7 : Newtonsche Darstellung des Interpolationspolynoms
Aus Bemerkung 7.6 erbibt sich rekursiv die Newtonsche Darstellung des Interpolationspolynoms :

$$P(t) = P_{0,1,\ldots,n}(t) = P_{0,1,\ldots,n-1}(t) + (t-t_0)(t-t_1)\ldots(t-t_{n-1}) \cdot f_{0,1,\ldots,n}$$
$$= \ldots$$

$$= f_0 + (t-t_0) f_{0,1} + (t-t_0)(t-t_1) f_{0,1,2} + \ldots + (t-t_0)(t-t_1)\ldots(t-t_{n-1}) f_{0,1,\ldots,n}$$
$$= f_0 + (t-t_0)[\, f_{0,1} + (t-t_1) f_{0,1,2} + \ldots + (t-t_{n-2})(\, f_{0,1,\ldots,n-1} + (t-t_{n-1}) f_{0,1,\ldots,n})]$$

Beispiel : Stützpunkte $(0,1)$, $(1,3)$, $(3,2)$

$$P(t) = 1 + (t-0)\left(2 + (t-1)\cdot\left(-\frac{5}{6}\right)\right)$$
$$P(2) = 1 + (2-0)\left(2 + (2-1)\left(-\frac{5}{6}\right)\right) = 1 + 2\left(2 - \frac{5}{6}\right) = 1 + \frac{7}{3} = \frac{10}{3}$$

Implementierung : Newton-Horner-Schema

$P := a_n$
for $i : n-1 : -1 : 0$
  $P := P \cdot (t - t_i) + a_i$
endfor

Satz 7.8 : Restglied der Polynominterpolation

Gegeben seien $f : C^{n+1}[a,b] \to R$ und das Interpolationspolynom $P(t)$ zu den Stützstellen $(t_i\ ,\ f(t_i))$, $i = 0,1,\ldots,n$ mit $n+1$ paarweise voneinander verschiedenen Stützstellen $t_i \in [a,b]$, $i = 0,1,\ldots,n$

Zu jedem $\bar{t} \in [a,b]$ gibt es ein $t \in [a,b]$ mit $f(\bar{t}) - P(\bar{t}) = \dfrac{w(\bar{t})\, f^{(n+1)}(t)}{(n+1)!}$ = Restglied ,
mit $w(t) := (t-t_0)(t-t_1)\ldots(t-t_n)$ und $t = t(\bar{t})$.

Folgerung 7.9 : Eigenschaften der dividierten Differenzen

a) Unter den Voraussetzungen von Satz 7.8 gilt für jedes $\bar{t} \in [a,b]$: $f(\bar{t}) - P(\bar{t}) = w(\bar{t}) \cdot f[t_0, t_1, \ldots, t_n, \bar{t}]$, wobei $f[t_0, t_1, \ldots, t_n, \bar{t}]$ die dividierte Differenz $f_{0,1,\ldots,n+1}$ zu den $n+2$ Stützpunkten $(t_i\ ,\ f(t_i)), (\bar{t},\ f(\bar{t}))$ bezeichnet.

b) Es gilt $f[t_0, t_1, \ldots, t_n] = \dfrac{f^{(n)}(t)}{n!}$ für ein $t \in [a,b]$.

c) Für $f \in \Pi_n$ ist $f[t_0, t_1, \ldots, t_n, t_{n+1}, \ldots, t_{n+k}] \equiv 0$, $k > 0$

Beispiel 7.10 : Restglied der Polynominterpolation

$$f(t) = \sin t \quad , \quad t_i = i \cdot \frac{p}{10} \quad , \quad i = 0,1,\ldots,5 \quad , \quad n = 5$$
$$\sin t - P(t) = \frac{1}{720}(t-t_0)(t-t_1)\ldots(t-t_5)(-\sin t)$$

$$\bar{t} \in \left[0, \frac{p}{2}\right]: \qquad \left|\sin \bar{t} - P(\bar{t})\right| \le \frac{1}{720}\left(\frac{p}{2}\right)^2\left(\frac{2p}{5}\right)^2\left(\frac{3p}{10}\right)^2 = \frac{p^{\,6}}{2 \cdot 10^5} \approx 0.005$$

beachte : Ausserhalb des Intervalls $\left(\min_i t_i \ , \max_i t_i\right)$ („Extrapolation“)
häufig sehr große Fehler $\left(\left|w(t)\right| >> 1\right)$.

Betrachte eine Folge $\left(\Delta_m\right)$ von Intervallteilungen mit $\Delta_m = \left\{a = t_0^{\,(m)} < t_1^{\,(m)} < \ldots < t_{n_m}^{\,(m)} = b\right\}$ mit zugehörigen Interpolationspolynomen $P(t\,;\Delta_m)$.

Satz von Faber :

Zu jeder solchen Folge existiert ein $f \in C[a,b]$ so , dass $P(t\,;\Delta_m)$ auf $[a,b]$ nicht gleichmäßig konvergiert.
(D.h. es gibt keine optimale Folge von Stützstellen.)

Beispiel : Funktion von Runge

$$f(t) = \frac{1}{1+t^2} \quad , \qquad [a,b] = [-5,5]$$

$$t_i^{\,(m)} = -5 + i \cdot \frac{10}{m} \qquad , \ i = 0,1,\ldots,m$$

Bemerkung 7.11 : Hermite-Interpolation

Interpolationsbedingungen $P^{(j)}(t_i) = f^{(j)}(t_i) \quad , \begin{pmatrix} i = 0,1,\ldots,n \\ j = 0,1,\ldots,c_i \end{pmatrix}$

klassische Hermite-Interpolation :

$$P \in \Pi_r \ \text{ mit } \ r = \sum_{i=0}^{n} c_i + n$$

Lagrange-Darstellung :

Bestimme $L_{ij}(t) \in \Pi_r$ mit

$$L_{ij}^{\,(l)}(t_k) = \begin{cases} 1 & falls\ i = k\ und\ \ j = l \\ 0 & \qquad\quad sonst \end{cases}$$

$\Rightarrow$

$$P(t) = \sum_{i=0}^{n}\sum_{j=0}^{c_i} f^{(j)}(t_i) L_{ij}(t)$$

Newtonsche Darstellung :

Knoten mehrfach angeben , Steigung $f[t_i, t_i]$ ersetzen durch vorgegebene Daten $f'(t_i)$,

$f[t_i, t_i, t_i]$ ersetzen durch vorgegebene Daten $\frac{1}{2} f''(t_i)$, ... usw.

Beispiel 1 :

Interpolationsbedingungen $P(t_i) = f_i \quad , \quad P'(t_i) = f_i' \quad , \quad P''(t_i) = f_i'' \qquad , i = 0,1$

Steigungsschema :

t_0	f_0	$f_{00} = f_0'$	$f_{000} = \frac{1}{2} f_0''$	
t_0	f_0	$f_{00} = f_0'$		$f_{0001} = \frac{f_{001} - f_{000}}{t_1 - t_0}$
t_0	f_0	$f_{01} = \frac{f_1 - f_0}{t_1 - t_0}$	$f_{001} = \frac{f_{01} - f_{00}}{t_1 - t_0}$	... usw.
t_1	f_1		$f_{011} = \frac{f_{11} - f_{01}}{t_1 - t_0}$	
t_1	f_1	$f_{11} = f_1'$		
t_1	f_1	$f_{11} = f_1'$	$f_{111} = \frac{1}{2} f_1''$	

$$P(t) = f_0 + (t - t_0) \cdot f_0' + (t - t_0)^2 \cdot \frac{1}{2} f_0'' + (t - t_0)^3 \cdot f_{0001} + (t - t_0)^3 (t - 1) \cdot f_{00011} + (t - t_0)^3 (t - t_1)^2 \cdot f_{000111}$$

Beispiel 2 : Taylorpolynom

Interpolationsbedingungen $P^{(k)}(t_0) = f^{(k)}(t_0) \quad , k = 0,1,..., c_0$

$$P(t) = \sum_{k=0}^{c_0} \frac{1}{k!} f^{(k)}(t_0)(t - t_0)^k$$

Bemerkung 7.12 : stückweise Hermite-Interpolation

Problem : Polynome hoher Ordnung neigen zu Oszillation

Kompromiss : Approximierende Funktionen , die nicht unendlich oft stetig differenzierbar sind

Intervallgitter : $\Delta := \{t_i : a = t_0 < t_1 < ... < t_m = b\}$ auf $[a, b]$

Approximation von $f : [a, b] \to R$ durch ein j aus dem Hermiteschen Funktionenraum

$H_\Delta^{(n)} := \{j \in C^{n-1}[a, b]\} \qquad , j|_{[t_i, t_{i+1}]} = p_i|_{[t_i, t_{i+1}]}$ („eingeschränkt auf“)

für ein $p_i \in \Pi_{2n-1} \qquad , i = 0,1,..., m-1$.

praktisch :

Bestimmung der Hermite-Polynome $P_i(t) \qquad , i = 0,1,..., m-1$

mit $P_i^{(k)}(t_i) = f^{(k)}(t_i) \qquad , \qquad P_i^{(k)}(t_{i+1}) = f^{(k)}(t_{i+1}) \qquad , k = 0,1,...,n-1$

$$j\big|_{[t_i,t_{i+1}]} := P_i\big|_{[t_i,t_{i+1}]} \quad , \; P_i \in \Pi_{2n-1}$$

Auswertung von j :

Bestimme i mit $t_i \le t \le t_{i+1}$

Spezialfall : äquidistantes Gitter $t_i := a + \dfrac{i(b-a)}{m}$, $i = 0,1,...,m$

$$i := \left\lfloor \frac{t-a}{h} \right\rfloor \text{ mit } h := \frac{b-a}{m}$$

allgemein : binäre Suche

$\underline{i} := 0$, $\bar{i} := m$;

do

{

$$i_* := \left\lfloor \frac{\underline{i}+\bar{i}}{2} \right\rfloor$$

if $t \ge t_{i_*}$ **then** $\underline{i} := i_*$

else $\bar{i} := i_*$;

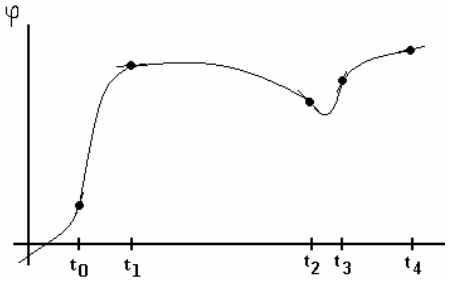
}

while $(\bar{i} - \underline{i} > 1)$;

$i := \underline{i}$;

Beispiel :

$n = 2$, $P_i \in \Pi_3$, $j \in C^1[a,b]$



(Polynome hoher Ordnung neigen zum Schwingen zwischen den Stützstellen.)

Restglied , Fehlerabschätzungen :

Approximation der Ableitungen von f :

$$\left| f^{(k)}(t) - P_i^{(k)}(t) \right| \le \frac{\left|(t-t_i)(t-t_{i+1})\right|^{n-k}}{k!(2n-2k)!} \cdot (t_{i+1}-t_i)^k \cdot \max_{t\in[t_i,t_{i+1}]} \left| f^{(2n)}(t) \right|$$

$$\leq \frac{\|\Delta\|^{2n-k}}{2^{2n-2k} \cdot k!(2n-2k)!} \cdot \max_{t\in[a,b]} \left| f^{(2n)}(t) \right|$$

7.2 Spline-Interpolation

Definition 7.13 : Polynomialer Spline

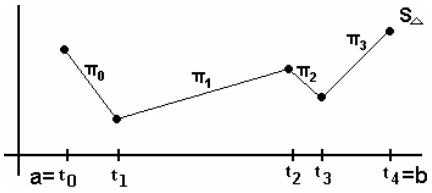
Sei $\Delta = \{t_0,...,t_{l+1}\}$ ein Gitter von $l+2$ paarweise voneinander verschiedenen Knoten t_i mit $a = t_0 < t_1 < ... < t_{l+1} = b$.

Ein polynomialer Spline vom Grad $k-1$ (Ordnung k) bzgl. Δ ist eine Funktion $s_\Delta \in C^{k-2}[a,b]$, die auf jedem Teilintervall $[t_i, t_{i+1}]$ mit einem $p_i \in \Pi_{k-1}$ übereinstimmt, $(i = 0,1,...,l)$.

Bezeichnung : $S_{k,\Delta}$:= Menge der Splines vom Grad $k-1$ bzgl. Δ

Beispiel 7.14 :

lineare Splines : Menge der stetigen , stückweise linearen Funktionen $(S_{2,\Delta})$



kubische Splines : stückweise kubische Funktion , zweimal stetig differenzierbar $(S_{4,\Delta})$

Bemerkung 7.15 : Eigenschaften polynomialer Splines

a) $S_{k,\Delta}$ ist ein Vektorraum

b) $\Pi_{k-1} \subset S_{k,\Delta}$, $\left((t-t_i)^{k-1}\right)_+ \in S_{k,\Delta}$ mit $\left((t-t_i)^{k-1}\right)_+ := \begin{cases} (t-t_i)^{k-1} & \text{falls } t \geq t_i \\ 0 & \text{sonst} \end{cases}$

c) Bedingungen zur eindeutigen Bestimmung eines kubischen Splines :
Interpolationsbedingungen : $s_\Delta(t_i) = f_i$, $(i = 0,1,...,l+1)$

i) $s_\Delta''(a) = s_\Delta''(b) = 0$ (natürlicher kubischer Spline)

ii) $s_\Delta^{(k)}(a) = s_\Delta^{(k)}(b)$, $k = 0,1,2$ (periodischer kubischer Spline – nur sinnvoll , falls $f_0 = f_{l+1}$)

iii) $s_\Delta'(a) = f_a'$, $s_\Delta'(b) = f_b'$ (eingespannter kubischer Spline / vollständiger kub. Spline / kubischer Hermite-Spline)

kubischer Spline :
4 Parameter je Teilintervall
$\Rightarrow 4 \cdot (l+1)$ Koeffizienten

Interpolationsbedingungen = $l+2$
aus C^2 folgen $3l$ Bedingungen
$\Rightarrow \sum = 4(l+1) - 2$

Daher : 2 Bedingungen aus den obigen i)-iii) auswählen

d) Die Bedingungen i)-iii) führen jeweils zu kubischen Splines s_Δ , welche die

Halbnorm $\|s_\Delta\| := \left(\int_a^b (s''(t))^2 dt \right)^{\frac{1}{2}}$ über einer gewissen Klasse von Funktionen minimieren.

(Krümmung wird minimiert.)

Bezeichnung : $K^r[a,b] := \{ f : f^{(k)} \text{ auf } [a,b] \text{ absolutstetig }, (k = 0,1,...,r-1), f^{(r)} \in L^2[a,b] \}$

$v \in L^2[a,b] \Leftrightarrow \int_a^b (v(t))^2 dt$ existiert (siehe Lebesgue/Maßtheorie)

Definition Absolutstetigkeit :

$f : [a,b] \to R$ heißt absolutstetig , falls es zu jedem $e > 0$ ein $d > 0$ gibt , so dass für beliebige a_i, b_i mit $a \le a_1 < b_1 < a_2 < b_2 < ... < b_n \le b$ und $\sum_i |b_i - a_i| < d$ gilt :

$\sum_i |f(b_i) - f(a_i)| < e$.

Absolutstetige Funktionen sind fast überall stetig differenzierbar und es gilt :

$f(t) - f(a) = \int_a^t f'(t) dt$ mit $t \in [a,b]$ sowie $\int_a^b f(t) g'(t) dt = [f(t) g(t)]_a^b - \int_a^b f'(t) g(t) dt$.

Lemma 7.16 : Identität von Holladay

Ist $f \in K^2[a,b]$, $\Delta = \{a = t_0 < t_1 < ... < t_{l+1} = b\}$ und $s_\Delta \in S_{4,\Delta}$, so gilt :

$$\|f - s_\Delta\|^2 = \|f\|^2 - \|s_\Delta\|^2 - 2\left([(f'(t) - s_\Delta'(t)) \cdot s_\Delta''(t)]_a^b - \sum_{i=0}^{l+1} [(f(t) - s_\Delta(t)) \cdot s_\Delta'''(t)]_{t_{i-1+0}}^{t_{i-0}} \right)$$

Satz 7.17 : Splines und Interpolierende minimaler Krümmung

Zu $\Delta = \{a = t_0 < t_1 < ... < t_{l+1} = b\}$ seien Stützwerte $\{y_0, y_1, ..., y_{l+1}\}$ und ein interpolierender Spline $s_\Delta \in S_{4,\Delta}$ gegeben.

i) Ist $s_\Delta''(a) = s_\Delta''(b) = 0$, so gilt für jedes $f \in K^2[a,b]$ mit $f(t_i) = y_i$: $\|f\| \ge \|s_\Delta\|$.

ii) Ist $s_\Delta^{(k)}(a) = s_\Delta^{(k)}(b)$, $(k = 0,1,2)$, so gilt für jedes $f \in K^2[a,b]$ mit $f^{(k)}(a) = f^{(k)}(b)$, $(k = 0,1,2)$ und $f(t_i) = y_i$: $\|f\| \ge \|s_\Delta\|$.

iii) Ist $f \in K^2[a,b]$, $f(t_i) = y_i$ und $f'(a) = s_\Delta'(a)$, $f'(b) = s_\Delta'(b)$, so gilt : $\|f\| \ge \|s_\Delta\|$.

In jedem der drei Fälle ist s_Δ eindeutig bestimmt.

Bemerkung 7.18 : Bestimmung der Splinekoeffizienten (für kubische Splines)

$$S_{4,\Delta}: \quad s_\Delta\big|_{[t_i,t_{i+1}]} = a_i + b_i(t-t_i) + \frac{1}{2}c_i(t-t_i)^2 + \frac{1}{6}d_i(t-t_i)^3 \qquad \text{mit}$$

$a_i = s_\Delta(t_i)$ (durch Interpolationsbedingungen vorgegeben)
$b_i = s_\Delta'(t_i)$
$c_i = s_\Delta''(t_i)$
$d_i = s_\Delta'''(t_i)\big|_{t\to t_i+0}$

Aus $s_\Delta''(t)\big|_{[t_i,t_{i+1}]}$ ist linear folgt : $s_\Delta''(t) = c_{i+1}\frac{t-t_i}{h_i} - c_i\frac{t-t_{i+1}}{h_i}$ mit $h_i := t_{i+1}-t_i$

$\Rightarrow$

$$s_\Delta(t) = c_{i+1}\frac{(t-t_i)^3}{6h_i} - c_i\frac{(t-t_{i+1})^3}{6h_i} + B_i(t-t_i) + A_i \qquad \left(\iint \text{ von } s_\Delta''\right)$$

Interpolationsbedingungen :

$$s_\Delta(t_i) = c_i\frac{h_i^2}{6} + A_i \overset{!}{=} f_i$$

$$s_\Delta(t_{i+1}) = c_{i+1}\frac{h_i^2}{6} + B_i h_i + A_i \overset{!}{=} f_{i+1}$$

$\Rightarrow$

$$A_i = f_i - \frac{h_i^2}{6}c_i$$

$$B_i = \frac{f_{i+1}-f_i}{h_i} - \frac{h_i}{6}(c_{i+1}-c_i)$$

Ausmultiplizieren von $(t-t_{i+1})^3 = \left((t-t_i) - \underbrace{(t_{i+1}-t_i)}_{=h_i}\right)^3$ und Koeffizientenvergleich ergibt :

$$a_i = f_i$$

$$b_i = \frac{f_{i+1}-f_i}{h_i} - \frac{h_i}{6}(2c_i + c_{i+1})$$

$$d_i = \frac{c_{i+1}-c_i}{h_i}$$

Bestimmung der c_i : aus den Stetigkeitsbedingungen für $s_\Delta'(t)$

$$s_\Delta'(t)\big|_{t_i+0} = b_i = \frac{f_{i+1}-f_i}{h_i} - \frac{h_i}{6}(2c_i + c_{i+1})$$

$$s_\Delta'(t)\big|_{t_i-0} = \left(b_{i-1} + h_{i-1}c_{i-1} + \frac{h_{i-1}^{\,2}}{2} d_{i-1} \right) = \frac{f_i - f_{i-1}}{h_{i-1}} - \frac{h_{i-1}}{6}(2c_{i-1} + c_i) + h_{i-1}c_{i-1} + \frac{h_{i-1}}{2}(c_i - c_{i-1})$$

$$= \frac{f_i - f_{i-1}}{h_{i-1}} + \frac{h_{i-1}}{6} c_{i-1} + \frac{h_i}{3} c_i$$

$\Rightarrow$

$$\frac{h_{i-1}}{6} c_{i-1} + \frac{h_{i-1} + h_i}{3} c_i + \frac{h_{i+1}}{6} c_{i+1} = \frac{f_{i+1} - f_i}{h_i} - \frac{f_i - f_{i-1}}{h_{i-1}} \qquad ,(i = 1,...,l)$$

mit $c_{l+1} := s_\Delta''(b)$

natürlicher kubischer Spline : $c_0 = c_{l+1} = 0$

$$\begin{pmatrix} \frac{h_0 + h_1}{3} & \frac{h_1}{6} & & \\ \frac{h_1}{6} & \frac{h_1 + h_2}{3} & \frac{h_2}{6} & \\ & \ddots & \ddots & \\ & & \frac{h_{l-1}}{6} & \frac{h_{l-1} + h_l}{3} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_l \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{f_2 - f_1}{h_1} - \frac{f_1 - f_0}{h_0} \\ \frac{f_3 - f_2}{h_2} - \frac{f_2 - f_1}{h_1} \\ \vdots \\ \frac{f_{l+1} - f_l}{h_l} - \frac{f_l - f_{l-1}}{h_l} \end{pmatrix}}_{= MATLAB-\text{Befehl "diff(f) "}}$$

Lösung des Gleichungssystems mittels Cholesky-Zerlegung für tridiagonale Matrizen.

Fragen : - Eindeutige Lösbarkeit ?
- Koeffizientenmatrix positiv definit ?

<u>Lemma 7.19 : Ein Kriterium für die Regularität von Matrizen</u>

Gilt für eine Matrix $A \in R^{n \times n}$ mit $a_{ii} = 2$ und $\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \leq 1 \qquad ,(i = 1,...,n)$,

so ist A regulär und $\|A^{-1}\|_\infty \leq 1$.

<u>Beweis :</u>

a) Für Vektoren $w, z \in R^n$ mit $w = Az$ bestimmt man $i_0 \in \{1,...,n\}$ so, dass $|z_{i_0}| = \max_{1 \leq i \leq n} |z_i| = \|z\|_\infty$.

Dann ist $\|w\|_\infty \geq |w_{i_0}| = \left| \sum_{j=1}^{n} a_{i_0,j} \cdot z_j \right| \geq |a_{i_0,i} \cdot z_{i_0}| - \left| \sum_{\substack{j=1 \\ j \neq i_0}}^{n} a_{i_0,j} \cdot z_j \right|$

$$\geq 2|z_{i_0}| - \sum_{\substack{j=1 \\ j \neq i_0}}^{n} \left(|a_{i_0,j}| \cdot \max_{1 \leq j \leq n} |z_j| \right)$$

$$\geq (2-1)\|z\|_\infty = \|z\|_\infty \qquad \Rightarrow \text{Eindeutigkeit}$$

b) Im homogenen linearen Gleichungssystem $Az = 0$ ist :

$\|w\|_\infty = 0 \Rightarrow \|z\|_\infty = 0 \Rightarrow z = 0 \Rightarrow A$ regulär

c) $\|A^{-1}\|_\infty = \sup_{w \neq 0} \frac{\|A^{-1} w\|_\infty}{\|w\|_\infty} = \sup_{w \neq 0} \frac{\|z\|_\infty}{\|w\|_\infty} \leq 1$ (wegen a))

Folgerung 7.20 : Eindeutigkeit des interpolierenden kubischen Splines

Duch Vorgabe von genau einer der Bedingungen i)-iii) aus Bem. 7.15 ist jeweils ein interpolierender kubischer Spline eindeutig bestimmt.

Bemerkung 7.21 : Konvergenzeigenschaften kubischer Splines

Zum vollständigen kubischen Spline ergibt sich das LGS $A_\Delta c_\Delta = b_\Delta$ mit

$A_\Delta \in R^{(l+2)\times(l+2)}$, $c_\Delta = (c_0, c_1, ..., c_l, c_{l+1})^T \in R^{l+2}$, $b_\Delta \in R^{l+2}$

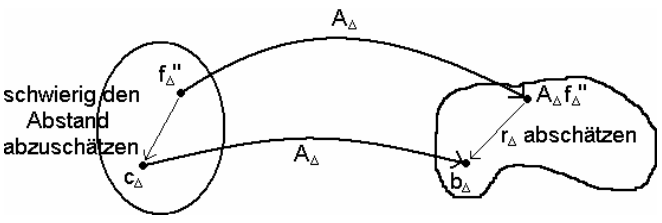
$$A_\Delta = \begin{pmatrix} 2 & 1 & & & \\ \frac{h_0}{h_0+h_1} & 2 & \frac{h_1}{h_0+h_1} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{h_{l-1}}{h_{l-1}+h_l} & 2 & \frac{h_l}{h_{l-1}+h_l} \\ & & & 1 & 2 \end{pmatrix} \quad , \quad b_\Delta = \begin{pmatrix} \frac{6}{h_0}\left(\frac{f_1 - f_0}{h_0} - f_0'\right) \\ \frac{6}{h_0+h_1}\left(\frac{f_2 - f_1}{h_1} - \frac{f_1 - f_0}{h_0}\right) \\ \vdots \\ \frac{6}{h_{l-1}+h_l}\left(\frac{f_{l+1} - f_l}{h_l} - \frac{f_l - f_{l-1}}{h_{l-1}}\right) \\ \frac{6}{h_l}\left(f_{l+1}' - \frac{f_{l+1} - f_l}{h_l}\right) \end{pmatrix}$$

mit $f_0' := f'(a)$ und $f_{l+1}' := f'(b)$.

Abschätzung für $\max_i |c_i - f''(t_i)|$?

Idee : Einsetzen von $f''(t_i)$ in $A_\Delta c_\Delta = b_\Delta$ und Abschätzung des Residuums $r_\Delta := b_\Delta - A_\Delta f_\Delta''$

mit $f_\Delta'' := (f''(t_0), f''(t_1), ..., f''(t_{l+1}))^T \in R^{l+2}$.



Satz 7.22 : Approximationsgüte der zweiten Ableitung

Mit den Bezeichnungen aus Bem. 7.21 gilt für den vollständigen (eingespannten) interpolierten kubischen Spline :

$$\left\| c_\Delta - f_\Delta '' \right\|_\infty \le \left\| r_\Delta \right\|_\infty \le \frac{3}{4} M \cdot \left\| \Delta \right\|^2 \quad ,$$

sofern $f \in C^4[a,b]$ und $\left| f^4(t) \right| \le M$ mit $(t \in [a,b])$ ist.

Satz 7.23 : Approximationsgüte vollständiger kubischer Splines

Gegeben sei eine Funktion $f \in C^4[a,b]$ mit $\left\| f^{(4)} \right\|_\infty \le M$, sowie ein Gitter $\Delta = \{a = t_0 < t_1 < ... < t_{l+1} = b\}$

mit einer Konstanten $K \ge \dfrac{\max\limits_{0 \le i \le l} h_i}{\min\limits_{0 \le i \le l} h_i}$.

Dann gibt es zum vollständigen interpolierenden kubischen Spline s_Δ Konstanten C_0, C_1, C_2, C_3 die von Δ und K unabhängig sind , und für die gilt :

$$\left| f^{(4)}(t) - s_\Delta{}^{(4)}(t) \right| \le C_k \cdot M \cdot K \cdot \left\| A \right\|^{4-k} \quad ,(k = 0,1,2,3)$$

mit $t \in [a,b]$, sofern $s_\Delta{}^{(k)}$ in t definiert ist.

Folgerung 7.24 : Gleichmäßige Konvergenz interpolierender Splines

Zu einer Gitterfolge $(\Delta_m)_{m \ge 0}$ mit $\left\| \Delta_m \right\| \to 0$ und $\sup\limits_{m,j} \dfrac{\left\| \Delta_m \right\|}{h_j{}^{(m)}} \le K < \infty$ konvergieren die vollständigen interpolierenden kubischen Splines $\left(s_{\Delta_m} \right)_{m \ge 0}$ gleichmäßig auf $[a,b]$ gegen $f \in C^4[a,b]$, falls $\left\| \Delta_m \right\| \to 0$.

Ebenso konvergieren die Ableitungen von s_{Δ_m} gleichmäßig gegen die entsprechenden Ableitungen von f .

Bemerkung 7.25 : B-Splines

Statt der expliziten Berechnung der Splinekoeffizienten ist häufig eine Darstellung $s_\Delta(t) = \sum\limits_{i=0}^{l+k-1} a_i B_i(t)$ mit einer geeigneten Basis $\{B_0, B_1, ..., B_{l+k-2}, B_{l+k-1}\}$ von $s_{k,\Delta}$ mit Koeffizienten $a_i \in R$ vorteilhaft.

Beispiel :

- approximierender Spline $s_\Delta(t_i) \approx f_i \quad ,(i = 0,1,...,m)$ mit $m >> l$

$$\sum_{i=0}^{m} \left(s_\Delta(t_i) - f_i \right)^2 \to \min$$

$\Rightarrow a_i$ als Lösung eines überbestimmten linearen Gleichungssystems. (vgl. Bem. 4.8)

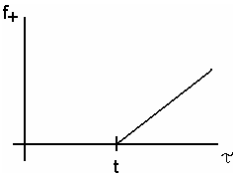
- Spline s_Δ mit zusätzlichen Forderungen wie Nichtnegativität , Monotonie , Konvexität , ...

Definition : B-Splines

Zu einer Knotenfolge $\ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_{i+r}, t_{i+r+1}$ mit $t_j \le t_{j+1} \quad , (j \in Z)$ und $t_i < t_{i+r}$

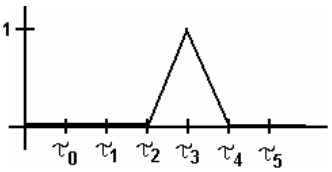
heisst $B_{i,r}(t) := (t_{i+r} - t_i) f_t^{\,r-1}[t_i, t_{i+1}, \ldots, t_{i+r}]$

mit $f_t^{\,r-1}(t) = (f_t(t))^{r-1}$, $f_t(t) := (t-t)_+ := \begin{cases} t - t & \text{für } t > t \\ 0 & \text{sonst} \end{cases}$



i-ter Spline der Ordnung k .

Beispiel :



$B_{3,2}(t)$ - „Hütchenfunktion“

$r = 1$:

$$f_t^{\,r-1}(t) = (f_t(t))^0 = \begin{cases} 1 \text{ für } t > t \\ 0 \quad \text{sonst} \end{cases}$$

$$f_t^{\,r-1}[t_i, t_{i+1}] = \begin{cases} \dfrac{1-1}{t_{i+1} - t_i} = 0 & \text{für } t < t_i \\ \dfrac{1-0}{t_{i+1} - t_i} & \text{für } t_i \le t \le t_{i+1} \\ \dfrac{0-0}{t_{i+1} - t_i} = 0 & \text{für } t > t_{i+1} \end{cases}$$

Darstellung : $s_\Delta(t) = \sum_i a_i B_i(t)$

Rekursive Berechnung :

Unter Verwendung der Produktregel für dividierte Differenzen Newtonsche Darstellung des Interpolationspolynoms von $f_t^{\,r-1}(t)$ zu Stützstellen $t_i, t_{i+1}, \ldots, t_{i+r}$:

$$p_{r-1}(t) = \sum_{s=i}^{i+r} \left(f_t^{\,r-1}[t_i, \ldots, t_s] \cdot (t - t_i) \cdot \ldots \cdot (t - t_{s-1}) \right) \in \Pi_r$$

Für $t \in [t_i, \ldots, t_{i+r}]$ kann $f_t^{\,r-1}[t_i, \ldots, t_{i+r}]$ numerisch stabil als konvexe Linearkombination von $f_t^{\,r-2}[t_i, \ldots, t_{i+r-1}]$ und $f_t^{\,r-2}[t_{i+1}, \ldots, t_{i+r}]$ berechnet werden.

Für $t < t_i$ und für $t_{i+r} < t$ fällt $f_t^{\,r-1}(t)$ in $[t_i, t_{i+r}]$ mit einem Polynom vom Grad $\leq r-1$ zusammen $\Rightarrow f_t^{\,r-1}[t_i,...,t_{i+r}] = 0$.

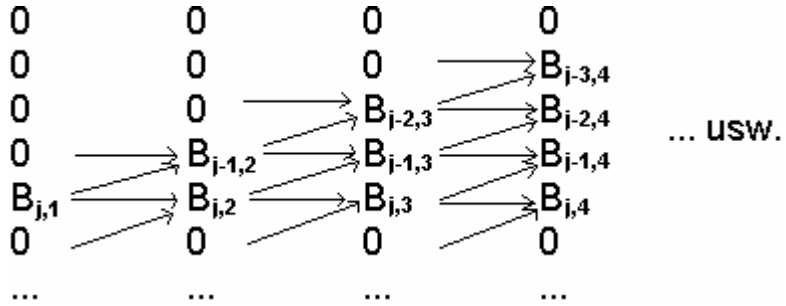
Ist $t_i < t_{i+r-1}$ und $t_{i+1} < t_{i+r}$, so folgt :

$$B_{i,r}(t) = \frac{t - t_i}{t_{i+r-1} - t_i} B_{i,r-1}(t) + \frac{t_{i+1} - t}{t_{i+r} - t_{i+1}} B_{i+1,r-1}(t). \qquad \textbf{(*)}$$

Zu gegebenem $t \in R$ gibt es höchstens r B-Splines $B_{i,r}(t) \neq 0$.

Sei j so gewählt , dass $t \in [t_j, t_{j+1}] \Rightarrow B_{i,r}(t) = 0$, für $i \leq j - r$ oder $i \geq j + 1$.

Schema :



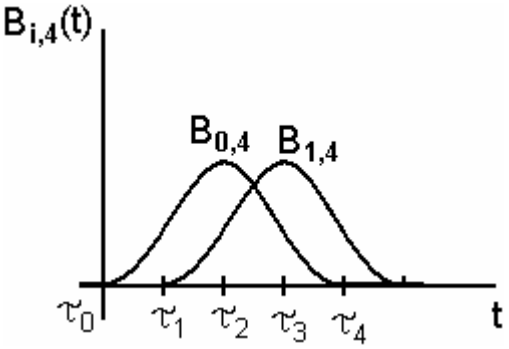
Beispiel :

$$t_i = i \ , (i = 0,1,2,...) \ , \quad t = 3.5 \in [t_3, t_4] \ , \ j = 3 \qquad \text{(siehe *)}$$

$i \setminus r$	1	2	3	4
0	0	0	0	1/48
1	0	0	1/8	23/48
2	0	1/2	3/4	23/48
3	1	1/2	1/8	1/48
4	0	0	0	0

z.B. : $B_{2,3}(3.5) = \frac{3.5-2}{4-2} B_{2,2}(3.5) + \frac{5-3.5}{5-3} B_{3,2}(3.5) = \frac{1.5}{2} \cdot \frac{1}{2} + \frac{1.5}{2} \cdot \frac{1}{2} = \frac{3}{8} + \frac{3}{8} = \frac{3}{4}$

$r = k = 4$: $\quad s_\Delta(3.5) = \sum_i a_i B_{i,4}(3.5) = \frac{a_0 + 23a_1 + 23a_2 + a_3}{48}$



Eigenschaften :

a) Nichtnegativität $B_{i,r}(t) \geq 0 \quad , \quad t \in R$

b) Lokaler Träger $\text{supp}\, B_{i,r} \subset [t_i, t_{i+r}]$

c) $\sum_i B_{i,r}(t) \equiv 1 \quad , \quad t \in R$

d) Zum Gitter $\Delta = \{t_i : a = t_0 < t_1 < ... < t_{l+1} = b\}$ betrachte die Knotenfolge

$t_{-k+1} = t_{-k+2} = ... = t_{-1} = t_0 := t_0 = a$

$t_j = t_j \quad ,(j = 1,...,l)$

$t_{l+1} = t_{l+2} = ... = t_{l+k} := t_{l+1} = b$

und die B-Splines $B_{-k+1,k}(t)$, $B_{-k+2,k}(t)$, ... , $B_{l,k}(t)$

Satz : (Curry , Schoenberg)

Zu $k = 4$ betrachte $l + k$ Interpolationsbedingungen $f_j = \sum_{i=-k+1}^{l} a_i B_{i,k}(x_j) \qquad ,(j = -k+1,...,l)$

$\Rightarrow$ Lineares Gleichungssystem der Dimension $l + k$ mit Bandmatrix der Breite k zur Bestimmung von $a_{-k+1},...,a_l$.

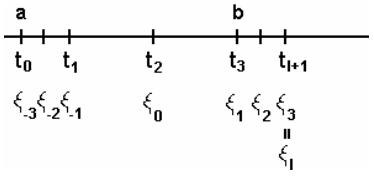
Elemente der Koeffizientenmatrix : $B_{i,k}(x_j)$ aus obigem Rechenschema ,

zweckmäßig : $x_{-k+1} < x_{-k+2} < ... < x_l$.

Satz : (Schoenberg , Whitney)

Der interpolierende Spline ist genau dann eindeutig bestimmt , wenn $B_{i,k}(x_j) \neq 0 \quad ,(i = -k+1,...,l)$.

praktisch : für $k = 4$



z.B. für gerades k :

$x_{-k+1} := t_0 \quad , \quad x_j := t_{j+k/2} \qquad ,\left(j = -k/2 + 1 , ... , -k/2 + l\right) \qquad d.h.\ t_1,...,t_l$

zusätzliche Interpolationsbedingungen :

$x_{-k+1+j} \in (t_0, t_1) \quad , \quad x_{l-j} \in (t_l, t_{l+1}) \qquad ,\left(j = 1 , ... , k/2 - 1\right)$

7.3 Trigonometrische Interpolation

Problemstellung : Interpolation periodische Daten durch trigonometrische Polynome

$N = 2n+1:$ (ungerade)

$$T_R^N := \left\{ \Phi_{2n+1}(t) := \frac{a_0}{2} + \sum_{j=1}^{n} \left(a_j \cos(jt) + b_j \sin(jt) \right) \right\} \qquad , R \text{ bedeutet : } a_j, b_j \in R$$

$N = 2n:$ (gerade)

$$T_R^N := \left\{ \Phi_{2n}(t) := \frac{a_0}{2} + \left(\sum_{j=1}^{n-1} \left(a_j \cos(jt) + b_j \sin(jt) \right) \right) + \frac{a_n}{2} \cos(nt) \right\}$$

Die komplexe Darstellung kommt ohne Fallunterscheidung aus :

$$T_C^N := \left\{ \Phi_N(t) := \sum_{j=0}^{N-1} c_j \cdot e^{ijt} \quad , c_j \in C \right\}$$

Die Ansatzfunktionen $\{1, \cos(jt), \sin(jt)\}$ bzw. $\{e^{ijt}\}$ sind linear unabhängig.

$\Rightarrow$ Die Interpolationsaufgabe zu N Stützstellen (t_k, f_k) $,(k = 0,1,..., N-1)$ ist eindeutig lösbar.

Typische Aufgabenstellung :
Digitale Datenverarbeitung , Datenerfassung im festen Takt

$\Rightarrow$ äquidistante Knoten , normiert auf $t_k = k \cdot \frac{2p}{N}$

Bemerkung 7.27 : Trigonometrische Interpolation auf äquidistantem Gitter

Zu $t_k = k \cdot \frac{2p}{N}$ sind $w_k := e^{it_k} = e^{ik\frac{2p}{N}}$ die N-ten Einheitswurzeln $\left((w_k)^n = 1\right)$

Die Interpolationsbedingungen $\Phi_N(t_k) = f_k$ $,(k = 0,..., N-1)$ bestimmen Φ_N mit :

$$\begin{pmatrix} 1 & w_0 & {w_0}^2 & \cdots & {w_0}^{N-1} \\ 1 & w_1 & {w_1}^2 & \cdots & {w_1}^{N-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & w_{N-1} & {w_{N-1}}^2 & \cdots & {w_{N-1}}^{N-1} \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix} \qquad ,(\det \neq 0)$$

Lemma 7.28 : Komplexe und reelle trigonometrische Polynome

Gilt für das (komplexe) trigonometrische Polynom $\Phi_N(t) = \sum_{j=0}^{N-1} c_j e^{ijt}$ mit $\Phi_n(t) \in R$, $t \in R$, so gilt :

$\Phi_N \in T_R^N$ mit $a_j = 2\,\mathrm{Re}(c_j) = c_j + c_{N-j}$ und $b_j = -2\,\mathrm{Im}(c_j) = i(c_j - c_{N-j})$

Lemma 7.29 : Eigenschaften der Basisfunktionen

a) Für die N-ten Einheitswurzeln $w_j := e^{ij\frac{2p}{N}}$ gilt : $\sum_{j=0}^{N-1} w_j{}^k w_k{}^{-l} = N \cdot d_{k,l}$.

b) Die Basisfunktionen $y_k(t) := e^{ikt}$ der (komplexen) trigonometrischen Interpolation sind orthonormal bezüglich des diskreten Skalarprodukts $< f, g >:= \frac{1}{N}\sum_{j=0}^{N-1}\left(f(t_j)\cdot\overline{g(t_j)}\right)$, d.h. $< y_k, y_l >= d_{k,l}$.

Satz 7.30 : Lösung der trigonometrischen Interpolationsaufgabe

Für äquidistante Stützstellen $t_k = k\frac{2p}{N}$ ist die Lösung der trigonometrischen Interpolationsaufgabe $\Phi_n(t_k) = f_k$ gegeben durch :

$$\Phi_n(t) = \sum_{j=0}^{N-1} c_j e^{ijt} \qquad \text{mit } c_j := \frac{1}{N}\sum_{k=0}^{N-1} f_k w_k{}^{-j}$$

Beweis :

$$\Phi_n(t_l) = \sum_{j=0}^{N-1} c_j e^{ijt_l} = \sum_{j=0}^{N-1} c_j \underbrace{\left(e^{il\frac{2p}{N}}\right)^j}_{=w_l} \qquad \text{- wegen } t_l = l\frac{2p}{N}$$

$$= \sum_{j=0}^{N-1}\left(\frac{1}{N}\sum_{k=0}^{N-1} f_k w_k{}^{-j}\right)\cdot w_l{}^j \qquad \text{- } c_j \text{ eingesetzt}$$

$$= \sum_{j=0}^{N-1} f_k \cdot \left(\frac{1}{N}\sum_{j=0}^{N-1} w_k{}^{-j} w_l{}^j\right)$$

$$= \sum_{j=0}^{N-1} f_k \cdot d_{k,l} \qquad \text{- nach Lemma 7.29a) mit } w_k{}^{-j} = w_j{}^{-k} \text{ , } w_l{}^j = w_j{}^l$$

$$= f_l \qquad ,(l = 0,1,...,N-1)$$

Bemerkung 7.31 : Diskrete Fourier-Transformation

Für $2p$ - periodische Funktionen $f \in L^2(R)$ erhält man mit den Fourierkoeffizienten

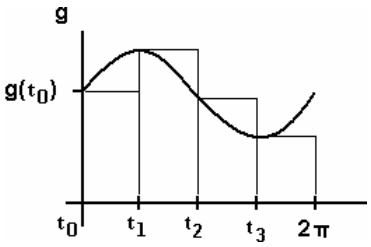
$$\hat{f}(j) := \frac{1}{2p}\int_0^{2p} f(t)e^{-ijt}dt \qquad , j \in Z$$

die abgebrochene Fourier-Reihe $f_n(t) := \sum_{j=-n}^{n} \hat{f}(j)e^{ijt}$.

Setzt man für $N = 2n+1$: $c_{-j} := c_{N-j}$, $j = 1,...,n$, so ist

$$\Phi_N(t_k) = \sum_{j=0}^{N-1} c_j e^{ijt_k} = \sum_{j=-n}^{n} c_j e^{ijt_k} \ .$$

Approximiert man $\int_0^{2p} g(t)dt \approx \frac{2p}{N}\sum_{k=0}^{N-1} g(t_k)$,



so ergibt sich : $\hat{f}(j) \approx \frac{1}{N}\sum_{k=0}^{N-1} f(t_k)e^{-ijt_k} = \frac{1}{N}\sum_{k=0}^{N-1} f_k w_k^{-j} = c_j$ (letzter Schritt nach 7.30)

Die Abbildung $F_N : C^N \to C^N$ mit $(f_k)_{k=0}^{N-1} \mapsto (c_j)_{j=0}^{N-1}$ aus Satz 7.30 heißt Diskrete Fourier-Transformation.

Die Umkehrabbildung $F_N^{-1} : (c_j)_{j=0}^{N-1} \to (f_k)_{k=0}^{N-1}$ heißt inverse DFT oder Fourier-Synthese.

$$\left(f_k := \sum_{j=0}^{N-1} c_j w_j^{\,k} \quad ,(k = 0,1,...,N-1) \right)$$

<u>Anwendung in der Signalverarbeitung :</u>
Elimination hochfrequenter Komponenten , die häufig durch Messfehler verfälscht (Messrauschen) und darüber hinaus die praktische Anwendung oft nicht relevant ist.
„Abschneiden“ hoher Frequenzen bei Hinzunahme der ersten $j_{\max}$ Frequenzen :

$$\tilde{f}_k := \sum_{j=0}^{j_{\max}} c_j w_j^{\,k} + \sum_{j=1}^{j_{\max}} c_{N-j} w_{N-j}^{\,k} \qquad , (k = 0,1,...,N-1)$$

Ist das Eingangssignal mit der Frequenz f_{input} abgetastet , so enthält das Ausgangssignal alle Frequenzen

bis $\bar{f}_{output} := \frac{j_{\max}}{\frac{N}{2}} \cdot f_{input}$.

Matlab-Befehle : fft , ifft

<u>Bemerkung 7.32 : Schnelle Fourier-Transformation (FFT)</u>

Problem : Standard-Algorithmus zur Auswertung von F_N oder F_N^{-1} erfordert $O(n^2)$ Rechenoperationen.
(Matrix-Vektor-Multiplikation)

<u>Algorithmus : Cooley-Tuckey :</u>

Sei $N = 2M$ gerade und $w = e^{i\frac{2p}{N}}$.

Dann gilt für $a_j = \sum_{k=0}^{N-1} f_k w^{kj}$: $a_{2l} = \sum_{k=0}^{M-1} g_k x^{kl}$ und $a_{2l+1} = \sum_{k=0}^{M-1} h_k x^{kl}$

mit $M = \frac{N}{2}$ und $x := w^2$ und $g_k := f_k + f_{k+M}$ und $h_k := (f_k - f_{k+M}) \cdot w^k$

Mit $2M$ Additionen und $2M$ Multiplikationen $\left(w^k \to w^{k+1} = w \cdot w^k \quad , \quad h_k = (...) \cdot w^k\right)$ wird die Berechnung von N Summen der Länge N zurückgeführt auf $2M = N$ Summen der Länge $M = \frac{N}{2}$.

Die rekursive Anwendung ist besonders einfach für $N = 2^p$
$\Rightarrow$
Der Aufwand zur Berechnung von $a_0, a_1, ..., a_{N-1}$ (Analyse oder Synthese)
beträgt $2N \log_2 N$ Multiplikationen.

Für $N = 8192$ ist $O(n^2) \approx 64 \cdot 10^6$ und $2N \log_2 N \approx 10^5$.

Speicherschema :
wünschenswert : Überspeicherung von (f_k) durch (a_j) ohne aufwendigen Komponententausch , jedoch unter Beibehaltung der einfachen Termstruktur für a_{2l} und a_{2l+1}.

Problem : Einfaches Überspeichern tauscht Komponenten.
Lösung : "Bit-Reversal" – Reihenfolge der Bits umdrehen

k	k_{dual}	$N = 8 \to M = 4$	$N = 4 \to M = 2$	k_{dual}
0	000	0	0	000
1	001	2	4	100
2	010	4	2	010
3	011	6	6	110
4	100	1	1	001
5	101	3	5	101
6	110	5	3	011
7	111	7	7	111

Permutation $s : \{0,1,...,N-1\} \to \{0,1,...,N-1\}$ mit $s\left(\sum_{l=0}^{N-1} a_l 2^l\right) := \sum_{l=0}^{N-1} a_{N-1-l} 2^l \qquad , a_i \in \{0,1\}$

Einfache Implementierung durch Bit-Manipulationen.

Algorithmus 7.33 : Schnelle Fourier-Transformation (FFT)

Sei $N = 2^p$ und $w = e^{i\frac{2p}{N}}$ oder $w = e^{-i\frac{2p}{N}}$.

Eingabe : $f_0, f_1, ..., f_{N-1} \in C$

Ausgabe : $a_0, a_1, ..., a_{N-1} \in C$ mit $a_j := \sum_{k=0}^{N-1} f_k w^{kj}$

Algorithmus :

$N_{red} := n;$ $z := w;$

while $N_{red} > 1$ **do**

$M_{red} := N_{red} / 2;$

for $j = 0 : (N / N_{red} - 1)$ **do**

$l := j \cdot N_{red};$

for $k = 0 : (M_{red} - 1)$ **do** <- g's und h's ausrechnen

$a := f_{l+k} + f_{l+k+M_{red}};$

$f_{l+k+M_{red}} := (f_{l+k} - f_{l+k+M_{red}}) \cdot z^k;$

$f_{l+k} := a;$

endfor;

endfor;

$N_{red} := M_{red};$

$z := z^2;$

endwhile;

for $k = 0 : (N - 1)$

$a_{s(k)} := f_k;$

Die FFT ist typisches Beispiel eines 'divide-and-conquer' – Algorithmus.
(Ist in Signalprozessoren hardwaremäßig implementiert.)

Bem. 7.34 : Auswertung trigonometrischer Polynome

Auswertung von $\sum_{j=1}^{n} a_j \cos(jt)$, $\sum_{j=1}^{n} b_j \sin(jt)$?

Fall $t = t_k$ $\Rightarrow$ iFFT (Synthese)

Fall $t \neq t_k$:

Algorithmus von Goertzel :

Für $t \neq rp$, $(r = 0, \pm 1, \pm 2, ...)$ berechnen sich

$$U_j := \frac{1}{\sin t} \sum_{k=j}^{n-1} y_k \cdot \sin\big((k - j + 1) \cdot t\big) \quad , (j = 1, ..., n-1)$$

mit $U_1 \cdot \sin t = \sum_{k=1}^{n-1} y_k \sin(kt)$ und $U_1 \cdot \cos t - U_2 = \sum_{k=1}^{n-1} y_k \cos(kt)$ rekursiv aus :

$$U_j := y_j + 2U_{j+1} \cdot \cos t - U_{j+2} \qquad , \ (j = n-1, n-2, ..., 1)$$

mit den zusätzlich initialisierten $U_{n+1} = U_n := 0$

Vorteil : Statt (n-1)mal muss der Winkelausdruck nur 1mal ausgewertet werden.

Nachteil : Obige Drei-Term-Rekursion ist numerisch instabil für $|t| << 1$:

Für $|t| << 1$ ist die Verwendung von $\cos t$ in der rekursiven Berechnung der U_j mittels der Drei-Term-Rekursion nachteilig. (vgl. Bsp. 3.32 Numerik I)
Relative Fehler $O(u)$ in $c = \cos t$ entsprechen einem absoluten Fehler $O(u)$ in t , hingegen entspricht einem relativen Fehler $O(u)$ in t ein absoluter Fehler $O(u \cdot |t|) << O(u)$ in t.

$\Rightarrow$ Verwendung von $\cos t$ für $|t| << 1$ vermeiden !

$\Rightarrow$ Verbesserung : gekoppelte Zwei-Term-Rekursion :

Algorithmus von Reinsch

Fall $\cos t > 0$:

$U_{n+1} := 0$
$\Delta U_n := 0$
for $j = n-1 : -1 : 0$
$U_{j+1} := U_{j+2} + \Delta U_{j+1}$
$\Delta U_j := y_j + \Delta U_{j+1} - 4\sin^2\left(\frac{t}{2}\right) \cdot U_{j+1}$ **(*)**
endfor

Fall $\cos t \le 0$:

$U_{n+1} := 0$
$\Delta U_n := 0$
for $j = n-1 : -1 : 0$
$U_{j+1} := -U_{j+2} + \Delta U_{j+1}$
$\Delta U_j := y_j - \Delta U_{j+1} + 4\cos^2\left(\frac{t}{2}\right) \cdot U_{j+1}$
endfor

beachte : Der Winkelausdruck muss nur 1mal berechnet werden

$\Rightarrow$

$$\sum_{k=1}^{n-1} y_k \sin(kt) = U_1 \sin t$$

$$\sum_{k=1}^{n-1} y_k \cos(kt) = \Delta U_0 + 2 \cdot \left\{ \begin{array}{l} \sin^2\left(\frac{t}{2}\right) \text{ für } \cos t > 0 \\ -\cos^2\left(\frac{t}{2}\right) \text{ für } \cos t \le 0 \end{array} \right\} \cdot U_1$$

Begründung für (*) :

$$\Delta U_j - \Delta U_{j+1} = U_j - U_{j+1}$$

$$-(U_{j+1} - U_{j+2}) = U_j + U_{j+2} - 2U_{j+1}$$

$$\overset{\text{Goertzel}}{=} y_j + 2(\cos t - 1)\cdot U_{j+1}$$

$$= y_j + 2\left(\cos\left(\frac{t}{2}+\frac{t}{2}\right)-1\right)\cdot U_{j+1}$$

$$= y_j + 2\left(\cos^2\left(\frac{t}{2}\right)-\sin^2\left(\frac{t}{2}\right)-1\right)\cdot U_{j+1}$$

$$= y_j + 2\left(\left(1-\sin^2\left(\frac{t}{2}\right)\right)-\sin^2\left(\frac{t}{2}\right)-1\right)\cdot U_{j+1}$$

$$= y_j + 2\left(-2\sin^2\left(\frac{t}{2}\right)\right)\cdot U_{j+1}$$

$$= y_j - 4\sin^2\left(\frac{t}{2}\right)\cdot U_{j+1}$$

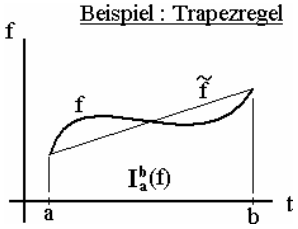
8. Numerische Integration reeller Funktionen

Problemstellung :

geg. : $f : [a,b] \to R$ stückweise stetig

ges. : Näherung für $I(f) := I_a^b(f) = \int_a^b f(t)dt$

Beispiel : Trapezregel



Berechnung numerischer Integrale = Quadratur / Quadraturformel

Idee :

Ersetze f durch eine „einfachere“ Funktion $\tilde{f}$ und verwende $I_a^b(f) \approx \int_a^b \tilde{f}(t)dt$,

wobei $\tilde{f}$ eine Interpolation von f ist , durch Polynome , Splines , ...

Eigenschaften von $I(f)$:

a) Linearität : $I(af + bg) = a \cdot I(f) + b \cdot I(g)$

b) Positivität : $f \geq 0 \Rightarrow I(f) \geq 0$

c) Additivität : $I_a^c(f) + I_c^b(f) = I_a^b(f)$ für ein $c \in (a,b)$

Bem 8.2 : Kondition der Quadratur

Wegen $\left| \int_a^b (f + d_f)dt - \int_a^b f \; dt \right| = \left| \int_a^b d_f(t) \; dt \right| \leq \int_a^b |d_f(t)| dt = \|d_f\|_1$ hat die Auswertung von $I(f)$ bzgl. der L_1-Norm die Kondition $k_{abs} = 1$ und $k_{rel} = \dfrac{I(|f|)}{|I(f)|}$.

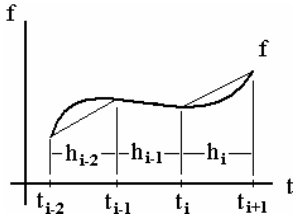
In Analogie zur Auslöschung bei Subtraktion zweier annähernd gleichgroßer Zahlen ist der kritische Fall durch (stark) oszillierende Funktionen f gegeben.

8.1 Newton-Cotes-Quadratur

Bem 8.3 : Zusammengesetzte Trapezregel bzw. Trapezsummen

Gitter $\Delta = \{a = t_0 < t_1 < ... < t_n = b\}$ mit Schrittweiten $h_i := t_{i+1} - t_i$ $,(i = 0,...,n-1)$

$$I_a^b(f) \approx T^{(n)} := \sum_{i=0}^{n-1} h_i \cdot \frac{f(t_i) + f(t_{i+1})}{2}$$



Wegen $\min_{t_i \leq t \leq t_{i+1}} f(t) \leq \dfrac{f(t_i) + f(t_{i+1})}{2} \leq \max_{t_i \leq t \leq t_{i+1}} f(t)$ ist die Folge der $T^{(n)}$ nach oben und unten durch die Riemannschen Ober- bzw. Untersummen beschränkt.

$\Rightarrow$ Konvergenz für $f \in C[a,b]$

Definition 8.4 : Quadraturformeln

Gegeben seien $t_0,...,t_n \in [a,b]$ (=Knoten) und die zugehörigen Gewichte $w_0,...,w_n \in R$ mit $\sum_{i=0}^{n} w_i = 1$.

Dann heisst $\tilde{I}(f) := (b-a) \cdot \sum_{i=0}^{n} w_i f(t_i)$ Quadraturformel zur numerischen Berechnung von $I_a^b(f)$.

Eine Quadraturformel heisst positiv , falls alle $w_i > 0$.

Bem 8.5 : Interpolation und Quadratur

Idee : Approximiere f durch ein Interpolationspolynom zu den Stützstellen $t_0,...,t_n \in [a,b]$.

Und zwar mit den Lagrangeschen Basispolynomen : $f(t) \approx \tilde{f}(t) := \sum_{i=0}^{n} f(t_i) L_i^{(n)}(t)$.

z.B. Trapezregel auf einem Teilintervall = lineare Interpolation

Ergebnis :

$$\tilde{I}(f) := I(\tilde{f}) = \int_a^b \tilde{f}(t)\,dt = \sum_{i=0}^{n}\left(f(t_i) \cdot \int_a^b L_i^{(n)}(t)\,dt \right) = \sum_{i=0}^{n} w_i f(t_i)$$

mit $w_i := \frac{1}{b-a} \cdot \int_a^b L_i^{(n)}(t)\,dt$

Eigenschaften :

- $\tilde{I}(p) = I(p) \qquad ,(p \in \Pi_n)$
- Sind $n+1$ paarweise verschiedene Knoten $t_0,...,t_n \in [a,b]$ gegeben , so bestimmt die Bedingung $\tilde{I}(p) = I(p)$ eindeutig die Gewichte der Quadraturformel $(b-a) \cdot \sum_{i=0}^{n} w_i f(t_i)$, denn durch Einsetzen der Lagrangeschen Basispolynome folgt :

$$\int_a^b L_j^{(n)}(t)\,dt = I\left(L_j^{(n)}\right) \overset{!}{=} \tilde{I}\left(L_j^{(n)}\right) = (b-a) \cdot \sum_{i=0}^{n} w_i L_j^{(n)}(t_i)$$

$$= (b-a) \cdot \sum_{i=0}^{n} w_i d_{i,j} = (b-a) \cdot w_j \qquad ,(j = 0,...,n)$$

Bem. 8.6 : Newton-Cotes-Formeln

Idee : Wähle Stützstellen $t_0,...,t_n$ äquidistant in $[a,b]$.

$\Rightarrow$

Newton-Cotes-Formeln :

$$t_i = a + i \cdot \frac{b-a}{n} \qquad , (i = 0,...,n)$$

$$\tilde{I}(f) = (b-a) \cdot \sum_{i=0}^{n} w_i f(t_i) = (b-a) \cdot \sum_{i=0}^{n} w_i f\left(a + i \cdot \frac{b-a}{n} \right)$$

Newton-Cotes-Gewichte :

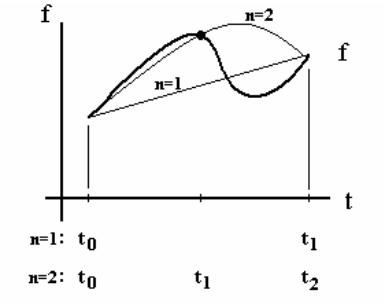
$$w_i = \frac{1}{b-a} \int_a^b \left(\prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t-(a+j \cdot h)}{(a+i \cdot h)-(a+j \cdot h)} \right) dt = \frac{1}{n} \int_0^n \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{s-j}{i-j}\,ds$$

mit der Schrittweite $h := \frac{b-a}{n}$ und $s := \frac{t-a}{h}$. $\qquad \left(ds = \frac{1}{h}dt = \frac{n}{b-a}dt \right)$

Beispiele :

n	w_i	Name	Fehler
1	1/2 , 1/2	Trapezregel	$\frac{h^3}{12} f''(t)$

2	1/6 , 2/3 , 1/6	Keplersche Fassregel	$\frac{h^5}{90} f^{(4)}(t)$
3	1/8 , 3/8 , 3/8 , 1/8	Newtonsche 3/8-Regel	$\frac{3h^5}{80} f^{(4)}(t)$



n=1 : Trapez
n=2 : quadratisches Interpolationspolynom

Jeweils mit einem von f, a und b abhängigen $t \in [a,b]$.

Die Positivität ist garantiert für $n \leq 7$; ab $n = 8$ auch negative Gewichte.

Lemma 8.7 : Mittelwertsatz in Integralform

Sei $g, h \in C[a,b]$.

Hat g auf $[a,b]$ stets ein und dasselbe Vorzeichen , d.h. $g(t) \geq 0$ oder $g(t) \leq 0$ für $t \in [a,b]$,

so gibt es ein $t \in [a,b]$ mit $\int_a^b h(t)g(t)\,dt = h(t) \cdot \int_a^b g(t)\,dt$.

Lemma 8.8 : Approximationsfehler der Trapezregel

Sei $f \in C^2[a,b]$ und $h := b-a$.

Dann gibt es ein $t \in [a,b]$, so dass sich der Approximationsfehler der

Trapezregel $\tilde{I}(f) = T := \frac{b-a}{2}(f(a)+f(b))$ darstellen lässt als : $T - \int_a^b f(t)\,dt = \frac{h^3}{12} f''(t)$.

Lemma 8.9 : Approximationsfehler der Keplerschen Fassregel

Die Keplersche Fassregel $S = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right)$ ist für beliebige

Polynome $p \in \Pi_3$ exakt.

Ist $f \in C^4[a,b]$, so lässt sich der Approximationsfehler darstellen als : $S - \int_a^b f(t)\,dt = \frac{h^5}{90} f^{(4)}(t)$,

mit $h := \frac{b-a}{2}$ und $t \in [a,b]$.

Lemma 8.10 : Approximationsfehler der zusammengesetzten Trapezregel

Zu äquidistanten Stützstellen $t_i = a + i \cdot h \quad , (i = 0,...,n) \qquad , h := \frac{b-a}{n}$

sei $T_i := \frac{h}{2}\left(f(t_i) + f(t_{i+1})\right)$ die Trapezregel auf $[t_i, t_{i+1}]$.

Der Approximationsfehler $T(h) = \sum_{i=0}^{n-1} T_i = \frac{b-a}{n} \cdot \left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+i \cdot h) \right)$

lässt sich für $f \in C^2[a,b]$ darstellen als : $T(h) - \int_a^b f(t)\, dt = \frac{h^2}{12}(b-a) f''(t)$,

mit einem $t \in [a,b]$.

8.2 Gauß-Christoffel-Quadratur

Bem 8.11 : Problemstellung :

Wie sind für vorgegebene $n \in N$ die $n+1$ Knoten $t_0^{(n)}, t_1^{(n)}, ..., t_n^{(n)}$ zu wählen , damit man

Gewichte $w_0^{(n)}, w_1^{(n)}, ..., w_n^{(n)}$ für die Quadraturformel $\tilde{I}_n(f) := \sum_{i=0}^{n} \left(w_i^{(n)} \cdot f\left(t_i^{(n)}\right) \right)$ angeben kann ,

welche alle Polynome $p \in \Pi_N$ mit möglichst großem $N \in N$ exakt integriert $\left(\tilde{I}_n(p) = I(p)\right)$?

Hypothese :

$N = 2n+1$ möglich , da $2n+2$ freie Parameter $t_i^{(n)}, w_i^{(n)}$ $, (i = 0,1,...,n)$

Analytischer Rahmen :

$I(f) = \int_a^b w(t) f(t)\, dt$ gewichtete Integrale

$w(t) > 0$ so , dass die Folge der Momente $m_k := \int_a^b w(t) \cdot t^k \, dt$ existiert und beschränkt bleibt.

(Zulässig ist auch $a = -\infty$, $b = +\infty$)

Beispiele :

$w(t)$	$\frac{1}{\sqrt{1-t^2}}$	e^{-t}	e^{-t^2}	1
Intervall $[a,b]$	$[-1,1]$	$[0,\infty)$	$(-\infty,\infty)$	$[-1,1]$

Lemma 8.12 : Orthogonalität von Knotenpolynomen

Ist für jedes $k \le n$ die Quadraturformel $\tilde{I}_k$ für alle Polynome $p \in \Pi_{2k+1}$ exakt , d.h. $\tilde{I}_k(p) = I(p)$,

so sind die Polynome $\{P_0, P_1, ..., P_n\}$ mit $P_k(t) := \left(t - t_0^{(k)}\right)\left(t - t_1^{(k)}\right)..\left(t - t_k^{(k)}\right) \in \Pi_k$ $, (k = 0,...,n)$

orthogonal bezüglich des von w induzierten Skalarprodukts $< f, g > := \int_a^b w(t) f(t) g(t)\, dt$.

Lemma 8.13 : Eigenschaften von Orthogonalpolynomen

a) Zu einem gegebenen Skalarprodukt $< f, g >:= \int_a^b w(t) f(t) g(t)\, dt$ gibt es genau dann eine Familie $\{P_0, P_1, ..., P_n\}$ von Polynomen $P_k \in \Pi_k$ mit führenden Koeffizienten =1 und $< P_j, P_k >= d_{j,k} \cdot < P_k, P_k >$.

Beweis : Gram-Schmidtsches Orthogonalisierungsverfahren angewendet auf $\{1, t, t^2, ..., t^n\}$ (wobei die Normierung entfällt)

b) Jedes dieser Polynome P_k hat genau k einfache Nullstellen in (a, b).

Lemma 8.14 : Gewichte der Gauß-Quadraturformeln

Seien $t_0^{(n)}, ..., t_n^{(n)}$ die Nullstellen des $(n+1)$-ten Orthogonalpolynoms P_{n+1}.

Dann gilt für jede Quadraturformel $\tilde{I}_n(f) = \sum_{i=0}^{n} w_i^{(n)} f\left(t_i^{(n)}\right)$:

$$w_i^{(n)} = \int_a^b w(t) L_i^{(n)}(t)\, dt \qquad ,(i = 0, ..., n) \quad \Leftrightarrow \tilde{I}_n \text{ ist exakt auf } \Pi_n$$

$$\Leftrightarrow \tilde{I}_n \text{ ist exakt auf } \Pi_{2n+1}$$

Satz 8.15 : Gauß-Christoffel-Quadratur

a) Es existieren eindeutig bestimmte Knoten $t_0^{(n)}, ..., t_n^{(n)}$ und Gewichte $w_0^{(n)}, ..., w_n^{(n)}$, so dass die Quadraturformel $\tilde{I}_n(f) = \sum_{i=0}^{n} w_i^{(n)} f\left(t_i^{(n)}\right)$ Polynome bis zum Grad $2n+1$ exakt integriert , d.h.

$$\tilde{I}_n(p) = \int_a^b w(t) p(t)\, dt \qquad , p \in \Pi_{2n+1}$$ - d.h. die obige Hypothese gilt

b) Die Knoten $t_0^{(n)}, ..., t_n^{(n)}$ sind die Nullstellen des $(n+1)$-ten Orthogonalpolynoms P_{n+1} bzgl. der Gewichtsfunktion $w(t)$ und die Gewichte ergeben sich aus $w_i^{(n)} = \int_a^b w(t) L_i^{(n)}(t)\, dt$ mit den Lagrangeschen Basispolynomen $L_i^{(n)}(t) \qquad ,(i = 0,1,...,n)$.

c) Die Gewichte $w_i^{(n)}$ sind positiv , d.h. die Quadraturformel $\tilde{I}_n$ ist positiv.

d) Es gilt : $w_i^{(n)} = \dfrac{< P_n, P_n >}{P_{n+1}'\left(t_i^{(n)}\right) \cdot P_n\left(t_i^{(n)}\right)}$

Def. 8.16 : Gauß-Christoffel-Quadratur

Die nach Satz 8.15 eindeutig bestimmten positiven Quadraturformeln $\tilde{I}_n$ heißen Gauß-Christoffel-Formeln

zum Gewicht w.

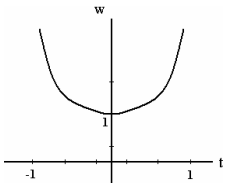
Satz 8.17 : Approximationsfehler der GCQ

Für jedes $f \in C^{2n+2}[a,b]$ lässt sich der Approximationsfehler der GCQ ausdrücken durch :

$$\int_a^b w(t)\, f(t)\, dt - \tilde{I}_n(f) = \frac{f^{(2n+2)}(t)}{(2n+2)!} \cdot < P_{n+1}, P_{n+1} > \qquad \text{für ein } t \in [a,b].$$

Beispiel 8.18 : Gauß-Tschebyscheff-Quadratur

$$w(t) = \frac{1}{\sqrt{1-t^2}} \quad , \quad t \in (-1,1)$$



Die in Bemerkung 5.12.c) eingeführten Tschebscheff-Polynome $T_k = \cos k \cdot \arccos t \quad , t \in [-1,1]$

sind orthogonal bzgl. $< f, g > := \int_{-1}^{1} \frac{f(t)g(t)}{\sqrt{1-t^2}}\, dt$, denn

$$\int_{-1}^{1} \frac{T_k(t) \cdot T_j(t)}{\sqrt{1-t^2}}\, dt = \int_p^0 \frac{-\sin x \cdot \cos(kx) \cdot \cos(jx)}{\sin x}\, dx \qquad (t = \cos x \;\Rightarrow\; dt = -\sin x\, dx)$$

$$= \int_0^p \cos(kx) \cdot \cos(jx)\, dx = \begin{cases} p & , \text{falls } k = j = 0 \\ p/2 & , \text{falls } k = j > 0 \\ 0 & \text{sonst} \end{cases}$$

Die Tschebyscheff-Knoten $t_i^{(n)}$ ergeben sich als Nullstellen der Orthogonalpolynome

$$P_{n+1}(t) := \frac{1}{2^n} T_{n+1}(t) \quad \text{zu } t_i^{(n)} = \cos\left(\frac{2i+1}{2n+2} p\right) \quad , (i = 0,1,...,n) \qquad \textbf{(*)}$$

Gewichte für $n > 0$ nach Satz 8.15.d) :

$$w_i^{(n)} = \frac{1}{\frac{1}{2^n} T_{n+1}'\left(t_i^{(n)}\right) \cdot \frac{1}{2^{n-1}} T_n\left(t_i^{(n)}\right)} \cdot \int_{-1}^{1} \frac{\left(\frac{1}{2^{n-1}} T_n(t)\right)^2}{\sqrt{1-t^2}}\, dt$$

$$= \frac{2\frac{p}{2}}{T_{n+1}'\left(t_i^{(n)}\right) \cdot T_n\left(t_i^{(n)}\right)}$$

Mit $T_{n+1}'(t) \cdot T_n(t) = -\sin(n+1) \cdot \arccos t \cdot (n+1) \cdot \frac{-1}{\sqrt{1-t^2}} \cdot \cos n \cdot \arccos t$ folgt :

$$= (n+1) \frac{\sin((n+1)x) \cdot \cos(nx)}{\sin x} \qquad , (t = \cos x)$$

$$= (n+1)\frac{\sin((2n+1)x) - \cos((n+1)x)\cdot\sin(nx)}{\sin x}$$
$$= (n+1)\frac{\sin((2n+2)x)\cdot\cos x - \cos((2n+2)x)\cdot\sin x - \cos((n+1)x)\cdot\sin(nx)}{\sin x}$$

Einsetzen von $x = \frac{2i+1}{2n+2}p$ ergibt :

- $\sin((2n+2)x) = 0$
- $\cos((n+1)x) = \cos\left(\frac{2i+1}{2}p\right) = 0$
- $\cos((2n+2)x) = \cos((2i+1)p) = -1$

$$\Rightarrow w_i^{(n)} = \frac{p}{n+1}$$

Ergebnis der Gauß-Tschebyscheff-Quadratur :

$$\tilde{I}_n(f) = \sum_{i=0}^{n} w_i^{(n)} f\left(t_i^{(n)}\right) = \frac{p}{n+1}\cdot\sum_{i=0}^{n} f\left(\cos\left(\frac{2i+1}{2n+2}p\right)\right) \approx \int_{-1}^{1}\frac{f(t)}{\sqrt{1-t^2}}dt$$

Approximationsfehler :

$$\int_{-1}^{1}\frac{f(t)}{\sqrt{1-t^2}}dt - \tilde{I}_n(f) = \frac{p}{2^{2n+1}\cdot(2n+2)!}\cdot f^{(2n+2)}(t) \qquad \text{für ein } t \in [-1,1].$$

Desweiteren :

$$< P_{n+1}, P_{n+1} > = \frac{p}{2}\cdot\left(\frac{1}{2^n}\right)^2$$

Bem. 8.19 : Typische Gauß-Christoffel-Quadraturformeln

$w(t)$	Intervall $[a,b]$	orthogonales Polynomsystem
$\frac{1}{\sqrt{1-t^2}}$	$[-1,1]$	Tschebyscheff-Polynome T_n
e^{-t}	$[0,\infty)$	Laguerre-Polynome L_n
e^{-t^2}	$(-\infty,\infty)$	Hermite-Polynome H_n
1	$[-1,1]$	Legendre-Polynome P_n

Vorteile :

Sehr hohe Approximationsordnung für sehr glatte Integranden schon bei geringer Knotenzahl , auch für singuläre Integranden und/oder unendliche Integrationsintervalle.

Nachteile :

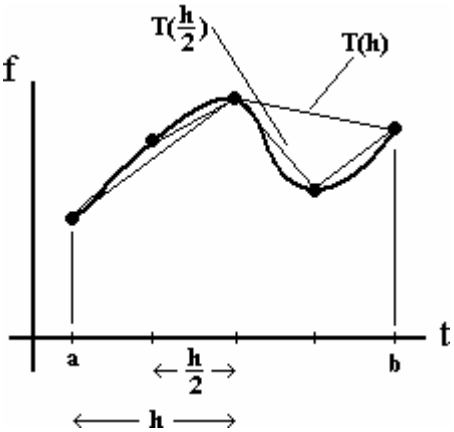
- Im Allgemeinen aufwendige Berechnung der Knoten und Gewichte durch Berechnung der orthogonalen Polynome oder anschliessende Nullstellen-Bestimmung.
- Erhöht man zur Verringerung des Approximationsfehlers die Zahl der Knoten , so sind i.A. sämtliche Funktionswerte $f\left(t_i^{(n)}\right)$ neu zu berechnen.

8.3 Extrapolationsverfahren , Romberg-Quadratur

Beispiel 8.20 : Simpsonregel

Berechnung zweier Näherungswerte $T(h)$ und $T\left(\frac{h}{2}\right)$ mit der zusammengesetzten Trapezregel.

(vgl. Bem. 8.3 und Lemma 8.10)



$$T(h) = \frac{h}{2} f(a) + h \cdot f(a+h) + \ldots + \frac{h}{2} f(b)$$

$$T\left(\frac{h}{2}\right) = \frac{h}{4} f(a) + \frac{h}{2} f\left(a+\frac{h}{2}\right) + \frac{h}{2} f(a+h) + \ldots + \frac{h}{4} f(b)$$

$$T(h) = I(f) + \frac{h^2}{12}(b-a) \cdot f''(t_h)$$

$$T\left(\frac{h}{2}\right) = I(f) + \frac{h^2}{48}(b-a) \cdot f''\left(t_{h/2}\right)$$

Idee :

Ist $f''(t_h) \approx f''\left(t_{h/2}\right)$, so ergibt sich eine sehr viel genauere Näherung mit der Simpsonregel :

$$S\left(\frac{h}{2}\right) := T\left(\frac{h}{2}\right) + \frac{1}{3}\left(T\left(\frac{h}{2}\right) - T(h)\right)$$

Es gilt :

$$S\left(\frac{h}{2}\right) = \frac{h}{6} f(a) + \frac{2h}{3} f\left(a+\frac{h}{2}\right) + \frac{h}{3} f(a+h) + \ldots + \frac{h}{6} f(b)$$

$$= \sum_{i=0}^{n-1} S_i \qquad \text{mit} \quad S_i := \frac{h}{6}\left(f(a+ih) + 4 \cdot f\left(a + \left(i+\frac{1}{2}\right)h\right) + f(a+(i+1)h)\right)$$

$$S_i \approx \int_{a+ih}^{a+(i+1)h} f(t)\,dt \qquad \text{Keplersche Fassregel}$$

$$S(h) \approx \int_a^b f(t)\,dt \qquad \text{Simpsonregel}$$

Analog zu Lemma 8.10 zeigt man $S(h) - \int_a^b f(t)\,dt = \frac{h^4}{180}(b-a)\cdot f^{(4)}(t_S)$ mit einem $t_S \in (a,b)$.

$\Rightarrow$ Fehler i.A. viel kleiner als bei $T\left(\frac{h}{2}\right)$, keine zusätzlichen Auswertungen von f erforderlich.

$S\left(\frac{h}{2}\right)$ ergibt sich durch Auswertung des linearen Interpolationspolynoms $p(s)$

durch $(h, T(h))$ und $\left(\frac{h}{2}, T\left(\frac{h}{2}\right)\right)$ für $s = 0$. (Extrapolationsverfahren)

Satz 8.21 : Euler-MacLaurinsche Summenformel

Ist $g \in C^{2m+2}[0,1]$, so gilt :

$$\int_0^1 g(t)\,dt = \frac{g(0)+g(1)}{2} + \sum_{k=1}^{m} \frac{B_{2k}}{(2k)!}\cdot\left(g^{(2k-1)}(0) - g^{(2k-1)}(1)\right) - \frac{B_{2m+2}}{(2m+2)!}\cdot g^{(2m+2)}(t)$$

mit einem $t \in (0,1)$.

Hierbei bezeichnet B_k die Bernoullizahlen $B_k(0)$ mit

$B_1(x) := x - \frac{1}{2}$ und $B_{k+1}'(x) = (k+1)\cdot B_k(x)$, $(k \geq 1)$

z.B. $B_2 = \frac{1}{6}$, $B_4 = \frac{1}{30}$, $B_6 = \frac{1}{42}$, $B_8 = -\frac{1}{30}$, $\left(B_{2k} \sim (2k)!\ \text{für}\ k \to \infty\right)$

Folgerung 8.22 : h^2-Entwicklung des Approximationsfehler der zusammengesetzten Trapezregel

Unterteilt man $[a,b]$ äquidistant in N Teilintervalle der Länge $h := \frac{b-a}{N}$, so ergibt die wiederholte Anwendung von Satz 8.21

$$T(h) = \int_a^b f(t)\,dt + \sum_{k=1}^{m} \underbrace{h^{2k}\cdot\frac{B_{2k}}{(2k)!}}_{\text{Fehler siehe } \frac{h^2}{12}\text{, oben } \frac{B_2}{2}} \cdot\left(f^{(2k-1)}(b) - f^{(2k-1)}(a)\right) + h^{2m+2}\cdot\frac{B_{2m+2}}{(2m+2)!}\cdot f^{(2m+2)}(t)$$

mit einem $t \in (a,b)$

Beweis : Ausgehend von fortlaufender partieller Integration : $\int_0^1 g(t)\,dt = B_1(t)\cdot g(t)\Big|_0^1 - \int_0^1 B_1(t)\cdot g'(t)\,dt$ ... usw.

Bem. 8.23 : Asymptotische Entwicklung

Ein Näherungsverfahren $T(h)$ zur Berechnung einer Größe t_0 besitzt eine asymptotische Entwicklung in h^p bis zur Ordnung $p \cdot m$, falls es Konstanten $C_p, C_{2p}, C_{3p}, ..., C_{mp}$ gibt , so dass gilt :

$$T(h) = t_0 + C_p h^p + C_{2p} h^{2p} + ... + C_{mp} h^{mp} + O\left(h^{(m+1)p}\right) \quad , (h \to 0)$$

Die zusammengesetzte Trapezregel hat für Funktionen $f \in C^{2m+2}[a,b]$ eine asymptotische Entwicklung in h^2 bis zur Ordnung $2m$. (vgl. Folgerung 8.22)

Bem. 8.24 : Extrapolationsverfahren

Ansatz : Auswertung von $T(h)$ für k verschiedene Schrittweiten $h = h_{i-k+1}$, $h = h_{i-k+2}$, ... , $h = h_i$ und Bestimmung des Interpolationspolynoms $P_{ik} \in \Pi_{k-1}$ zu den k Stützpunkten $\left(\left(h_{i-k+1}{}^p, T(h_{i-k+1}{}^p)\right), ..., \left(h_i{}^p, T(h_i{}^p)\right)\right)$

Näherungswert : $I(f) \approx T_{ik} := P_{ik}(0) \quad , (k = 1, ..., i)$

Auswertung von $P_{ik}(0)$ mittels Neville-Schema (vgl. Bem. 7.5) :

$$T_{i1} := T(h_i)$$

$$T_{ik} := T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{h_{i-k+1}}{h_i}\right)^p - 1} \quad , (k = 2, ..., i)$$

Extrapolationstabelle :

$$\begin{array}{ccccc} T_{11} & & & & \\ T_{21} & \to & T_{22} & & \\ \vdots & & \to & & \\ \vdots & & & \ddots & \\ T_{k1} & \to & T_{k2} & \to & T_{kk} \end{array}$$

Satz 8.25 : Approximationsfehler des Extrapolationsverfahrens

Sei $T(h)$ ein Verfahren mit einer asymptotischen Entwicklung in h^p bis zur Ordnung $m \cdot p$.

Für die Fehler der Extrapolationswerte T_{ik} zu den paarweise voneinander verschiedenen Schrittweiten $h_1, ..., h_m$ gilt :

$$\left|T_{ik} - t_0\right| = \left|C_{kp}\right| \cdot h_{i-k+1}{}^p \cdot ... \cdot h_{i-1}{}^p \cdot h_i{}^p + \sum_{j=i-k+1}^{i} O\left(h_j{}^{(k+1)p}\right) \quad , (k = 1, ..., i \le m \quad , \quad h_j \le h \to 0)$$

Lemma 8.26 : Eigenschaften der Lagrangeschen Basispolynome

Für die Lagrangeschen Basispolynome $L_j^{(n)}$, $(j = 0,1,...,n)$ zu den paarweise voneinander verschiedenen Stützstellen $t_0, t_1, ..., t_n$ gilt : $\sum_{j=0}^{n} L_j^{(n)}(0) \cdot t_j^{\,r}) = \begin{cases} 1 & \text{für } r = 0 \\ 0 & \text{für } r = 1,...,n \\ (-1)^n t_0 t_1 ... t_n & \text{für } r = n+1 \end{cases}$.

Beweis :

$P(t) = t^r$ ist Interpolationspolynom zu den Stützpunkten $(t_0, t_0^{\,r}), (t_1, t_1^{\,r}), ..., (t_n, t_n^{\,r})$ für $r = 0,1,...,n$.

$\Rightarrow P(t) = t^r = \sum_{j=0}^{n} \left(L_j^{(n)}(t) \cdot P(t_j) \right) = \sum_{j=0}^{n} \left(L_j^{(n)}(t) \cdot t_j^{\,r} \right)$ und damit die Behauptung für $r \le n$ durch Einsetzen von $t = 0$.

Zu $r = n+1$ betrachtet man $q(t) := t^{n+1} - \sum_{j=0}^{n} \left(L_j^{(n)}(t) \cdot t_j^{\,n+1} \right) \in \Pi_{n+1}$

Das Polynom q hat den führenden Koeffizienten 1 und Nullstellen $t_0, t_1, ..., t_n$

$\Rightarrow q(t) = (t - t_0)(t - t_1)...(t - t_n)$ und für Einsetzen von $t = 0$ gilt :

$\sum_{j=0}^{n} \left(L_j^{(n)}(0) \cdot t_j^{\,n+1} \right) = -q(0) = (-1)^n t_0 ... t_n$

q.e.d.

Algorithmus 8.27 : Extrapolationsverfahren

geg.: Grundschrittweite H

Schritt 1: wähle Schrittweiten $h_1, h_2, ...$ mit $h_j = \frac{H}{n_j}$, $(n_{j+1} > n_j)$ und setze $i := 1$

Schritt 2: Berechne $T_{i1} = T(h_i)$

Schritt 3: Berechne $T_{i2}, ..., T_{ii}$ mit dem Schema von Neville :

$$T_{ik} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{n_i}{n_{i-k+1}} \right)^p - 1}$$

Schritt 4: Abbruch , falls T_{ii} ausreichend genau oder $i \ge i_{\max}$

Sonst i:=i+1 und gehe zu Schritt 2

Die Anwendung dieses Algorithmus auf die zusammengesetzte Trapezregel heißt **Romberg-Quadratur**.

Wie sind die Schrittweiten und Abbruchkriterien zu wählen ?

Bem. 8.28 : Romberg-Quadratur

Aufwand : A_k ... Anzahl der zur Berechnung von T_{kk} benötigten Funktionsauswertungen

Aufwandsfolge $\tilde{A} = \{A_1, A_2, ...\}$ zur Folge $F = \{n_1, n_2, ...\}$ mit $0 < n_1 < n_2 < ...$

Rombergfolge :

$F_R = \{1,2,4,8,16,...\} \quad , (n_i = 2^{i-1})$

$\tilde{A}_R = \{2,3,5,9,17,...\} \quad , (A_i = n_i + 1)$

$$T(h) = \underbrace{\frac{h}{2}\left(f(a) + 2\sum_{j=1}^{n-1} f(a+2jh) + f(b) \right)}_{\text{Summanden zu den geradzahligen Indizes}} + \underbrace{h \cdot \sum_{j=1}^{n} f(a+(2j-1)h)}_{\text{Summanden zu den ungeraden Indizes}}$$

$$= \frac{1}{2} T \underbrace{(2h)}_{\text{neu zu berechnen}} + \underbrace{h \cdot \sum_{j=1}^{n} f(a+(2j-1)h)}_{\text{bleibt wiezuvor}} \qquad \text{mit } h := \frac{H}{2n}$$

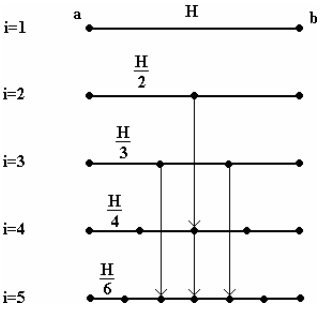
$$T_{i+1,1} = \frac{1}{2} T_{i1} + h_{i+1} \cdot \sum_{j=1}^{n_i} f(a + (2j-1) \cdot h_{i+1}) \quad \text{mit } h_i := \frac{H}{n_i} \text{ und } n_i = 2^{i-1}$$

Es gilt : $T_{kk} = H \cdot \sum_{j=1}^{A_k} l_j f_j$ mit positiven Gewichten l_j

Bulirschfolge :

$F_B = \{1,2,3,4,6,8,12,16,24,...\}$ Doppeltes des Vorvorgängers bzw. :

$$n_i = \begin{cases} 1 & \text{falls } i = 1 \\ 2^k & \text{falls } i = 2k \\ 3 \cdot 2^{k-1} & \text{falls } i = 2k+1 > 1 \end{cases}$$



$\tilde{A}_B = \{2,3,5,7,9,13,...\}$

Der Aufwand wächst langsamer als für die Rombergfolge , jedoch treten für $k \geq 3$ negative Gewichte l_j in T_{kk} auf.

Harmonische Folge :

$F_H = \{1,2,3,4,5,6,...\} \quad , (n_i = i)$

$A_H = \{2,3,5,7,11,13,19,...\}$

Zur Quadratur ungeeignet , da bei der Berechnung von T_{i1} kaum auf bereits zuvor berechnete

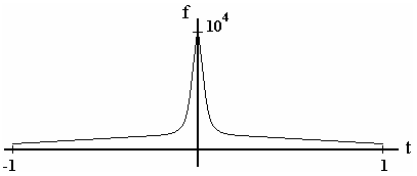
Funktionswerte von f zurückgegriffen werden kann.

Jedoch sehr gut geeignet für Probleme , bei denen A_k weitgehend unabhängig von $n_1, n_2, ..., n_{k-1}$ ist.

(Beispiel : Extrapolationsverfahren für Anfangswertprobleme gewöhnlicher Differentialgleichungen)

Beispiel. 8.29 : Nadelimpuls

$$I(f) = \int_{-1}^{1} \frac{1}{10^{-4} + t^2} dt$$



Rombergfolge :

k	A_k	T_{kk}	$e_{kk} := \frac{\lvert T_{kk} - I(f)\rvert}{\lvert I(f)\rvert}$ (relativer Fehler)
1	2	1.998	0.99 (99%)
2	3	13334.0	42.0 (4200%)
3	5	2627.7	7.6
4	9	1551.9	4.0
...	...	...	...
8	129	293.01	0.061
...	...	...	...
13	4097	312.16	$1.1 \cdot 10^{-9}$

Dieselbe Fehlergenauigkeit erhält man mit TRAPEX (extrapolierende Trapezregel – siehe Deuflhard) mit 321 Funktions-Auswertungen.

Problem : Hoher Rechenaufwand der klassischen Romberg-Quadratur

Ausweg :

i) Zerlege [-1,1] in Teilintervalle $[a_i, a_{i+1}]$ mit $a = a_0 < a_1 < ... < a_N = b$

$$\Rightarrow I(f) = \sum_{j=1}^{N-1} I_{a_i}^{a_{i+1}}(f)$$

ii) Approximiere $I_{a_i}^{a_{i+1}}(f)$ mittels Romberg-Quadratur

$T_{kk} \rightarrow T_{k(i),k(i)}$

unterschiedliche Tiefe der Extrapolationsschemata in den einzelnen Teilintervallen

Adaptiver Algorithmus :

Bestimme a_i und $k(i)$ während der Berechnung automatisch so , dass bei möglichst niedrigem Gesamtrechenaufwand der Fehler unterhalb einer vorgesehenen Genauigkeitsschranke ***TOL*** bleibt.

Grundprinzip :

Gleichverteilung des Fehlers , d.h. Bestimmung der a_i so , dass jedes der N Teilintervalle einen annähernd gleich großen Beitrag zum Gesamtfehler leistet.

Fehlerschätzer :

Benötigt wird ein möglichst guter Schätzwert für $|T_{kk} - I(f)|$ bzw. für $\left| T_{k(i),k(i)} - \int_{a_i}^{a_{i+1}} f(t)\, dt \right|$

(, d.h. für jedes Intervall.)

ideal :

Obere und untere Schranke für den tatsächlichen Fehler , zumindest asymptotisch für $\frac{H}{n_k} \to 0$.

Aus Satz 8.25 folgt :

$$|T_{kk} - I(f)| = |C_{2k}| \cdot h_1^{\,2} \cdot \ldots \cdot h_{k-1}^{\,2} h_k^{\,2} + \sum_{j=1}^{k} O\left(h_j^{\,2(k+1)}\right)$$

$$|T_{k,k-1} - I(f)| = |C_{2(k-1)}| \cdot h_2^{\,2} \cdot \ldots \cdot h_{k-1}^{\,2} h_k^{\,2} + \sum_{j=2}^{k} O\left(h_j^{\,2(k+1)}\right)$$

(Der zweite Summand wird verschwindend klein und wird nicht betrachtet.)

Für ausreichend kleine Schranken $(h_1 << 1)$ und Konstanten $|C_{2k}| \approx |C_{2(k-1)}|$ gilt :

$|T_{kk} - I(f)| << |T_{k,k-1} - I(f)|$, also ist wegen

$$|T_{k,k-1} - I(f)| - |T_{kk} - I(f)| \le |T_{k,k-1} - T_{kk}| \le |T_{k,k-1} - I(f)| + |T_{kk} - I(f)|$$

$\Rightarrow$

$|T_{k,k-1} - T_{kk}|$ ist ein guter Schätzwert für $T_{k,k-1}$.

Beachte :

Man schätzt unter Verwendung von $T_{k,k-1}$ und T_{kk} den Fehler von $T_{k,k-1}$, verwendet jedoch anschließend den im Allgemeinen genaueren Näherungswert T_{kk} für $I(f)$.

Algorithmus 8.27 Schritt 4 lautet somit : Abbruch, falls $|T_{i,i-1} - T_{ii}| \le TOL$ oder $i \ge i_{\max}$

9. Numerische Lösung nichtlinearer Gleichungssysteme

geg. : $F : D \to R^n$ mit einer offenen Menge $D \subset R^n$

ges. : $\{x : F(x) = 0\}$

Spezialfälle :

a) F ist linear

$F(x) = Ax - b$

$\Rightarrow$ Lösung von $Ax - b = 0$ (vgl. Kapitel 2 „Lösen von linearen Gleichungssystemen“)

b) $n = 1$

Nullstellen einer nichtlinearen reellen Funktion

Qualitativer Unterschied zu linearen Gleichungssystemen :

Eindeutigkeit der Lösung von $F(x) = 0$ kann im Allgemeinen nur lokal in einer Umgebung einer Nullstelle x^* von F garantiert werden , z.B. mit dem Satz über die Implizite Funktion.

Hinreichende Bedingung :

$F = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$ ist stetig differenzierbar und die Jacobimatrix $F_x = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$ ist regulär.

9.1 Nichtlineare Gleichungssysteme und Fixpunktiteration

Fixpunktgleichungen $\Phi(x) = x$ sind spezielle nichtlineare Gleichungssysteme $F(x) = 0$ mit $F(x) := \Phi(x) - x$.

Idee :

Iterative Bestimmung des Fixpunktes mittels Fixpunktiteration

$$x_{k+1} := \Phi(x_k) \qquad ,(k \geq 0)$$

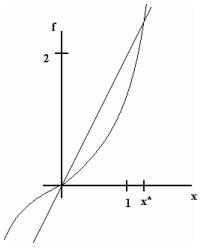
für einen geeigneten Startwert $x_0 \in R^n$.

(Wie ist die Funktion Φ und der Startwert x_0 zu bestimmen ?)

Umgekehrt lässt sich jedes nichtlineare Gleichungssystem $F(x) = 0$ als Fixpunktgleichung $\Phi(x) = x$ schreiben.
z.B. mit $\Phi(x) := x - A^{-1} \cdot F(x)$ mit einer beliebigen regulären Matrix $A \in R^{n \times n}$.

$$F(x) = 0 \quad \Leftrightarrow \quad \Phi(x) = x$$

Beispiel 9.3 : Fixpunktiteration zur Lösung nichtlinearer Gleichungen



Gesucht : $x^* \in \left(0, \frac{p}{2}\right)$ mit $f(x^*) = 0$.

Naheliegende Fixpunktgleichungen :

a) $x = \frac{1}{2}\tan x =: j_1(x)$

b) $x = \arctan(2x) =: j_2(x)$

k	$x_{k+1} = j_1(x_k)$	$x_{k+1} = j_2(x_k)$
0	1.2	1.2
1	1.2860	1.1760
2	1.70 > $p/2$	1.1687
...	divergent	...
6		1.1655
7		1.1655

Definition 9.4 : Kontrahierende Abbildung

Sei $D \subset R^n$ eine offene Menge und $\Phi : \overline{D} \to R^n$. ($\overline{D}$ = Abschluss von D)

Die Abbildung Φ heißt kontrahierend auf $\overline{D}$, wenn es eine Konstante $a \in [0,1)$ gibt mit

$$\|\Phi(x) - \Phi(y)\| \le a \cdot \|x - y\| \quad , (x, y \in \overline{D})$$

Lemma 9.5 : Stetig differenzierbare kontrahierende Abbildung

Sei $D \subset R^n$ eine konvexe , offene Menge und $\Phi : \overline{D} \to R^n$ stetig differenzierbar.

Existiert $a := \sup_{x \in \overline{D}} \|\Phi_x(x)\|$ und ist $a < 1$, so ist Φ kontrahierend auf $\overline{D}$.

Satz 9.6 : Banachscher Fixpunktsatz

Sei $E \subset R^n$ kompakt und $\Phi : E \to E$ kontrahierend mit einer Lipschitz-Konstanten $a < 1$.
Dann gilt :

a) Φ hat genau einen Fixpunkt x^* in E , d.h. $\Phi(x^*) = x^*$.

b) Für jeden Startwert $x_0 \in E$ konvergiert die Fixpunktiteration $x_{k+1} = \Phi(x_k)$ gegen x^* und es gilt :

i) $\|x^* - x_k\| \le \frac{a^k}{1-a} \cdot \|x_1 - x_0\|$ (a-priori-Fehlerschranke)

ii) $\|x^* - x_{k+1}\| \le \frac{a}{1-a} \cdot \|x_{k+1} - x_k\|$ (a-posteriori-Fehlerschranke)

Bem. 9.7 : praktische Aspekte der Fixpunktiteration

a) Bestimmung einer Näherung für a

a-priori : Für stetig differenzierbares Φ eine obere Schranke für $\|\Phi_x\|$ berechnen.

a-posteriori : $x_{k+2} - x_{k+1} = \Phi(x_{k+1}) - \Phi(x_k)$

$$\Rightarrow a \approx a_k \quad \text{mit} \quad a_k := \frac{\|x_{k+2} - x_{k+1}\|}{\|x_{k+1} - x_k\|}$$

b) Abbruch

Gegeben seien Fehlerschranken $ATOL$ und $RTOL$ für den absoluten und relativen Fehler.
Ein Schätzwert für a sei bekannt.

i) Anzahl der Iterationsschritte vorab festlegen :

$$x^* \approx x_k \qquad \text{mit} \qquad a^k \le \frac{(1-a)\cdot\left(ATOL + RTOL\cdot\|x_0\|\right)}{\|x_1 - x_0\|}$$

$$\Rightarrow \qquad k := 1 + \frac{\log\left(\dfrac{(1-a)\cdot\left(ATOL + RTOL\cdot\|x_0\|\right)}{\|x_1 - x_0\|}\right)}{\log a}$$

ii) Abbruchkriterium während der Iteration überprüfen :

$x^* \approx x_{k+1}$, falls $\dfrac{a_k}{1-a_k}\cdot\|x_{k+1} - x_k\| \le ATOL + RTOL\cdot\|x_k\|$ mit a_k wie oben

Ist für ein $k_0 > 0$ $a_{k_0} > 1$, so deutet dies auf Divergenz von (x_k)

$\Rightarrow$ (ergebnisloser) Abbruch des Verfahrens

c) <u>Konvergenzbeschleunigung</u>

Mittels Δ^2 - Methode von Aitken oder Methode von Steffensen

<u>Definition 9.8 : Konvergenz , Konvergenzordnung</u>

a) Ein Iterationsverfahren $x_{k+1} = \Psi_k(x_0, x_1, ..., x_k)$ heißt konvergent , wenn die Folge der Iterierten (x_k) konvergiert.
Anderenfalls heißt sie divergent.

b) Das Verfahren hat die Konvergenzordnung $p \ge 1$, wenn $x^* := \lim_{k\to\infty} x_k$ existiert und eine Konstante $C \ge 0$ ($p = 1:\ C = a \in [0,1)$) angegeben werden kann , so dass
$\|x_{k+1} - x^*\| \le C\cdot\|x_k - x^*\|^p$, $(k \ge k_0)$.

Das Verfahren konvergiert linear , falls $p = 1$, quadratisch , falls $p = 2$
und superlinear , falls gilt : $\|x_{k+1} - x^*\| \le C_k\cdot\|x_k - x^*\|$ mit $C_k \ge 0$ und $\lim_{k\to\infty} C_k = 0$.

<u>9.2 Newton-Verfahren</u>

geg.: $F : R^n \to R^n$ stetig differenzierbar
ges.: x^* mit $F(x^*) = 0$

<u>Ansatz :</u>

Linearisierung von F in einer Umgebung von x_k :

$F_k(x) := F(x_k) + F_x(x_k)\cdot(x - x_k)$

$F_k(x) \approx F(x)$ in einer (kleinen) Umgebung von x_k

Bestimme x_{k+1} als Nullstelle von F_k : $x_{k+1} := x_k + p_k$ mit $F_x(x_k)\cdot p_k = -F(x_k)$

<u>Aufwand pro Iterationsschritt :</u>

1 F - Auswertung

1 F_x - Auswertung
1 LGS lösen

Fixpunktdarstellung (formal) :

$$x_{k+1} = \Phi(x_k) \quad \text{mit} \quad \Phi(x) := x - \left(F_x(x)\right)^{-1} \cdot F(x)$$

Affine Invarianz :

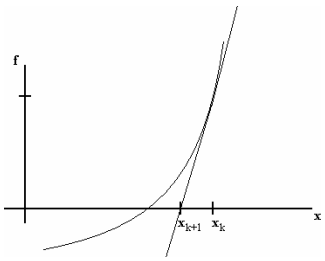
Für jede reguläre Matrix $A \in R^{n \times n}$ ist $F(x) = 0 \quad \Leftrightarrow \quad A \cdot F(x) = 0$

$\Rightarrow \quad \|F(x)\|$ ist als Abbruchkriterium ungeeignet , da z.B. mit 10^{-20} skaliert werden kann.

Das Newton-Verfahren ergibt (in exakter Arithmetik) für beide Gleichungssysteme dieselbe Folge (x_k) , es ist **affin-invariant** :

$$\begin{aligned}\Phi_A(x) &= x - \left(\frac{\partial}{\partial x}\left(A \cdot F(x)\right)\right)^{-1} \cdot A \cdot F(x) \\ &= x - \left(\frac{\partial}{\partial x} F(x)\right)^{-1} \cdot F(x) \\ &= \Phi(x)\end{aligned}$$

Geometrische Interpretation für n=1 :



x_{k+1} ist Schnittpunkt der im Punkt $(x_k, f(x_k))$ an den Graphen von f gelegten Tangente mit der Abszissenachse.

Beispiel 9.10 : Verfahren von Heron

Newton-Verfahren für $0 = f(x) := x^2 - a$ - d.h. Berechnung von $\sqrt{a}$

$$x_{k+1} = x_k - \frac{x_k^{\,2} - a}{2x_k} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$$

praktisch :

binäre Gleitpunktdarstellung $a = m \cdot 2^p$ mit $0.5 < m \leq 1$

$$\sqrt{a} = \begin{cases} 2^r \cdot \sqrt{m} & \text{, falls } \quad p = 2r \\ 2^r \cdot \sqrt{m} \cdot \sqrt{1/2} & \text{, falls } \quad p = 2r-1 \end{cases}$$

Begründung : Sehr schnelle Konvergenz des Newton-Verfahrens für $m \in (0.5, 1]$.

Beispiel :

$m = 0.81 \quad , \quad x_0 = 1$

k	x_k
0	1.0000000000
1	0.9050000000
2	0.9000138122
3	0.9000000001

Satz 9.11 : Konvergenz des Newton-Verfahrens

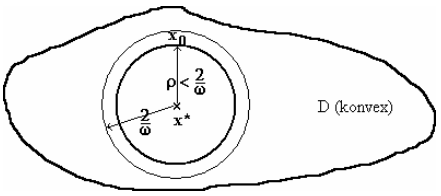
Sei $D \subset R^n$ offen und konvex , und $F : D \to R^n$ stetig differenzierbar mit regulärer Jacobimatrix $F_x(x)$ für alle $x \in D$.

Für F_x gelte die Lipschitz-Bedingung $\left\| (F_x(x))^{-1} \cdot (F_x(x + s \cdot v) - F_x(x)) \cdot v \right\| \le s \cdot w \cdot \|v\|^2$

mit $s \in [0,1] \quad , \quad x \in D \quad , \quad v \in R^n$ mit $x + v \in D$, Konstante $w > 0$.

Hat $F(x) = 0$ eine Lösung $x^* \in D$ und gilt für den Startwert $x_0 \in D$: $r := \|x^* - x_0\| < \frac{2}{w}$,

und ist $B_r(x^*) := \{x \in R^n : \|x - x^*\| < r\} \subset D$, dann bleibt die durch das Newton-Verfahren definierte Folge (x_k) für $k > 0$ in $B_r(x^*)$ und konvergiert gegen x^* : $\lim_{k \to \infty} x_k = x^*$.



Es gilt $\left\| x_{k+1} - x^* \right\| \le \frac{w}{2} \cdot \left\| x_k - x^* \right\|^2 \qquad , (k \ge 0)$

(D.h. in etwa Verdopplung der genauen Stellen pro Schritt.)

Darüber hinaus ist x^* einzige Lösung von $F(x) = 0$ in $B_{\frac{2}{w}}(x^*)$.

Beweis :

Linearisiert man F um einen Punkt $x \in D$, so folgt aus dem Hauptsatz der Differential- und Integralrechnung folgende Abschätzung des Restgliedes für beliebige $y \in D$:

$$F(y) - (F(x) + F_x(x) \cdot (y - x))$$

Führe Funktion Ψ ein mit : $\Psi(s) := F(x + s(y - x))$

$(F(y) = \Psi(1) \, , \, F(x) = \Psi(0))$

$$= \int_0^1 \Psi'(s)\, ds$$

$$= \int_0^1 \left(F_x(x+s(y-x)) - F_x(x)\right)\cdot(y-x)\, ds$$

$$\Rightarrow \left\| \left(F_x(x)\right)^{-1} \cdot \left(F(y) - F(x) - F_x(x)\cdot(y-x)\right) \right\| \le \int_0^1 s\cdot w \cdot \|y-x\|^2 ds \quad = \frac{w}{2}\cdot\|y-x\|^2$$

$$\Rightarrow x_{k+1} - x^* = x_k - \left(F_x(x_k)\right)^{-1}\cdot F(x_k) - x^*$$

$$= \left(F_x(x_k)\right)^{-1}\cdot\left(\underbrace{F(x^*)}_{=0} - F(x_k) - F_x(x_k))\cdot(x^* - x_k)\right)$$

$$\left\|x_{k+1} - x^*\right\| \le \frac{w}{2}\cdot\left\|x_k - x^*\right\|^2$$

Mittels vollständiger Induktion folgt hieraus $x_k \in B_r(x^*)$, $(k>0)$, denn

aus $0 < \left\|x_k - x^*\right\| \le r$ ergibt sich $\left\|x_{k+1} - x^*\right\| \le \underbrace{\frac{w}{2}\cdot\left\|x_k - x^*\right\|}_{\le r\cdot\frac{w}{2}<1}\cdot\underbrace{\left\|x_k - x^*\right\|}_{\le r} < r$.

Weiterhin ist $\left\|x_{k+1} - x^*\right\| \le a\cdot\left\|x_k - x^*\right\|$ mit $a := r\cdot\frac{w}{2} < 1$, also konvergiert (x_k) gegen x^* .

Ist x^{**} eine Nullstelle von F in $B_{\frac{2}{w}}(x^*)$, dann folgt oben mit $y := x^{**}$, $x := x^*$:

$$\left\|x^{**} - x^*\right\| = \left\| \left(F_x(x^*)\right)^{-1}\cdot\left(\underbrace{F(x^{**})}_{=0} - \underbrace{F(x^*)}_{=0} - F_x(x^*)\cdot(x^{**} - x^*)\right)\right\|$$

$$\le \frac{w}{2}\cdot\underbrace{\left\|x^{**} - x\right\|}_{<\frac{2}{w}}\cdot\left\|x^{**} - x^*\right\| \quad < \left\|x^{**} - x^*\right\|$$

, also folgt $x^* = x^{**}$ und damit die lokale Eindeutigkeit der Lösung von $F(x) = 0$. **q.e.d.**

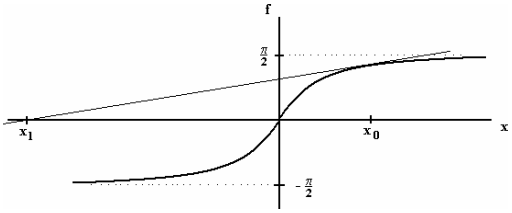
Bem. 9.12 : Gedämpftes Newton-Verfahren

Beispiel :

ges.: Nullstelle von $f(x) = \arctan x$

klassisches Newton-Verfahren :

$$x_{k+1} = x_k + p_k \qquad \text{mit} \qquad p_k := -\frac{f(x_k)}{f'(x_k)}$$



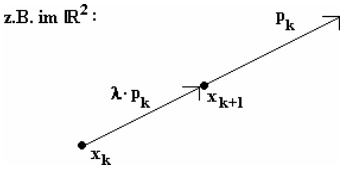
z.B. mit dem Startwert $x_0 = 10$ versagt das Verfahren (divergiert) : $\lim_{k\to\infty} f'(x_k) = 0$

Schlussfolgerung : Das Newton-Verfahren konvergiert lokal sehr schnell , hat jedoch of schlechte globale Konvergenz.

Idee : Modifiziere das Newton-Verfahren so , dass $\left\|(F_x(x_k))^{-1}\cdot F(x_{k+1})\right\| \le \left\|(F_x(x_k))^{-1}\cdot F(x_k)\right\|$.

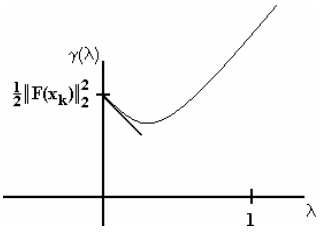
Betrachte zu p_k mit $F_x(x_k)\cdot p_k = -F(x_k)$ Vektoren $x_{k+1}(l) := x_k + l\cdot p_k$

und $g(l) := \frac{1}{2}\cdot\left\|F(x_k + l\cdot p_k)\right\|_2^2$. $\left(= \frac{1}{2}F^T F\right)$



Wegen $g'(l) = \left(F_x{}^T(x_k + l\cdot p_k)\cdot F(x_k + l\cdot p_k)\right)^T \cdot \underbrace{p_k}_{=-(F_x(x_k))^{-1}\cdot F(x_k)}$

ist $g'(0) = -F^T(x_k)\cdot F(x_k) = -\left\|F(x_k)\right\|_2^2 < 0$



Strategie :

Verringere beginnend mit $l_k{}^{(0)} := 1$ die Schrittweite l_k so lange ,

$\left(l_k{}^{(l+1)} := c\cdot l_k{}^{(l)} \text{ mit } c\in(0,1) \text{ , z.B. } c = 0.5\right)$

bis $\left\|(F_x(x_k))^{-1}\cdot F\left(x_k + l_k{}^{(l)}\cdot p_k\right)\right\|_2 \le \underbrace{\left\|(F_x(x_k))^{-1}\cdot F(x_k)\right\|}_{=-p_k}$

und setze $l_k := l_k{}^{(l)}$, $x_{k+1} := x_k + l_k p_k$.

Zusatzaufwand pro Reduktionsschritt :

1 F - Auswertung und jeweils 1 Vorwärts- und Rückwärtssubstitution

Varianten :

1.) Setze $l_k^{(0)} := \min\left(\frac{l_{k-1}}{d}, 1\right)$ mit einer Konstanten $d < 1$, z.B. d=1/2

2.) Forderung $\left\| (F_x(x_k))^{-1} \cdot F\left(x_k + l_k^{(l)} \cdot p_k\right) \right\|_2 \le \left(1 - \frac{l_k}{2}\right) \cdot \|p_k\|_2$, um möglichst „optimale“ Dämpfungsfaktoren l_k zu erreichen

praktische Erfahrung :

Gegenüber dem klassischen Newton-Verfahren verbessert sich das Konvergenzverhalten oft erheblich. Lokal quadratische Konvergenz jedoch nur , falls $l_k = 1 \quad , (k \ge k_0)$.

(D.h. l also wieder auf 1 gesetzt wird.)

Bem. 9.13 : Vereinfachtes Newton-Verfahren

Der große numerische Aufwand für häufige Neuberechnung und LR-Zerlegung der Jacobimatrix lässt sich verringern , indem die Jacobimatrix über möglichst viele Iterationsschritte konstant gehalten wird :

$$x_{k+l} := x_{k+l-1} + p_{k+l-1} \qquad \text{mit} \quad F_x(x_k) \cdot p_{k+l-1} = -F(x_{k+l-1})$$

(konvergiert linear ! , nicht quadratisch !)

Konvergenz :

$$x_{k+l} = \Phi_k(x_{k+l-1}) \qquad \text{mit} \quad \Phi_k(x) = x - (F_x(x_k))^{-1} \cdot F(x)$$

$$\frac{\partial}{\partial x}\Phi_k(x) = I - (F_x(x_k))^{-1} \cdot F_x(x) = (F_x(x_k))^{-1} \cdot (F_x(x_k) - F_x(x))$$

Ist $(F_x(x_k))^{-1} \cdot F_x(x)$ Lipschitz-stetig mit der Konstanten L , so ist

$$\left\| \frac{\partial}{\partial x}\Phi_k(x) \right\| = \left\| (F_x(x_k))^{-1} \cdot (F_x(x_k) - F_x(x)) \right\| \le L \cdot \|x - x_k\| \quad \text{und}$$

$$a := \sup\left\{ \left\| \frac{\partial}{\partial x}\Phi_k(x) \right\| : \|x - x_k\| \le \Delta_0 \right\} \le L \cdot \Delta_0 \ .$$

Also ist Φ_k kontraktiv , falls Δ_0 hinreichend klein.

Neuberechnung der Jacobimatrix :

Schätzwert für Kontraktionskonstante $a_{k+l} := \frac{\|x_{k+l} - x_{k+l-1}\|}{\|x_{k+l-1} - x_{k+l-2}\|}$ (Bem. 9.7.a)

Neuberechnung der Jacobimatrix $F_x(x_{k+l})$, falls $a_{k+l} \ge a_0$. (typische Werte : $a_0 \in [0.25 , 0.5]$)

Algorithmus :

geg.: Startwert x_0 , $a_0 \in (0,1)$

Schritt 0 : $k := 0 \quad , \quad a := 1$

Schritt 1 : if $a > a_0$ then - $J := F_x(x_k)$
- LR-Zerlegung von J
- $l := 0$, $a := 0$

Schritt 2 : Funktionsauswertung $F := F(x_k)$

Schritt 3 : Bestimme p_k mit $J \cdot p_k = -F$ mittels Vorwärts- und Rückwärtssubstitution

Schritt 4 : $x := x_k + p_k$
$l := l + 1$
if $l > 1$ then $a := \dfrac{\|p_k\|}{\|p_{k-1}\|}$

Schritt 5 : if $(l = 1 \quad or \quad a < 1)$ then - $x_{k+1} := x$
- $k := k + 1$
if Konvergenz then stop
else goto Schritt 1

Erweiterung : Kombination des vereinfachten mit dem gedämpften Newton-Verfahren

Bem. 9.14 : Berechnung der Jacobimatrix

a) Analytische Differentiation :
Empfehlenswert ist die Anwendung mathematischer Hilfsprogramme (Mathematica,Matlab) und Export der Ableitungen als C- oder Fortran-Quelltext.

Vorteil : - analytisch exakter Ausdruck zur Berechnung der Ableitungen

Nachteile : - große Bearbeitungszeiten und hoher Speicherbedarf für kompliziertere Probleme
- Der analytische Ausdruck für $F(x)$ muss in der Syntax des Hilfsprogramms formuliert werden.

b) Algorithmische Differentiation (AD) :
Ausgangspunkt : C- oder Fortran-Unterprogramm zur Auswertung einer Funktion $F : R^n \to R^m$
(input : x , output : $F(x)$)

Werkzeuge : z.B. ADIFOR , ADOL-C

Ergebnis : Zusätzliches C- oder Fortran-Unterprogramm zur Auswertung der Jacobimatrix $F_x : R^n \to R^{m \times n}$ (input : x , output : $F_x(x)$)

Vorteil : - analytisch exakte Berechnungsvorschrift

Nachteile : - sehr großer Speicherbedarf für sehr große Probleme
- Unterstützt wird in der Regel ausschließlich der Sprachstandard.

c) Differenzenapproximation :
$$\left(\frac{\partial F}{\partial x}(x)\right)_{\bullet j} = \frac{F(x + \Delta \cdot e_j) - F(x)}{\Delta}$$ (d.h. alle Elemente der j-ten Spalte)

Aufwand zur Berechnung von F_x für $F : R^n \to R^m$: n+1 Auswertungen von F

Wahl von Δ : Approximationsfehler $O(\Delta)$ und Verstärkung der Rundungsfehler $O\left(\frac{eps}{\Delta}\right)$

$\Psi(\Delta) := \Delta + \frac{eps}{\Delta}$ wird minimal für $\Delta = \sqrt{eps}$

$\Rightarrow \Delta := \sqrt{eps} \cdot \max\left\{|x_j|, \sqrt[4]{eps}\right\}$

(Wobei sich in der Praxis $\Delta := \sqrt{eps} \cdot \max\left\{\sqrt{|x_j|}, \sqrt[4]{eps}\right\}$ besser verhält.)

praktisch : Wähle Δ so , dass Δ und x_j gleiches Vorzeichen haben.

Alternativen : - Differenzenquotienten 2. Ordnung :

$$\frac{F(x + \Delta \cdot e_j) - F(x - \Delta \cdot e_j)}{2\Delta}$$

(geringerer Fehler , jedoch Verdopplung des Aufwands)

- Extrapolationsverfahren :
  Hiermit ist prinzipiell eine sehr genaue Berechnung der Ableitungen möglich , für das Newton-Verfahren jedoch zu aufwendig.

Vorteil : - sehr einfache Implementierung

Nachteile : - hoher Aufwand für $n >> 1$
- Sehr großer Fehler , falls F nicht genau ausgewertet werden kann.

typisches Beispiel : Auswertung von F beinhaltet selbst ein iteratives Verfahren (nested iterations)

d) Approximation der Jacobimatrix :
Für große Probleme kann es vorteilhaft sein , betragsmäßig sehr kleine Elemente der Jacobimatrix zu vernachlässigen.
$\Rightarrow$ Reduktion der Anzahl der Nicht-Null-Elemente

e) Matrixfreie Algorithmen :
Löst man im Newton-Verfahren die linearen Gleichungssysteme $J \cdot p_k = -F(x_k)$ nicht exakt , sondern iterativ , so kann im CG-Verfahren (und entsprechenden Verfahren) für nichtsymmetrische Matrizen die Berechnung von J vermieden werden , weil nicht J selbst , sondern ausschließlich Matrix-Vektor-Produkte $J \cdot v$ auszuwerten sind :

$$J \cdot v = \frac{F(x + \Delta \cdot v) - F(x)}{\Delta}$$

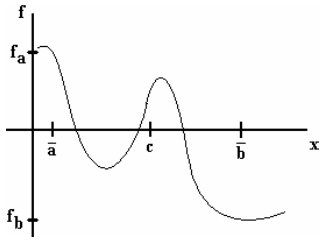
Zusammenfassung :
- Newton-Verfahren kann divergieren.
- Gedämpftes Newton-Verfahren konvergiert nur nahe der Nullstelle $(l \approx 1)$ quadratisch , sonst linear.
- Vereinfachtes Newton-Verfahren : für hochdimensionale Funktionen
  Da sich in der Nähe der Nullstelle die Ableitung kaum mehr ändert , braucht man die Jacobimatrix/Ableitungen nicht in jedem Schritt neu berechnen.

9.3 Nullstellen reeller Funktionen

Bem 9.15 : Bisektionsverfahren

geg. : $f \in C[\bar{a},\bar{b}]$ mit Vorzeichenwechsel in $[\bar{a},\bar{b}] \Rightarrow f(\bar{a}) \cdot f(\bar{b}) < 0$

ges. : $x^* \in (\bar{a},\bar{b})$ mit $f(x^*) = 0$, Existenz von x^* folgt aus Zwischenwertsatz



Algorithmus :

Initialisierung : $a := \bar{a}$, $b := \bar{b}$, $f_a := f(a)$, $f_b := f(b)$

repeat

$$c := \frac{a+b}{2} \quad , \quad f_c := f(c)$$

if $f_a \cdot f_c < 0$ **then** $b := c$, $f_b := f_c$

else $a := c$, $f_a := f_c$

until $(b - a \le TOL)$ *OR* $(f_c = 0)$

$$x^* \approx x_{bisec} := \begin{cases} c \text{ , falls } & f_c = 0 \\ \dfrac{a+b}{2} & \text{sonst} \end{cases}$$

Eigenschaften :

- einfache Implementierung
- stets konvergent (Bei Rechnung in exa kter Arithmetik gilt : $f(x^*) = 0$ für ein $x^* \in [a,b]$.)
- In Gleitpunktarithmetik muss [a,b] keine Einschließung der Nullstelle ergeben (Alternative : Intervallarithmetik)
- Anzahl der Iterationsschritte : $k \ge 1 + \log_2\left(\dfrac{\bar{b}-\bar{a}}{TOL}\right)$

Bem 9.16 : Regula falsi

Idee : Einschließung der Nullstelle wie im Bisektionsverfahren , aber Berücksichtigung des Funktionsverlaufs in [a,b].

Algorithmus :

Bestimme c als Nullstelle des (linearen) Interpolationspolynoms zu $(a, f_a), (b, f_b)$:

$$f_a + \frac{c-a}{b-a} \cdot (f_b - f_a) \overset{!}{=} 0$$

$\Rightarrow$

$$c = \frac{a \cdot f_b - b \cdot f_a}{f_b - f_a}$$

Konvergenz :

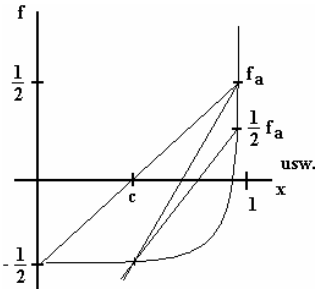
- Häufig schneller als im Bisektionsverfahren.

- In exakter Arithmetik gilt : Intervall [a,b] ist eine Einschließung der Nullstelle x^* von f .

Problem :

Langsame Konvergenz , wenn während der Iteration eines der beiden Intervallenden unverändert bleibt.

Beispiel : $f(x) = x^{10} - \frac{1}{2}$, $x^* = \sqrt[10]{\frac{1}{2}} \approx 1$, $[\overline{a}, \overline{b}] = [0,1]$



Verbesserung :

Bleibt a über 2 Iterationsschritte unverändert , so setze $f_a := \frac{1}{2} f_a$. (analog für b)

verbesserter Algorithmus :

Initialisierung : $a := \overline{a}$, $b := \overline{b}$, $f_a := f(a)$, $f_b := f(b)$, $i := 0$

repeat

$$c = \frac{a \cdot f_b - b \cdot f_a}{f_b - f_a} \quad , \quad f_c := f(c)$$

if $f_a \cdot f_c < 0$ **then**

$b := c$, $f_b := f_c$

if $i = -1$ **then** $f_a := \frac{1}{2} f_a$

$i := -1$

else

$a := c$, $f_a := f_c$

if $i = 1$ **then** $f_b := \frac{1}{2} f_b$

$i := 1$

until $(b - a \le TOL)$ *OR* $(f_c = 0)$

$$x^* \approx x_{reg} := \begin{cases} c \text{ , falls } f_c = 0 \\ \frac{a+b}{2} \quad \text{sonst} \end{cases}$$

Bem 9.17 : Sekantenverfahren

Idee : Nutze (ähnlich wie im Newton-Verfahren) Informationen über f in einer (kleinen) Umgebung von x_k , vermeide jedoch die Auswertung von f'.

Vorteil : Schnellere Konvergenz als regula falsi , falls „gute“ Startwerte bekannt sind.

Nachteil : Keine Einschließung der Nullstelle.

Verfahren :

Bestimme zu gegebenen $(x_{k-1}, f_{k-1}), (x_k, f_k)$, (mit $f_k = f(x_k)$ das lineare Interpolationspolynom und wähle x_{k+1} als dessen Nullstelle :

$$f_k + \frac{x_{k+1} - x_k}{x_{k-1} - x_k} \cdot (f_{k-1} - f_k) \stackrel{!}{=} 0 \qquad \textbf{(*)}$$

$\Rightarrow$

$$x_{k+1} := \frac{x_{k-1} \cdot f_k - x_k \cdot f_{k-1}}{f_k - f_{k-1}}$$

Alternative Interpretation : Inverse Interpolation

Bestimme das Interpolationspolynom p zu den Stützpunkten (f_{k-i}, x_{k-i}) , $(i = 0,1,...,l)$ und wähle $x_{k+1} := p(0)$.

Die Existenz von p lässt sich nachweisen für paarweise voneinander verschiedene x_{k-i} mit $|x_{k-i} - x^*| << 1$, $(i = 0,1,...,l)$, sofern x^* einfache Nullstelle von f ist.

praktische : $l =$ 2 oder 3 oder 4

Konvergenz :

$$x_{k+1} = x_k - \left(f_k \cdot \frac{x_{k-1} - x_k}{f_{k-1} - f_k} \right) \qquad \text{(nach Umstellen von (*))}$$

$$= x_k - (f_k - f(x^*)) \cdot \frac{1}{\frac{f_{k-1} - f_k}{x_{k-1} - x_k}}$$

$$= x_k - f[x_k, x^*] \cdot (x_k - x^*)$$

$$= \frac{1}{f[x_{k-1}, x_k]}$$

$\Rightarrow$

$$x_{k+1} - x^* = (x_k - x^*) \cdot \left(1 - \frac{f[x_k, x^*]}{f[x_{k-1}, x_k]} \right)$$

$$= \left(x_k - x^*\right) \cdot \left(x_{k-1} - x^*\right) \cdot \frac{f\left[x_{k-1}, x_k, x^*\right]}{f\left[x_{k-1}, x_k\right]}$$

Ist x^* einfache Nullstelle von $f \in C^2$ und x_{k-1}, x_k in einer hinreichend kleinen Umgebung von x^*, so gibt es eine Konstante $M > 0$ mit $\left|\frac{f\left[x_{k-1}, x_k, x^*\right]}{f\left[x_{k-1}, x_k\right]}\right| \le M$,

denn $f\left[x_{k-1}, x_k\right] = f'\left(x_{k-1} + J \cdot (x_l - x_{k-1})\right)$ für ein $J \in (0,1)$

und $f\left[x_{k-1}, x_k, x^*\right] = \frac{1}{2} f''(x)$ für ein $x \in [a,b]$ mit $x_{k-1}, x_k, x \in [a,b]$.

Mit $e_k := M \cdot \left|x_k - x^*\right|$ folgt $e_{k+1} \le e_k \cdot e_{k-1}$ und hieraus mittels vollständiger Induktion $e_k \le K^{q^k}$, $(k \ge 0)$ mit $K := \max\left\{e_0, \sqrt[q]{e_1}\right\}$ und $q = \frac{1}{2}\left(1+\sqrt{5}\right)$, denn $q^2 = q+1$ und $K^{q^k} \cdot K^{q^{k-1}} = K^{q^k + q^{k-1}} = K^{q^{k-1}\cdot(q+1)} = K^{q^{k-1}\cdot q^2} = K^{q^{k+1}}$.

Wählt man Startwerte x_0, x_1 so , dass $K < 1$, so folgt die Konvergenz des Sekantenverfahrens.

Die Ordnung beträgt mindestens $q = \frac{1}{2}\left(1+\sqrt{5}\right) \approx 1.618$

Bem 9.18 : Nullstellen von Polynomen

Enger Zusammenhang zwischen Polynomnullstellen und Eigenwerten von Matrizen.
(Die Eigenwertberechnung ist numerisch oft günstiger.)

Newton-Verfahren zur Berechnung einer Nullstelle von $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$

Auswertung von $p(x_k), p'(x_k)$ mittels Horner-Schema

Zur Wahl der Startwerte vgl. Literatur *Stoer* , Abschnitt 5.5.

Hat man eine Nullstelle x_1 (näherungsweise) bestimmt , so wendet man zur Bestimmung weiterer Nullstellen das Newton-Verfahren auf $p_1(x) := \frac{p(x)}{x - x_1}$ an. **(*)**

Numerisch stabile Auswertung von $\frac{p_1(x_k)}{p_1'(x_k)}$ mittels Trick von Maehly :

$$x_{k+1} = x_k - \frac{p_1(x_k)}{p_1'(x_k)} = x_k - \frac{p(x_k)}{p'(x_k) - \frac{p(x_k)}{x_k - x_1}}$$

- wegen der Ableitung von (*) : $p_1'(x) = \frac{p'(x)\cdot(x-x_1) - p(x)}{(x-x_1)^2}$

9.4 : Minimierungsprobleme

Bem 9.19 : Newton-Verfahren für Minimierungsprobleme

geg. : $f : R^n \to R$, f stetig differenzierbar

ges. : x^* mit $f(x^*) = \min_{x \in R^n} f(x)$ (= freies Minimierungsproblem)

Notwendige Bedingung für lokales Extremum : $\nabla f(x^*) = 0$

Hinweis : Gradient von f : $\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{pmatrix}$

Ist x^* einzige Nullstelle von ∇f und wächst $f(x)$ für $\|x\| \to \infty$ unbeschränkt , d.h. gibt es zu jedem $C > 0$ ein $M > 0$ mit $f(x) \geq C$ für alle $x \in R^n$ mit $\|x\| \geq M$, so ist $\nabla f(x^*) = 0$ notwendig und hinreichend für eine globale Minimalstelle von f .

Berechnung von x^* druch Anwendung des (gedämpften) Newton-Verfahrens auf $F(x) = 0$ mit $F(x) := \nabla f(x)$:

$$x_{k+1} = x_k - l_k \cdot \left(\nabla^2 f(x_k)\right)^{-1} \cdot \nabla f(x_k)$$

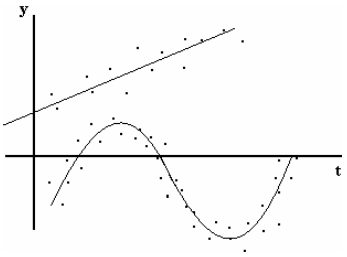
mit $\nabla^2 f(x)$ = Hessematrix = $\left(\frac{\partial^2 f}{\partial x_i \partial x_j}(x) \right)_{i,j=1}^{n} \in R^{n \times n}$.

Bestimmung des Dämpfungsparameters l_k so , dass $f(x_{k+1}) < f(x_k)$.

Praktisch sind häufig beschränkte Minimierungsprobleme anzutreffen.
D.h. Menge $E \subset R^n$ und gesucht : $x^* \in E$ mit $f(x^*) = \min \{ f(x) : x \in E \}$

Bem. 9.20 : Nichtlineare Ausgleichsprobleme

geg. : Messdaten (t_k, y_k) , $(k = 1,...,m)$ denen ein (angenommener) funktionaler Zusammenhang $y = j(t ; x_1, x_2, ..., x_n)$ zu Grunde liegt.



linear : $y = a \cdot t + b$

nichtlinear : $y = a \cdot \sin(w \cdot t + j) + b$

ges. : Parameter $x_1, x_2, ..., x_n$ so , dass $j(t_k ; x_1, x_2, ..., x_n)$ möglichst gut y_k approximiert :

$$\|F(x)\|_2 \to \min \qquad \text{mit } F(x_1,x_2,...,x_n) := \begin{pmatrix} y_1 - j(t_1\,;\,x_1,x_2,...,x_n) \\ \vdots \\ y_m - j(t_m\,;\,x_1,x_2,...,x_n) \end{pmatrix}$$

„Ausgleichsrechnung“

Spezialfall :

Ist j linear in $x_1, x_2, ..., x_n$, so ergibt sich $x_1, x_2, ..., x_n$ als Lösung eines überbestimmten linearen Gleichungssystems. Numerische Lösung mittels QR-Zerlegung.

allgemein :

Mit $f(x) := \frac{1}{2}\|F(x)\|_2^2$ ergibt sich in Bem 9.19

$$\nabla f(x) = (F_x(x))^T \cdot F(x) \qquad \text{und}$$
$$\nabla^2 f(x) = (F_x(x))^T \cdot F_x(x) + (F_{xx}(x))^T \cdot F(x).$$

Hat $F_x(x)$ Vollrang und ist $\|F(x^*)\| << 1$, so ist $(F_x(x))^T \cdot F_x(x)$ symmetrisch und positiv definit und $\nabla^2 f(x)$ regulär in einer Umgebung von x^* $\Rightarrow$ Newton-Verfahren anwendbar

Problem : Auswertung von $F_{xx}(x)$ kann sehr aufwendig sein

Bem. 9.12 : Gauß-Newton-Verfahren

geg. : $F : R^n \to R^m$, F stetig differenzierbar

ges. : $x^* \in R^n$ mit $\|F(x^*)\|_2 = \min_x \|F(x)\|_2$

Ansatz :

Linearisierung von F in x_k

$$F(x) \approx F_k(x) := F(x_k) + F_x(x_k) \cdot (x - x_k)$$

Bestimmung von $\bar{x}$ als Lösung des linearen Ausgleichsproblems

$$\min_x \frac{1}{2}\|F_k(x)\|_2^2 = \min_x \frac{1}{2}\|F(x_k) + F_x(x_k) \cdot (x - x_k)\|_2^2$$
$$= \min_x \frac{1}{2}\|Ax - b\|_2^2$$

mit $A := F_x(x_k) \in R^{m \times n}$, $b := F_x(x_k) \cdot x_k - F(x_k)$

gedämpftes Newton-Verfahren :

$$p_k := \bar{x} - x_k \quad , \quad x_{k+1} = x_k + l_k p_k$$

Normalgleichungen :

$$(F_x(x_k))^T \cdot F_x(x_k) \cdot (\bar{x} - x_k) = -(F_x(x_k))^T \cdot F(x_k)$$

Im Vergleich zum Newton-Verfahren entfällt die aufwendige Auswertung von $(F_{xx}(x_k))^T \cdot F(x_k)$

in $\nabla^2 f(x_k)$.

Ist $\|F(x_k)\| << 1$, so ist die Konvergenz jedoch ähnlich schnell wie im Newton-Verfahren

für $f(x) := \frac{1}{2}\|F(x)\|_2^2$.

Häufig Konvergenzprobleme , falls $\|F(x^*)\| = O(1)$.

Praktisch ergibt sich für derartige Probleme jedoch ohnehin eine sehr schlechte Übereinstimmung von y_k und $j(t_k ; x_1, x_2, ..., x_n)$.

10. Numerische Lösung von Differentialgleichungen

Matlab-Befehl zum Lösen von DGL : ode45

Bem 10.1 : Systeme gewöhnlicher DGL

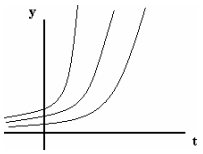
geg. : $f : [0,T] \times R^{n_y} \to R^{n_y}$ f System gew. DGL 1. Ordnung - $y'(t) = f(t, y(t))$ **(*)**

ges. : Lösung $y : [0,T] \to R^{n_y}$, $y \in C^1$, und für jedes $t \in [0,T]$ soll (*) erfüllt sein.

Beispiel :

$y' = y \quad , \quad n_y = 1$

D.h. $y(t) = c \cdot e^t \quad , c \in R$



Spezialfall :

$f(t, y) = f(t)$

$$y(t) = y(0) + \int_0^t f(t)\, dt$$

(Die Menge der Quadraturprobleme (Integration) ist Untermenge der Anfangswertprobleme von DGL.)

klassisch : $y(t_n)$ für „viele“ $t_n \in [0,T]$

besser : „stetige Lösungsdarstellung“ (dense output)

$\tilde{y}(t) \in C[0,T]$ mit $\tilde{y}(t) \approx y(t)$, $(t \in [0,T])$

Systeme höherer Ordnung : Äquivalenz zu Systemen 1.Ordnung (siehe Analysis III)

Nichtautonome Systeme :

Häufig betrachtet man nur autonom Systeme $y'(t) = f\big(t, y(t)\big)$, denn diese sind äquivalent zu $Y'(t) = F\big(Y(t)\big)$ mit $Y(t) = \begin{pmatrix} t \\ y(t) \end{pmatrix} \in R^{n_y+1}$, $F(Y) = \begin{pmatrix} 1 \\ f(t,y) \end{pmatrix}$

- Anfangswertprobleme :

  Vorgabe von n Anfangsbedingungen $y(0) = y_0 \in R^{n_y}$ zu (*)

  Satz von Picard-Lindelöf :
  Ist f stetig und Lipschitz-stetig bzgl. y , so hat das AWP zu (*) eine eindeutige Lösung.

  $$y^{(k+1)}(t) = y_0 + \int_0^t f\big(t, y^{(k)}(t)\big)\, dt \qquad ,(k \geq 0) \qquad , \; y^{(0)}(t) \equiv y_0 \qquad ,\big(t \in [0,T]\big)$$

  praktisch : sehr langsame Konvergenz $\Rightarrow$ für praktische Rechnungen ungeeignet
  (jedoch gut parallelisierbar)

- Randwertprobleme :

  Vorgabe von insgesamt n_y Bedingungen an die Randwerte $y(0)$ und $y(T)$:

  $$r\big(y(0), y(T)\big) = 0 \qquad \text{mit } r : R^{n_y} \times R^{n_y} \to R^{n_y}$$

  Beispiel :

  $$x'' + x = 0 \qquad \textbf{(*)}$$

  $$y = \begin{pmatrix} x \\ x' \end{pmatrix} \qquad , \qquad y_1' = y_2 \quad , \quad y_2' = -y_1$$

  $$y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} \qquad , \qquad f(t,y) = \begin{pmatrix} y_2 \\ -y_1 \end{pmatrix}$$

  AWP :

  $$x(0) = y_1(0) = x_0$$
  $$x'(0) = y_2(0) = x_0'$$

  RWP :

  $$[0,T] = [0,p]$$

  Lösung von (*) : $x(t) = c_1 \sin t + c_2 \cos t$ mit $c_1, c_2 \in R$

  a) $x(0) = x(p) = 0 \quad = y_1(0) = y_1(p)$
  $\Rightarrow c_2 = 0$
  D.h. $x(t) = c_1 \sin t$ mit $c_1 \in R$

  b) $x(0) = y_1(0) = 0 \quad , \quad x(p) = y_1(p) = 1$
  Aus $x(0) = 0$ folgt $c_2 = 0$
  Aus $x(p) = 1$ folgt $c_2 = -1$
  D.h. widersprüchliche Bedingungen $\Rightarrow$ nicht lösbar

Im Allgemeinen nur lokale Existenz- und Eindeutigkeitsaussagen in einer Umgebung einer (isolierten) Lösung.

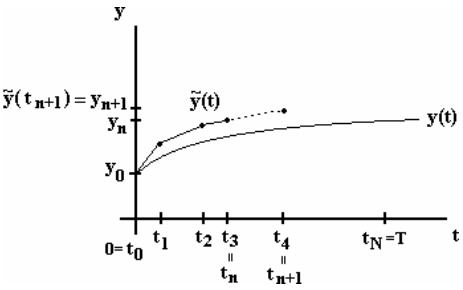
Bem 10.2 : Diskretisierungsverfahren für gewöhnliche DGL

Def. Zeitgitter : $\Delta := \{t_0, t_1, ..., t_N \ : \ 0 = t_0 < t_1 < ... < t_N = T\}$

Explizites Eulerverfahren :

Sei Näherungslösung $\tilde{y}(t)$ gegeben auf $[0, t_n]$ und $y_n := \tilde{y}(t_n)$.

Gesucht : Stetige Fortsetzung auf $[0, t_{n+1}]$ mit $t_{n+1} - t_n =: h_n > 0$.



$$t \in [t_n, t_{n+1}]: \quad y(t) = y(t_n) + \int_{t_n}^{t} y'(t)\, dt \overset{(*)}{=} y(t_n) + \int_{t_n}^{t} f(t, y(t))\, dt$$

$$\approx \tilde{y}(t_n) + \int_{t_n}^{t} f(t_n, \tilde{y}(t_n))\, dt = y_n + (t - t_n) \cdot f(t_n, y_n)$$

$$y(t_{n+1}) \approx \tilde{y}(t_{n+1}) := y_{n+1} := y_n + \underbrace{h_n}_{= t_{n+1} - t_n} \cdot f(t_n, y_n)$$

Verfahrensvorschrift des expliziten Eulerverfahrens :

$$\frac{y_{n+1} - y_n}{h_n} = f(t_n, y_n)$$

Stetige Lösungsdarstellung (Polygonzugverfahren) :

$$y(t_n + q \cdot h_n) \approx \tilde{y}(t_n + q \cdot h_n) := y_n + q \cdot h_n \cdot f(t_n, y_n) = y_n + q \cdot (y_{n+1} - y_n) \qquad , (q \in [0,1])$$

Trapezregel :

$$\tilde{y}(t_{n+1}) = \tilde{y}(t_n) + \frac{h_n}{2} \cdot \left(f(t_n, \tilde{y}(t_n)) + f(t_{n+1}, \tilde{y}(t_{n+1})) \right)$$

$$\frac{y_{n+1} - y_n}{h_n} = \frac{1}{2} \cdot \left(f(t_n, y_n) + f(t_{n+1}, y_{n+1}) \right)$$

- Nichtlineares Gleichungssystem zur Bestimmung von y_{n+1}
- Zur Lösung von Anfangswertproblemen nur geeignet , falls h_n wesentlich größer sein kann als im Eulerverfahren.
- Vorteilhaft für RWP , da symmetrisches Verfahren

Definition 10.3 : Einschrittverfahren (ESV)

Zur numerischen Lösung des AWP $y(0) = y_0$ zu (*) wird die Lösung $y(t)$ approximiert durch eine Gitterfunktion $y_\Delta : \Delta \to R^{n_y}$ mit $y_n := y_\Delta(t_n) = \tilde{y}(t_n) \approx y(t_n)$.

Einschrittverfahren haben die Form $y_{n+1} = y_n + h_n \cdot j\left(t_n, y_n ; h_n, f\right)$ mit der Verfahrensfunktion j .

Euler : $j\left(t_n, y_n ; h_n, f\right) = f\left(t_n, y_n\right)$

Trapezregel : j ergibt sich als Lösung eines nichtlinearen Gleichungssystems

Bem. 10.4 : Explizite Runge-Kutta-Verfahren

S-stufiges explizites Runge-Kutta-Verfahren

$$y_{n+1} = y_n + h_n \cdot \sum_{j=1}^{S} b_j \cdot f\left(t_n + c_j \cdot h_n \ , \ Y_{nj}\right)$$

mit Stufenvektoren $Y_{ni} = y_n + h_n \cdot \sum_{j=1}^{i-1} a_{ij} \cdot f\left(t_n + c_j \cdot h_n \ , \ Y_{nj}\right)$ $,(i = 1,...,S)$

$$Y_{n1} = y_n$$
$$Y_{n1}' := f\left(t_n + c_1 h_n \ , Y_{n1}\right)$$

$$Y_{n2} = y_n + h_n \cdot a_{21} \cdot f\left(t_n + c_1 h_n \ , Y_{n1}\right) = y_n + h_n \cdot a_{21} \cdot Y_{n1}'$$
$$Y_{n2}' := f\left(t_n + c_2 h_n \ , Y_{n2}\right)$$

...

$$y_{n+1} = y_n + h_n \cdot \sum_{j=1}^{S} b_j \cdot Y_{nj}'$$

mit :

b_j - Gewichte

c_i - Knoten

a_{ij} - Runge-Kutta-Paramter

Butcher-Schema :

c_2	a_{21}				
c_3	a_{31}	a_{32}			
$\vdots$	$\vdots$		$\ddots$		
c_S	a_{S1}	a_{S2}	$\cdots$	$a_{S,S-1}$	
	b_1	b_2	$\cdots$	b_{S-1}	b_S

Spezialfall :

$$y'(t) = 1$$

$$Y_{ni} = y_n + h_n \cdot \sum_{j=1}^{i-1} a_{ij} \cdot 1$$

Stufenvektoren $Y_{ni} \approx y(t_n + c_i h_n)$

$$y(t_n + c_i h_n) = y(t_n) + c_i h_n$$
$$c_i = \sum_{j=1}^{i-1} a_{ij} \qquad ,(i = 1,...,S)$$

stetige Lösungsdarstellung :

$$y(t_n + q \cdot h_n) \approx y_n + h_n \cdot \sum_{j=1}^{S} b_j(q) \cdot Y_{nj}'$$

mit $b_j(0) = 0$ und $b_j(1) = b_j$ für $j = 1,...,S$

Beispiel :

Arenstdorf-Orbit (siehe evt. Folien)

a) Explizites Eulerverfahren $h = 5.0_{E-4}$ sehr schlechte Näherung (unbrauchbar)
$4 \cdot 10^4$ Integrationsschritte und $4 \cdot 10^4$ Funktionsauswertungen

b) Explizites Eulerverfahren $h = 2.0_{E-5}$ akzeptable Näherung
10^6 Integrationsschritte und 10^6 Funktionsauswertungen

c) Runge-Kutta $h = 2.0_{E-3}$ akzeptable Näherung
$4 \cdot 10^4$ Integrationsschritte und $4 \cdot 10^4$ Funktionsauswertungen

d) DOPRI5 (Dormand/Prince) sehr gute Näherung
75 Integrationsschritte und 535 Funktionsauswertungen

zum klassischen Runge-Kutta-Verfahren :

$S = 4$ (d.h. 4stufig)

$$Y_{n1} = y_n$$
$$Y_{n2} = y_n + \frac{h_n}{2} \cdot f(t_n, y_n)$$
$$Y_{n3} = y_n + \frac{h_n}{2} \cdot f\left(t_n + \frac{h_n}{2}, Y_{n2}\right)$$
$$Y_{n4} = y_n + \frac{h_n}{2} \cdot f\left(t_n + \frac{h_n}{2}, Y_{n3}\right)$$

$$y_{n+1} = y_n + \frac{h_n}{6} \cdot \left(f(t_n, y_n) + 2 \cdot f\left(t_n + \frac{h_n}{2}, Y_{n2}\right) + 2 \cdot f\left(t_n + \frac{h_n}{2}, Y_{n3}\right) + f\left(t_n + \frac{h_n}{2}, Y_{n4}\right) \right)$$

Butcher-Schema :

1/2	1/2			
1/2	0	1/2		
1	0	10	1	

	1/6	1/3	1/3	1/6

Bem 10.5 : Diskretisierungsfehler

a) Lokaler Fehler des ESV (local error) :

$$y(t_{n+1}) - \left(y(t_n) + h_n \cdot j\left(t_n, y(t_n)\ ;\ h_n,\ f\right)\right)$$

Beispiel : Expliztes Euler-Verfahren

$$y_{n+1} = y_n + h_n \cdot f(t_n, y_n)$$

$$(le)_n := y(t_{n+1}) - \left(y(t_n) + h_n \cdot \underbrace{f\left(t_n, y(t_n)\right)}_{= y'(t_n)} \right)$$

$$y(t_{n+1}) = y(t_n + h_n) = y(t_n) + h_n \cdot y'(t_n) + O\left(h_n{}^2\right)$$

$$\Rightarrow (le)_n = O\left(h_n{}^2\right)$$

Runge-Kutta : $(le)_n = O\left(h_n{}^5\right)$

DOPRI5 : $(le)_n = O\left(h_n{}^6\right)$

b) Globaler Fehler :

$$e_N := \left\| y_N - y(t_N) \right\|$$

$$\left\| y_{n+1} - y(t_{n+1}) \right\| = \left\| \begin{array}{l} y_n + h_n \cdot j\left(t_n, y_n\ ;\ h_n,\ f\right) \ - \ \left(y(t_n) + h_n \cdot j\left(t_n, y(t_n)\ ;\ h_n,\ f\right)\right) \\ + \ \underbrace{\left(y(t_n) + h_n \cdot j\left(t_n, y(t_n)\ ;\ h_n,\ f\right)\right) - y(t_{n+1})}_{= -(le)_n} \end{array} \right\|$$

wegen Dreiecksungleichung folgt

$$\leq \left\| y_n + h_n \cdot j\left(t_n, y_n\ ;\ h_n,\ f\right) \ - \ \left(y(t_n) + h_n \cdot j\left(t_n, y(t_n)\ ;\ h_n,\ f\right)\right) \right\| + \left\| (le)_n \right\|$$

$$\leq \left\| y_n - y(t_n) \right\| + h_n \cdot \left\| j\left(t_n, y_n\ ;\ h_n,\ f\right) - j\left(t_n, y(t_n)\ ;\ h_n,\ f\right) \right\| + \left\| (le)_n \right\|$$

$$\leq \left\| y_n - y(t_n) \right\| + h_n \cdot L \cdot \left\| y_n - y(t_n) \right\| + \left\| (le)_n \right\| \qquad \text{(Lipschitz-Konstante L)}$$

$$= (1 + h_n \cdot L) \cdot \left\| y_n - y(t_n) \right\| + \left\| (le)_n \right\|$$

mit $e_n := \left\| y_n - y(t_n) \right\|$ folgt die Rekursion :

$$\begin{aligned} e_{n+1} &\leq (1 + h_n \cdot L) \cdot e_n + \left\| (le)_n \right\| \\ &\leq e^{h_n \cdot L} \cdot e_n + \left\| (le)_n \right\| \end{aligned}$$

$$e_n \leq e^{h_{n-1} \cdot L} \cdot e_{n-1} + \left\| (le)_{n-1} \right\|$$

Einsetzen :

$\Rightarrow$

$$e_{n+1} \leq e^{h_n \cdot L} \cdot e^{h_{n-1} \cdot L} \cdot e_{n-1} + \left\| (le)_n \right\| + e^{h_n \cdot L} + \left\| (le)_{n-1} \right\|$$

$$\leq \ldots$$

$$\leq e^{(h_n + h_{n-1} + \ldots + h_0) \cdot L} \cdot e_0 + \left\| (le)_n \right\| + e^{h_n \cdot L} + \left\| (le)_{n-1} \right\| + e^{(h_n + h_{n-1}) \cdot L} \cdot \left\| (le)_{n-2} \right\| + \ldots + e^{(h_n + h_{n-1} + \ldots + h_1) \cdot L} \cdot \left\| (le)_0 \right\|$$

wegen $h_0 + h_1 + \ldots + h_n = t_{n+1} - t_0 \leq T$ folgt

$$e_{n+1} \leq e^{L \cdot T} \cdot \sum_{i=0}^{n} \left\| (le)_i \right\|$$

Für $h_n = h = const$ ist $e_{n+1} \leq e^{L \cdot T} \cdot \sum_{i=0}^{n} C \cdot h^2 = e^{L \cdot T} \cdot n \cdot C \cdot h \cdot h$ und

wegen $n \cdot h < T$ ist $e_{n+1} \leq C \cdot e^{LT} \cdot T \cdot h$.

(=Beschreibung des globalen Fehlers des Euler-Verfahrens.)
D.h. Halbierung der Schrittweite entspricht einer Halbierung des Fehlers.

Für Runge-Kutta : $e_{n+1} \leq C \cdot e^{LT} \cdot T \cdot h^4$

Für DOPRI5 : $e_{n+1} \leq C \cdot e^{LT} \cdot T \cdot h^5$ (Halbierung der Schrittweite bedeutet $\frac{1}{32}$ Fehler)