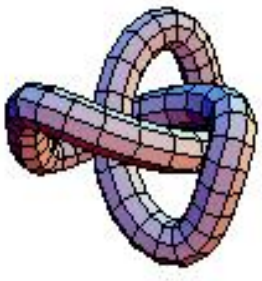


Numerik I
Vorlesungsskript

TU Chemnitz
Fakultät für Mathematik
Volker Mehrmann

Literatur:

(a) J.H. Wilkinson
Rundungsfehler, Springer

(b) G.H. Golub/C.F. Van Loan
Matrix Computations, Johns Hopkins University Press

(c) A. Frommer
Lösung linearer Gleichungssysteme auf Parallelrechnern, Vieweg

(d) J. Stoer
Numerische Mathematik I, Springer

e) H.R. Schwarz
Numerische Mathematik, Teubner

u.v.a.m.

Skript und weitere Informationen

http://www.tu–chemnitz.de/∼mehrmann/nla/

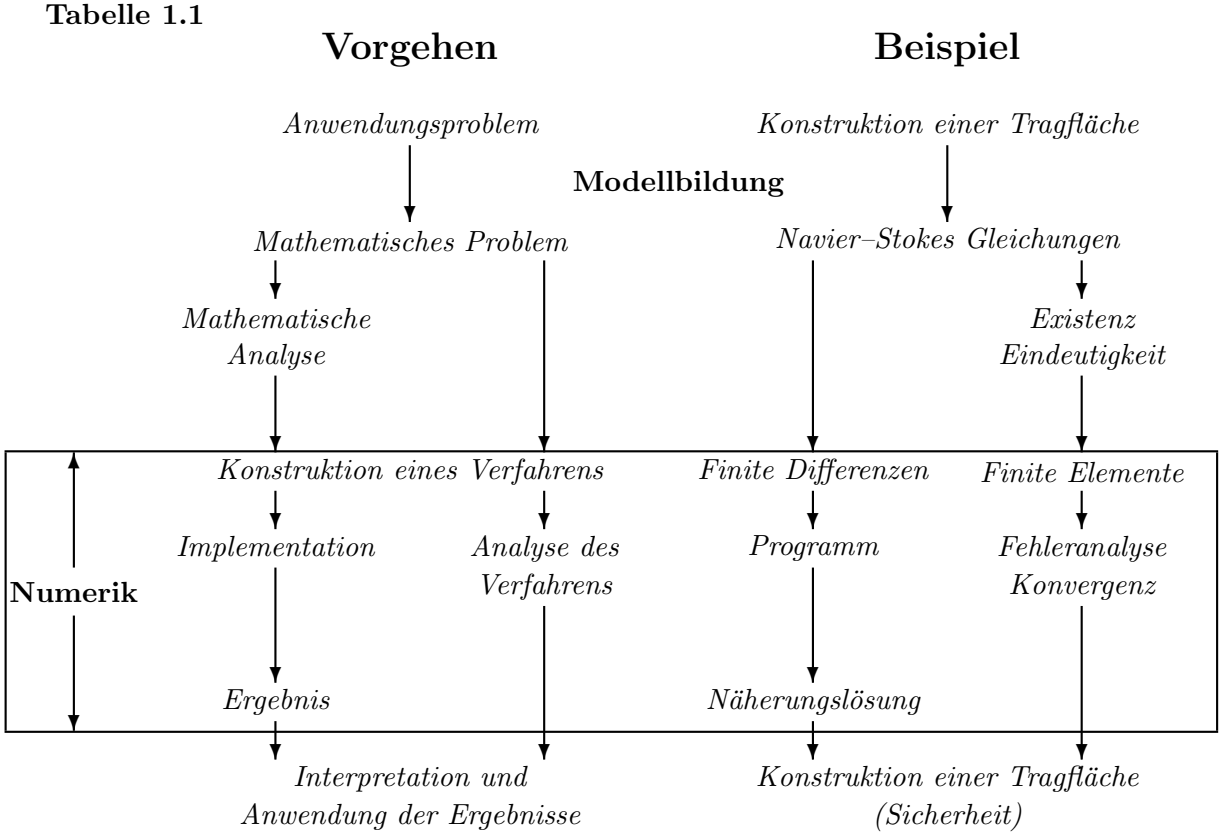
Inhaltsverzeichnis

Kapitel 1

Grundlagen

1.1 Was ist numerische Mathematik?

Tabelle 1.1



Gesichtspunkte:

- Genauigkeit des Verfahrens
- Verlässlichkeit der Ergebnisse
- Effizienz des Verfahrens
- Laufzeiten
- Black–box, special purpose,...

1.2 Zahlendarstellung auf dem Digitalrechner

Die Zahlendarstellung auf dem Rechner verwendet den folgenden Satz:

Satz 1.2 (p–adische Entwicklung)
Ist $r \in \mathbb{R}, p \in \mathbb{N}, p > 1$, so gibt es ein eindeutiges $j \in \{0,1\}$, ein eindeutiges $l \in \mathbb{Z}$ und für alle $k \in \mathbb{Z}$ mit $k \leqslant l$, eindeutige $\gamma_k \in \mathbb{Z}$, so dass

$$r = (-1)^j \sum_{k=-\infty}^{l} \gamma_k p^k \tag{1.3}$$

wobei für $r \neq 0, \gamma_l \neq 0$ und $j = l = 0$ für $r = 0, 0 \leqslant \gamma_k < p$ für alle $k \leqslant l, \gamma_k < p-1$ für unendliche viele $k \leqslant l$.

Bemerkung 1.4 *Die letzte Bedingung schließt Zahlen wie $3.9999\bar{9}$ aus.*

Beispiel 1.5 $r = 18.5 \qquad p = 2$

$$\begin{aligned} r &= (-1)^0(1*2^{-1} + 0*2^0 + 1*2^1 + 0*2^2 + 0*2^3 + 1*2^4) \\ &= 10010.1 \text{ } \textit{Dualzahldarstellung oder Binärdarstellung} \\ p &= 16 \\ r &= (-1)^0(8*16^{-1} + 2*16^0 + 1*16) \\ &= 12.8 \text{ } \textit{Hexadezimaldarstellung} \\ &\quad \textit{Ziffern } (0123456789ABCDEF). \end{aligned}$$

$\longrightarrow$ *Übung 1.43.*

Die Konvertierung zwischen verschiedenen Darstellungen geschieht durch Division durch p mit Rest für ganzzahligen Anteil und Multiplikation mit p für Nachpunktanteil. Nach Satz 1.2 ist

$$r = \left[\underbrace{(-1)^j \sum_{k=-\infty}^{l} (\gamma_k p^{k-l-1})}_{a} \right] \cdot \underbrace{p^{l+1}}_{b} \tag{1.6}$$

wobei $\frac{1}{p} < |a| < 1$, da $\gamma_l \neq 0$.

Definition 1.7 *Die Darstellung $r = a \cdot p^b$ heisst normalisierte Gleitpunktdarstellung von r bezüglich p. a heisst Mantisse, b der Exponent von r.*

$$\begin{aligned} a &= V_M \sum_{i=1}^{\infty} \alpha_i p^{-i} \quad \text{mit } \alpha_1 \neq 0 \\ b &= V_E \sum_{i=1}^{t} \beta_i p^{t-i}, \quad V_m, V_E \text{ Vorzeichen .} \end{aligned}$$

! Auf einem Rechner ist für die Darstellung (im allgemeinen, in normierter Sprache) nur eine feste Anzahl von n Stellen möglich. Diese n Stellen werden aufgeteilt in l Stellen für die Mantisse und m Stellen für den Exponenten. Die Zahl

$$V_M(\alpha_1 p^{-1} + \alpha_2 p^{-2} + \cdots + \alpha_l p^{-l}) p^{V_E(\beta_1 p^{m-1} + \cdots + \beta_m p^0)} \tag{1.8}$$

mit $\alpha_1 \neq 0$ wird durch die Angabe $V_M \alpha_1 \alpha_2 \cdots \alpha_l V_E \beta_1 \cdots \beta_m$ codiert.

Beispiel 1.9

$$\begin{aligned} r_1 &= -1234.567890, r_2 = 0.01234567890 \\ p &= 10, l = 9, m = 2 \end{aligned}$$

normalisierte Gleitpunktdarstellung

$$\begin{aligned} r_1 &= -0.123456789 * 10^4 \\ &= -123456789 + 04 \\ r_2 &= -0.123456789 * 10^{-1} \\ &= -123456789 - 01 \end{aligned}$$

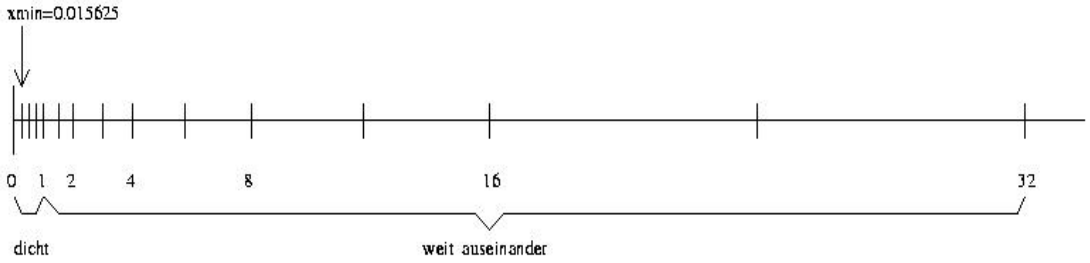
Definition 1.10 *$\mathbb{M}(p, l, m)$ bezeichnet die Menge der in normalisierter Gleitpunktdarstellung codierbaren (darstellbaren) Zahlen, (auch Maschinenzahlen genannt) mit Basis p, l–Stellen für die Mantisse und m–Stellen für den Exponenten (jeweils zur Basis p).*

Beispiel 1.11 $\mathbb{M}(2, 3, 3)$*, d.h.* $p = 2, l = m = 3$

verfügbare Mantissen			verfügbare Expontenten			
			± 000	$\widehat{=}$	± 0	$1,\ 1$
± 100	$\widehat{=}$	$\pm\frac{1}{2}$	± 001	$\widehat{=}$	± 1	$2,\ \frac{1}{2}$
			± 010	$\widehat{=}$	± 2	$4,\ \frac{1}{4}$
± 101	$\widehat{=}$	$\pm\frac{5}{8}$	± 011	$\widehat{=}$	± 3	$8,\ \frac{1}{8}$
			± 100	$\widehat{=}$	± 4	$16,\ \frac{1}{16}$
± 110	$\widehat{=}$	$\pm\frac{3}{4}$	± 101	$\widehat{=}$	± 5	$32,\ \frac{1}{32}$
			± 110	$\widehat{=}$	± 6	$64,\ \frac{1}{64}$
± 111	$\widehat{=}$	$\pm\frac{7}{8}$	± 111	$\widehat{=}$	± 7	$128,\ \frac{1}{128}$

kleinste positive Zahl $x_{\min} = \frac{1}{2} 2^{-7} = 2^{-8}$
größte positive Zahl $x_{\max} = \frac{7}{8} 2^7 = 112$

Maschinenzahlen sind wie folgt angeordnet



1.2.1 Rundungsfehler (Reduktionsfehler)

Von nun an sei p entweder 10 oder 2^k. Um eine beliebige reelle Zahl darzustellen wird sie gerundet in die Menge der Maschinenzahlen abgebildet,

$$\gamma : \mathbb{R} \longrightarrow \mathbb{M}(p, l, m).$$

Sei $x = \pm \left(\sum_{i=1}^{\infty} \alpha_i p^{-i}\right) \cdot p^t, \alpha_1 \neq 0, x \in Z = [x_{\min}, x_{\max}] \cup [-x_{\max}, -x_{\min}]$

Übliche Rundung auf l Stellen

$$\gamma(x) = \begin{cases} \pm & \left(\sum_{i=1}^{l} \alpha_i p^{-i}\right) \cdot p^t & \text{falls } \alpha_{l+1} < \frac{p}{2} \\ \pm & \left(\sum_{i=1}^{l} (\alpha_i p^{-i}) + p^{-l}\right) \cdot p^t & \text{falls } \alpha_{l+1} \geqslant \frac{p}{2}. \end{cases} \tag{1.12}$$

Bei Zahlen ausserhalb von Z gibt es rechnerabhängige Vorgehensweisen z.B.
$\longrightarrow$ Übung 1.45

$$|x| < x_{\min} \begin{cases} \gamma(x) = 0 \\ \gamma(x) = \text{sign } (x) x_{\min} \end{cases} \text{'underflow'}$$

$$|x| > x_{\max} \begin{cases} \gamma(x) = \text{sign } (x) x_{\max} \text{ mit Meldung 'overflow'} \\ \text{'overflow'} \end{cases}$$

$\longrightarrow$ Übung 1.45: bestimme $x_{\min}, x_{\max}$

Absoluter Rundungsfehler

$$|\gamma(x) - x| \leqslant \frac{p}{2}^{-l} p^t \tag{1.13}$$

Beweis:

abrunden:

$$\begin{aligned} |\gamma(x) - x| &= \left| \left(\sum_{i=1}^{\infty} \alpha_i p^{-i}\right) p^t - \left(\sum_{i=1}^{l} \alpha_i p^{-i}\right) p^t \right| \\ &= p^t \cdot \left| \sum_{i=l+1}^{\infty} \alpha_i p^{-i} \right| = p^t p^{-(l+1)} \left| \underbrace{\sum_{i=0}^{\infty} \alpha_{l+i+1} p^{-i}}_{\leqslant \frac{p}{2}} \right| \\ &\leqslant p^t \frac{p}{2}^{-l}. \end{aligned}$$

aufrunden:

$$\begin{aligned}
|\gamma(x)-x| &= \left|\left(\sum_{i=1}^{\infty}\alpha_i p^{-i}\right)p^t - \left(\sum_{i=1}^{l}(\alpha_i p^{-i})+p^{-l}\right)p^t\right| \\
&= \left|\sum_{i=l+1}^{\infty}\alpha_i p^{-i} - p^{-l}\right| p^t \\
&= p^t\left|-p^{-l}+\alpha_{l+1}p^{-(l+1)}+\cdots\right| \\
&= p^t p^{-l}\left|-1+\alpha_{l+1}p^{-1}+\alpha_{l+2}p^{-2}+\cdots\right| \\
&\leqslant p^t p^{-l}\left(-\frac{p}{2}p^{-1}+1\right) = p^t\frac{p}{2}^{-l}.
\end{aligned}$$

□

Relativer Rundungsfehler

$$\frac{|\gamma(x)-x|}{|x|} \leqslant \frac{\frac{p}{2}^{-l}p^t}{M_x p^t}$$

wobei die Mantisse M_x von x die Ungleichung $M_x \geqslant \frac{1}{p}$ erfüllt. Bei $M_x = \frac{1}{p}$,d.h. $x = \frac{p^t}{p}$ gibt es keinen Fehler. Also

$$\left|\frac{\gamma(x)-x}{x}\right| \geqslant \frac{\frac{p}{2}^{-l}}{\frac{1}{p}} = \frac{p}{2}p^{-l} \tag{1.14}$$

Die Zahl

$$\text{eps} := \frac{p}{2}p^{-l} =: (\text{macheps}) \tag{1.15}$$

heisst Maschinengenauigkeit (roundoff unit).
$\longrightarrow$ Übung: 1.45 Bestimme eps für Deine Maschine.

Es gilt immer $\text{eps} = \min_{\delta\in[x_{\min},x_{\max}]}|\gamma(1+\delta)| > 1$.
$1 = 1 * p^{-1}p^1$.

Bemerkung 1.16 *Es gilt also stets*

$$\begin{aligned}
\gamma(x) &= x\cdot(1+\varepsilon)\ \textit{mit}\ |\varepsilon| \leqslant \text{eps}, \\
\varepsilon &= \tfrac{\gamma(x)-x}{x}.
\end{aligned}$$

Beispiel 1.17
0.2 *dezimal* $l = 6, p = 2$
ergibt $0.\overline{0011}$ *in dualer Darstellung*
normalisiert $0.11\overline{0011} * 2^{-2}$
Rundung $l = 6$ *ergibt* $0.110011 * 2^{-2}$
Rückkonvertiert $\frac{51}{256} = 0.19921875$

Rundungsfehler entstehen beim Einlesen, Speichern, Konvertieren, Rechnen.

Man hat auf dem Rechner nur eine Pseudoarithmetik.
$\mathbb{M}$ ist nicht abgeschlossen bzgl. $+, -, *, \div$

Beispiel 1.18 $\mathbb{M}(10,5,1)$

$$\begin{aligned} x &= +25684+1 \;\widehat{=}\; 2.5684 \\ y &= +32791-2 \;\widehat{=}\; 0.0032791 \end{aligned}$$

$$\begin{aligned} x+y &= \underbrace{2.5716}_{5}781 \notin \mathbb{M}, x*y = 0.00\underbrace{842204}_{5}4044 \notin \mathbb{M} \\ x/y &= \underbrace{783.26}_{5}37004 \notin \mathbb{M}. \end{aligned}$$

Die übliche Arithmetik in $\mathbb{R}$ muss so realisiert werden, dass das Ergebnis wieder in $\mathbb{M}$ ist.

$\longrightarrow$ Übung
Im allgemeinen gilt:
Das Ergebnis der Pseudooperation $\overline{\nabla}$, $\nabla \in \{+,-,*,\div\}$ ist festgelegt durch

$$gl(x\nabla y) \equiv x\,\overline{\nabla}\,y = \gamma(x\nabla y), \text{ für } x,y \in \mathbb{M} \tag{1.19}$$

Hardwaremäßig wird üblicherweise mit längerer Mantisse gearbeitet als im Ergebnis, dies normalisiert und dann gerundet.

! Vorsicht: Nicht auf allen Rechnern, man kann böse Überrschungen erleben. (z.B. auf Rechnern die nicht mit IEEE–Standard–Arithmetik arbeiten z.B. Cray.)

Unter der Annahme (1.19) gilt:
Falls $|x\nabla y|$ in $[x_{\min}, x_{\max}]$ liegt, so gilt für den relativen Fehler

$$\left|\frac{x\,\overline{\nabla}\,y - x\nabla y}{x\nabla y}\right| = \left|\frac{\gamma(x\nabla y) - x\nabla y}{x\nabla y}\right| < \text{eps} \tag{1.20}$$

In $\mathbb{R}$ gelten Assoziativ–, Kommutativ– und Distributivgesetz, in der Rechnerarithmetik in $\mathbb{M}$ gilt im allgemeinen nur die Kommutativität für Addition und Multiplikation.

Beispiel 1.21 $\mathbb{M}(10,5,1)$

Assoziativität

$$\begin{aligned} &0.98765 + 0.012424 - 0.0065432 = 0.\underline{9935}308 \\ &0.98765 \oplus (0.012424 \ominus 0.0065432) = \\ &0.98765 \oplus 0.00588(08) = 0.\underline{99353}(08) \\ &(0.98765 \oplus 0.012424) \ominus 0.0065432 = \\ &\underbrace{1.000074}_{1.0001} \ominus 0.0065432 = 0.\underline{99351} \end{aligned}$$

Distributivität

$$\begin{aligned} &(4.2832 - 4.2821) * 5.7632 = 0.00\underline{633952} \\ &(4.2832 \ominus 4.2821) \otimes 5.7632 = \\ &0.0011 \otimes 5.7632 = 0.00\underline{63395}(2) \\ &(4.2832 \otimes 5.7632) \ominus (4.2821 \otimes 5.7632) = \\ &\quad \underbrace{24.684(93824)}_{24.685\ominus 24.679 = 0.006\underline{0000}} \quad \ominus 24.678(59872) \end{aligned}$$

! Folgerung: Mathematisch äquivalente Algorithmen zur Auswertung eines rationalen Ausdrucks können bei Durchführung auf dem Rechner zu wesentlich verschiedenen Ergebnissen führen, selbst wenn die Eingabedaten Maschinenzahlen sind. Nichtrationale Operationen wie

$\sqrt{}, \sin, \exp$ sind (nicht immer gut) softwaremäßig realsiert. Auch hierfür gilt im allgemeinen: Das Maschinenergebnis ist Rundung des exakten Ergebnisses.

$\longrightarrow$ Übung.

Auf einigen Maschinen ist selbst $\div$ nicht fest realisiert (z.B. Cray) oder man hat die Wahl zwischen (billig und schlecht oder teuer und gut).

1.3 Fehlerfortpflanzung

Unglücklicherweise planzen sich einmal bereits gemachte Fehler (z.B. bei Rundung) auch noch fort. Seien x, y mit Fehlern $\triangle x, \triangle y$ behaftet $\left(\left|\frac{\triangle x}{x}\right|, \left|\frac{\triangle y}{y}\right| \ll 1\right)$. Was passiert bei exakter Durchführung einer elementaren Operation mit diesen Fehlern, d.h. für $\nabla \in \{+, -, *, \div\}$ sei $x \nabla y \in \mathbb{M}$

Fortpflanzung des relativen Fehlers:

$$(x + \triangle x)\nabla(y + \triangle y) = x \nabla y + \underbrace{\triangle(x \nabla y)}_{\text{abs. Fehler des Ergebnisses}}$$

$\pm : x \pm y + \triangle x \pm \triangle y :$

$$\frac{\triangle(x \pm y)}{x \pm y} = \frac{\triangle x \pm \triangle y}{x \pm y} = \frac{x}{x \pm y}\frac{\triangle x}{x} \pm \frac{y}{x \pm y}\frac{\triangle y}{y} \tag{1.22}$$

Bemerkung 1.23 *Ist $|x \pm y|$ sehr klein gegenüber $|x|$ oder $|y|$, so kann der relative Fehler ausserordentlich verstärkt werden, z.B. wenn zwei annähernd gleich große Zahlen subtrahiert werden. Dieser Effekt heisst Auslöschung. Man muss bei der Aufstellung der Algorithmen darauf achten, dass dies soweit wie möglich vermieden wird.*

$\longrightarrow$ *Übung*

$$* : xy + \underbrace{x \triangle y + y \triangle x + \triangle y \triangle x}_{\triangle(x*y)} :$$

$$\frac{\triangle(x * y)}{x * y} = \frac{\triangle y}{y} + \frac{\triangle x}{x} + \frac{\triangle y \triangle x}{xy} \approx \frac{\triangle y}{y} + \frac{\triangle x}{x} \tag{1.24}$$

Die relativen Fehler addieren sich also im wesentlichen.

$$\div : \frac{x + \triangle x}{y + \triangle y} = \frac{x}{y} + \frac{y \triangle x - x \triangle y}{y(y + \triangle y)} :$$

$$\begin{aligned} \frac{\triangle(x \div y)}{x \div y} &= \frac{1}{x \div y}\left(\frac{y \triangle x - x \triangle y}{y(y + \triangle y)}\right) \\ &= \frac{y \triangle x - x \triangle y}{x(y + \triangle y)} = \frac{\triangle x}{x}\frac{y}{y + \triangle y} - \frac{\triangle y}{y + \triangle y} \\ &\approx \frac{\triangle x}{x} - \frac{\triangle y}{y} \end{aligned} \tag{1.25}$$

Die relativen Fehler subtrahieren sich im wesentlichen. Bei $*, \div$ tritt also keine wesentliche Verstärkung des Fehlers ein.

Beispiel 1.26 *Seien $a_1, \ldots, a_n$ Maschinenzahlen. Berechne $\sum_{i=1}^{n} a_i$ auf Rechner mit Maschinengenauigkeit* eps *. Verfolge die Fehler und deren Fortpflanzung.*

$\longrightarrow$ *Übung 1.44*

$$\left| gl\left(\sum_{i=1}^{n} a_i\right) - \sum_{i=1}^{n} a_i \right| \dot{\leqslant} \text{eps} \left(\sum_{i=1}^{n} (n-i+1)|a_i|\right)$$

Wie beurteilt man nun numerische Verfahren?

a) Genauigkeit und Verlässlichkeit des Ergebnisses.

b) Zeitbedarf, Speicherbedarf, Programmlänge, Kosten, (neuerdings Vektorisierbarkeit + Parallelisierbarkeit).

b) Hängt von der Architektur ab, bzw. von der Systemumgebung, i.a. kann man dieses gut abschätzen oder testen. Siehe auch 1.5.

a) Die Verlässlichkeit gibt an, wie ein Versagen eines Teilprogramms den Gesamtprozess beeinflusst. Bei der Genauigkeit hat man **alle möglichen auftretenden** Fehler zu berücksichtigen und ihren Einfluss auf das Ergebnis.
Fehler sind hier nicht Programmier– oder Hardwarefehler, sondern die Abweichung, eines nach richtig angewendeter Vorschrift erzeugten Resultats, vom gewünschten Ergebnis.

Es gibt verschiedene Fehlerarten:

1. Rundungs– oder Reduktionsfehler, s.o.

2. Datenfehler: Üblicherweise sind die Eingangsdaten ungenau, z. B. Messdaten mit Messfehlern. Das Lösen eines Problems entspricht dem Auswerten einer Funktion $f : D \longrightarrow W$ für Datenwerte $x \in D$. Anstelle von x hat man gestörten Wert $\tilde{x}$. Der Datenfehler ist dann $f(x) - f(\tilde{x})$. Die Empfindlichkeit der Lösung des Problems (Auswertung von f, gegenüber kleinen Änderungen in den Daten (in x) heisst Kondition des Problems. Die gute oder schlechte Kondition ist eine Eigenschaft des Problems und NICHT des Verfahrens. Später mehr.

3. Verfahrensfehler: Exakte Verfahren enden nach endlich vielen Operationen (bei exakter Rechnung) mit dem exakten Ergebnis $y = f(x)$. Näherungsverfahren (z.B. Iterationen) enden in Abhängigkeit von bestimmten Kriterien mit einer Näherung $\tilde{y}$ für die Lösung y.
Verfahrensfehler: $\tilde{y} - y$

Anwender-problem Daten x	$\longrightarrow$	Math. Problem Daten mit Unsicherheit $\tilde{x}$	$\longrightarrow$	Numer. Alg. Daten $\tilde{x}$	$\longrightarrow$	Maschinenprog. gerundete Daten $gl(\tilde{x})$
$\downarrow$		$\downarrow$		$\downarrow$		$\downarrow$
Lösung y	$\longrightarrow$ $\uparrow$	wahres Ergebnis y^*	$\longrightarrow$ $\uparrow$	Ergebnis des Alg. $\hat{y}$	$\longrightarrow$ $\uparrow$	Maschinen– Erg. $\tilde{y}$

Datenfehler Verfahrensfehler Rundungsfehler

Gegeben sei eine Funktion $f(a_1, \ldots, a_n)$ und eine durch Fehler gestörte Funktion $\tilde{f}(a_1, \ldots, a_n)$.

Fehleranalyse: Verfolgung der Auswirkung aller Fehler, die in den einzelnen Schritten bei der Berechnung von f auftreten können.

Vorwärtsanalyse: Verfolge den Fehler von Schritt zu Schritt und schätze für jedes Teilergebnis den akkumulierten Fehler ab, d.h. verfolge den Fehler, so dass das Ergebnis die Form $\tilde{f}(a_1, \ldots, a_n) = f(a_1, \ldots, a_n) + \delta$ hat, wobei δ akkumulierter Fehler ist.

Rückwärtsanalyse: Die Verfolgung des Fehlers geschieht so, dass jedes Zwischenergebnis als exakt berechnetes Ergebnis für gestörte Daten interpretiert wird, d.h. der akkumulierte Fehler im Teilergebnis wird als Datenfehlereffekt interpretiert. Also: $\tilde{f}(a_1, \ldots, a_n) = f(\tilde{a}_1, \ldots, \tilde{a}_n)$ mit gestörten Daten $\tilde{a}_1, \ldots, \tilde{a}_n$. Über den Fehler im Endergebnis erhält man so zunächst nur die Aussage, dass er einem Datenfehler bestimmter Größe entspricht (äquivalenter Datenfehler). Die Größe der äquivalenten Datenfehler dient dann zur Beurteilung der Algorithmen.

triviales Beispiel:

$$f(a_1, a_2) = a_1 + a_2, \ \tilde{f}(a_1, a_2) = a_1 \oplus a_2.$$

$$\begin{array}{lll} a_1 \oplus a_2 & \overset{\text{Vorwärtsanalyse}}{=} & \underbrace{a_1 + a_2}_{f(a_1,a_2)} + \underbrace{\varepsilon(a_1 + a_2)}_{\delta} \quad |\varepsilon| \leqslant \text{eps}\,, |\delta| \leqslant \text{eps}\,(|a_1| + |a_2|) \\ & \overset{\text{Rückwärtsanalyse}}{=} & \underbrace{a_1(1+\varepsilon) + a_2(1+\varepsilon)}_{f(\tilde{a}_1,\tilde{a_2})} \quad |\varepsilon| \leqslant \text{eps} \end{array}$$

Vorwärtsanalyse ist im allgemeinen kaum durchführbar, für die meisten Verfahren ist, wenn überhaupt, nur Rückwärtsanaylse bekannt. Ein idealer Algorithmus im Sinne der Rückwärtsanalyse hat einen äquivalenten Datenfehler von der Größenordnung der Rundungsfehler oder Eingangsfehler.
Aussagen über den Gesamtfehler erhält man über sogenannte Störungssätze.
Vorteil dieser Vorgehensweise: Auswirkungen der Arithmetik und des Verfahrens getrennt von der Kondition des Problems.

Beispiel 1.27 $x^2 - 2a_1x + a_2 = 0 \quad 0 < a_2 \ll 1 < a_1.$

Aufgabe: Finde die kleine Lösung

$$x^* = f(a_1, a_2) = a_1 - \sqrt{a_1^2 - a_2}$$

Kondition des Problems: Setze $a = (a_1, a_2)$ und betrachte Taylorentwicklung von

$$f(a + \triangle a) = \tilde{x}^*$$

$$\begin{array}{l} f(a + \triangle a) = f(a) + \frac{\partial f}{\partial a_1}(a) \triangle a_1 + \frac{\partial f}{\partial a_2}(a) \triangle a_2 \\ + \ \textit{Terme höherer Ordnung in } \triangle a_1, \triangle a_2. \end{array} \tag{1.28}$$

$$\frac{\tilde{x}^* - x^*}{x^*} = \underbrace{\frac{\partial f(a)}{\partial a_1} \frac{a_1}{x^*}}_{} \frac{\triangle a_1}{a_1} + \underbrace{\frac{\partial f(a)}{\partial a_2} \frac{a_2}{x^*}}_{} \frac{\triangle a_2}{a_2} + \ \textit{T.h.O.}$$

Verstärkungsfaktoren für
relativen Fehler in Daten.

$$\begin{aligned}
\frac{\partial f}{\partial a_1}(a) &= 1 - \frac{1}{2}\frac{2a_1}{\sqrt{a_1^2 - a_2}} = \frac{\sqrt{a_1^2 - a_2} - a_1}{\sqrt{a_1^2 - a_2}} = \frac{-x^*}{\sqrt{a_1^2 - a_2}} \\
\frac{\partial f}{\partial a_2}(a) &= \frac{1}{2}\frac{1}{\sqrt{a_1^2 - a_2}} \\
\frac{\partial f}{\partial a_1}(a)\frac{a_1}{x^*} &= \frac{-x^*}{\sqrt{a_1^2 - a_2}}\frac{a_1}{x^*} \approx -1 \quad \text{da } a_2 \ll 1 < a_1 \\
\frac{\partial f}{\partial_2}(a)\frac{a_2}{x^*} &= \frac{a_2}{2\sqrt{a_1^2 - a_2}x^*} = \frac{a_2}{\frac{2a_2}{\frac{a_1}{\sqrt{a_1^2-a_2}}+1}} \approx 1
\end{aligned}$$

Also ist das Problem gut konditioniert. Erwarten würden wir bei einem guten Algorithmus einen relativen Fehler von $c \cdot \text{eps}$*, mit kleinem* c*.*

Algorithmus:	$\mathbb{M}(10,5,1) a_1 = 6.002, a_2 = 0.01$
$y_1 := a_1 * a_1$	$y_1 = 36.024$
$y_2 := y_1 - a_2$	$y_2 = 36.014$
$y_3 := \sqrt{y_2}$	$y_3 = 6.0012$
$x := y_4 = a_1 - y_3$	$\tilde{x} = y_4 = 0.00080000$
	exakt 0.00083311
	$\frac{\triangle x}{x} = 0.039747$

Rückwärtsanalyse:

$$\begin{aligned}
& \left[a_1 - \sqrt{(a_1^2(1+\varepsilon_1) - a_2)(1+\varepsilon_2)}(1+\varepsilon_3)\right](1+\varepsilon_4),\ |\varepsilon_i| < \text{eps}\,. \\
= \quad & a_1(1+\varepsilon_4) - \sqrt{a_1^2(1+\varepsilon_4)^2 \underbrace{(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3)^2}_{\substack{1+\varepsilon_5 \\ |\varepsilon_5|<4\,\text{eps}\,.}} - a_2 \underbrace{(1+\varepsilon_2)(1+\varepsilon_3)^2(1+\varepsilon_4)^2}_{\substack{1+\varepsilon_6 \\ |\varepsilon_6|<5\,\text{eps}\,.}}} \\
= \quad & a_1(1+\varepsilon_4) - \sqrt{(a_1(1+\varepsilon_4))^2 - a_2 \underbrace{((1+\varepsilon_6) - \frac{a_1^2(1+\varepsilon_4)^2\varepsilon_5}{a_2})}_{\substack{1+\varepsilon_7 \\ |\varepsilon_7|<\text{eps}\,(5+4\frac{a_1^2}{|a_2|}))}}}
\end{aligned}$$

Der Rechner liefert die exakte Lösung von $x^2 - 2a_1(1+\varepsilon_4)x + a_2(1+\varepsilon_7) = 0$ *mit* $|\varepsilon_4| < \text{eps}$ *und* $|\varepsilon_7| < \text{eps}\,(5 + \underbrace{4\frac{a_1^2}{|a_2|}}_{\text{groß}}))$*, d.h. der Algorithmus ist ungeeignet.*

$\longrightarrow$ *Übung 1.46: besserer Alg.*

Definition 1.29 *Ein Algorithmus für den die Rückwärtsanalyse liefert, dass das berechnete Ergebnis das exakte Ergebnis eines gestörten Problems mit Störungen der Größenordnung* $c \cdot \text{eps}$ *ist, heisst numerisch rückwärts stabil. Ein Algorithmus für die Vorwärtsanalyse liefert, dass das berechnete Resultat einen relativen Fehler der Größenordnung* $c \cdot \text{eps}$ *hat,*

heisst numerisch vorwärts stabil. Ansonsten ist der Algorithmus nicht numerisch stabil. (Andere Stabilitätsbegriffe.)

! Faustregel: Gut konditioniertes Problem und stabiler Algorithmus $\Longrightarrow$ gute Ergebnisse. Schlecht konditioniertes Problem oder instabiler Algorithmus $\Longrightarrow$? Ergebnisse.

1.4 Moderne Rechnerarchitekturen

a) Klassischer von Neumann–Rechner:

Steuerwerk	Rechenwerk	Speicher	I/O Einheit
↓	↓		
Programm–arbeitung	Operationen mit Daten		

Alle vorkommenden Operationen wie Ein/Ausgabe, Speicherzugriff, arithmetische Operationen, etc. werden seriell nacheinander ausgeführt. In modernerer Form können Speicherzugriff und arithmetische Operationen gleichzeitig ausgeführt werden. Dies hat jedoch noch wenig Einfluss auf numerische Methoden.

b) Vektorrechner arbeiten im allgemeinen nach dem SIMD Prinzip. (single instruction, multiple date). In ihren Prozessoren ist ein Pipeline–Prinzip realisiert, d.h. verschiedene Phasen einer Operation können mit verschiedenen Operanden gleichzeitig durchgeführt werden.

Beispiel 1.30 *Addition zweier Dualzahlen*
$x = a \cdot 2^p, y = b \cdot 2^q, q \geqslant p$ *in normalisierter Darstellung. 4 Teilschritte*

1. *Exponentenvergleich:* $q - p$.
2. *Verschieben der Mantisse* x *um* $q - p$ *Stellen, so dass* $x = \tilde{a}2^q$.
3. *Addition der Mantissen* $\tilde{a} + b$.
4. *Normalisierte Darstellung von* $x + y$.

Bei einen seriellen Rechner werden alle 4 Operationen nacheinander ausgeführt, erst wenn Teil 4) für ein paar Operanden fertig ist, wird mit Teil 1 für das nächste Paar begonnen. In einem Vektorprozessor wird das nächste Paar in Teil 1 bearbeitet, sofort wenn ein Paar in Teil 2 ist. Sei τ *die Zeit für einen Zyklus*

$$
\begin{array}{lcl}
t = 0 & & \left[\begin{array}{c|c|c|c} x_1 & & & \\ y_1 & & & \end{array}\right] \xrightarrow{x_2,y_2} \left[\begin{array}{c|c|c|c} x_1 & & & \\ y_1 & & & \end{array}\right] \\
t = \tau & & \left[\begin{array}{c|c|c|c} & x_1 & & \\ & y_1 & & \end{array}\right] \xrightarrow{x_3,y_3} \left[\begin{array}{c|c|c|c} x_2 & x_1 & & \\ y_2 & y_1 & & \end{array}\right] \\
t = 2\tau & & \left[\begin{array}{c|c|c|c} & & x_1 & \\ & & y_1 & \end{array}\right] \xrightarrow{x_4,y_4} \left[\begin{array}{c|c|c|c} x_3 & x_2 & x_1 & \\ y_3 & y_2 & y_1 & \end{array}\right] \\
t = 3\tau & \xrightarrow{x_2,y_2} & \left[\begin{array}{c|c|c|c} & & & x_1 \\ & & & y_1 \end{array}\right] \xrightarrow{x_5,y_5} \left[\begin{array}{c|c|c|c} x_4 & x_3 & x_2 & x_1 \\ y_4 & y_3 & y_2 & y_1 \end{array}\right] \\
t = 4\tau & & \left[\begin{array}{c|c|c|c} x_2 & & & \\ y_2 & & & \end{array}\right] \xrightarrow{x_1+y_1} \left[\begin{array}{c|c|c|c} x_5 & x_4 & x_3 & x_2 \\ y_5 & y_5 & y_3 & y_2 \end{array}\right] \xrightarrow{x_6,y_6} \left[\begin{array}{c|c|c|c} x_5 & x_4 & x_3 & x_2 \\ y_5 & y_5 & y_3 & y_2 \end{array}\right] \xrightarrow{x_1+y_1} \\
t = 5\tau & & \left[\begin{array}{c|c|c|c} & x_2 & & \\ & y_2 & & \end{array}\right] \xrightarrow{x_7,y_7} \left[\begin{array}{c|c|c|c} x_6 & x_5 & x_4 & x_3 \\ y_6 & y_5 & y_4 & y_3 \end{array}\right] \xrightarrow{x_2+y_2}
\end{array}
$$

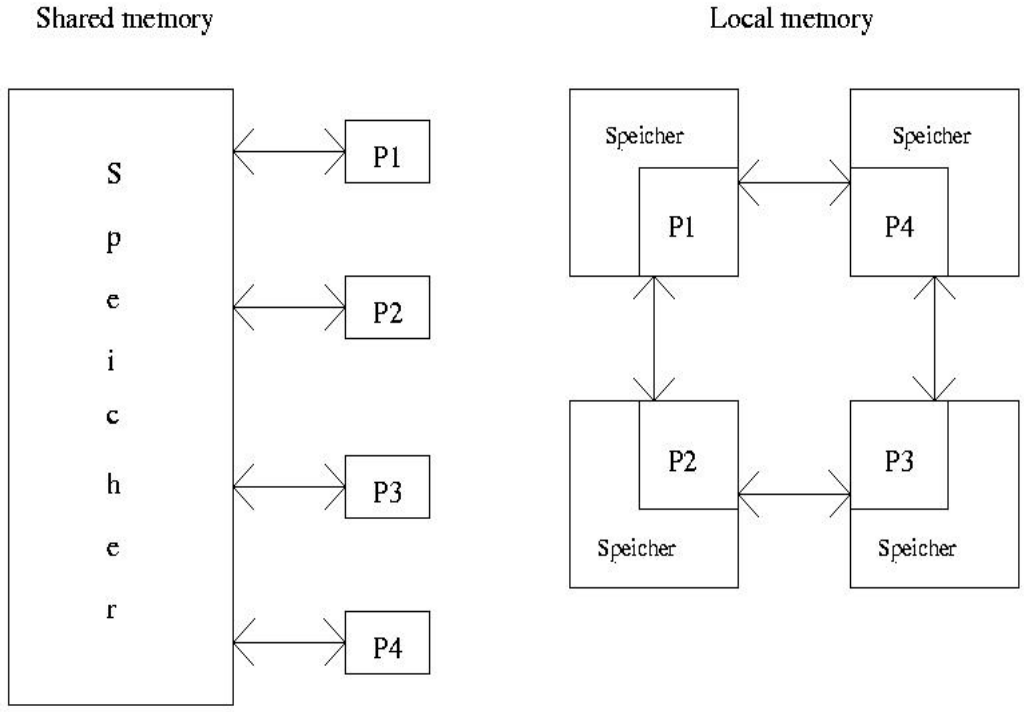
Nach Start–up Phase, je Takt ein Ergebnis, gut falls vielfach dieselben Operationen gemacht werden. Vektoradditon z.B.

Hohe Leistung beim Vektorrechner nur wenn der Datenfluss zum Prozessor gut funktioniert.

Register–Register–Maschinen
Vektoren werden in Zwischenspeichern, Vektorregistern gehalten, so lange man mit Daten aus dem aktuellen Register arbeitet sehr schnell, falls jedoch oft ein Register aus dem Hauptspeicher geladen werden muss, schlechter.

Speicher–Speicher–Maschinen
Vektorprozessor greift direkt auf Hauptspeicher. Im allgemeinen hohe Startup zeiten. Sehr gut für lange Vektoren.

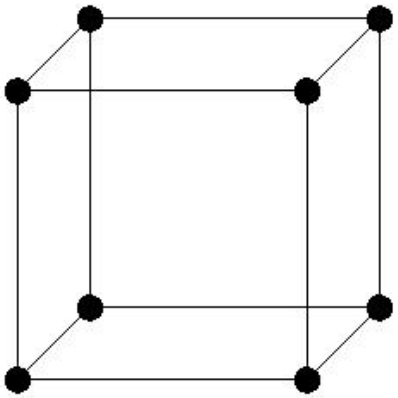
c) Parallelrechner
Mehrere Prozessoren führen Operationen gleichzeitig aus. Verschiedene Modelle.



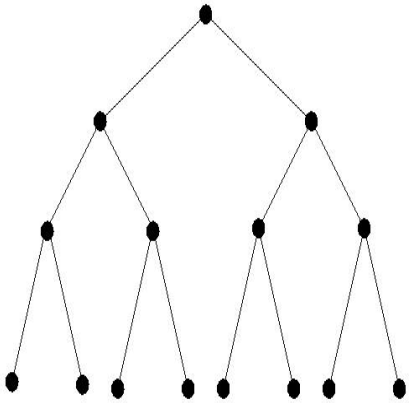
Falls ein zentraler Kontrollprozessor ein zentrales Programm abarbeitet und die Operanden dabei auf die anderen Prozessoren verteilt und alle dieselbe Operation ausführen SIMD–Rechner. Führt dagegen jeder Rechner auch sein eigenes Programm aus MIMD (multiple instruction, multiple data).
local memory bei vielen Prozessoren,
shared memory bei wenig Prozessoren.

Probleme: Verteilung der Lösung gleichmäßig
Datenkommunikation.

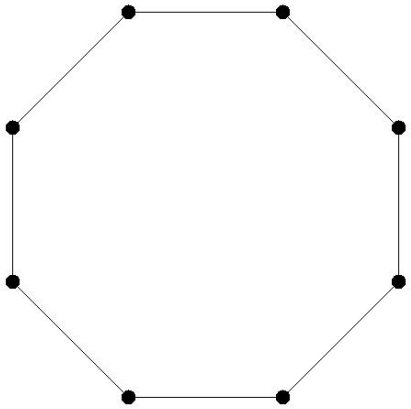
typische Architekturen: p Anzahl Prozessoren
l Anzahl links



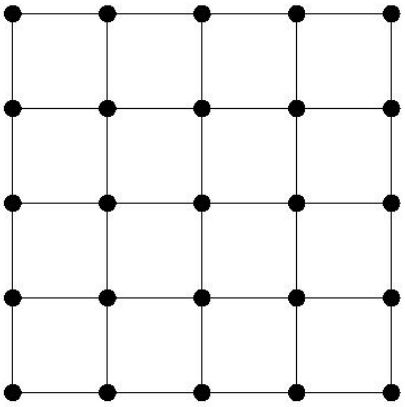
Hypercube, $p = 8, l = 3$



Binärbaum $p = 15, l \leqslant 3$



Ring, $p = 8, l = 2$



Gitter $p = 16, l \leqslant 4$

In vielen Algorithmen kann man auf Parallelität und Vektorisierbarkeit mit Rücksicht nehmen. Es ergeben sich unterschiedliche Beurteilungskriterien, je nach Rechnertyp. Es ist schwierig, allgemeine Aussagen zu machen, aber man muss diese Punkte heute immer mit berücksichtigen.

1.5 Die ':' Notation, flops und Normen

Im folgenden verwenden wir häufig die folgende Notation, die in vielen Softwareprogrammen und bei einigen Programmiersprachen verwendet wird. Für eine $m \times n$ Matrix $A = [a_{ij}] (A \in \mathbb{R}^{m,n}$ oder $A \in \mathbb{C}^{m,n})$ schreiben wir
$A(k,:) = [a_{k1}, \cdots, a_{k,n}]$,
dies ist also die k–te Zeile von A.

$$A(:,k) = \begin{bmatrix} a_{1k} \\ \vdots \\ a_{m,k} \end{bmatrix}$$

ist die k–te Spalte von A. Allgemein steht $1:n$ für $1, \ldots, n$.

Um Algorithmen vergleichen zu können werden vielfach Operationen gezählt. Ein bewährtes Maß, das oft verwendet wird ist ein sogenannter flop, d.h. eine Gleitkommaoperation $(x, -*\div)$.

Alle werden gleich gewichtet. Das war früher anderes. Früher wurden Additionen gegenüber Multiplikationen vernachlässigt oder die Kosten für die Anweisung

$$C(i,j) = C(i,j) + A(i,k) * B(k,j).$$

wurden als flop bezeichnet. Dies lässt sich bei den heutigen Rechnern nicht mehr rechtfertigen.

Normen:
Eine Vektornorm auf $\mathbb{R}^n$ ist eine Funktion $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ welche die folgenden Bedingungen erfüllt

$$\begin{array}{rcll} f(x) & \geqslant & 0 & \forall x \in \mathbb{R}^n \ (f(x) = 0 \Longleftrightarrow x = 0) \\ f(x+y) & \leqslant & f(x) + f(y) & \forall x, y \in \mathbb{R}^n \\ f(\alpha x) & = & |\alpha| f(x) & \forall \alpha \in \mathbb{R}, x \in \mathbb{R}^n \end{array} \tag{1.31}$$

Notation $f(x) = \|x\|$

p–Normen:

$$\begin{array}{rcl} \|x\|_p & = & (|x_1|^p + \cdots |x_n|^p)^{\frac{1}{p}}, \ p \geqslant 1 \\ \|x\|_1 & = & |x_1| + |x_2| + \cdots + |x_n| \\ \|x\|_2 & = & (|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}} \\ \|x\|_\infty & = & \max\limits_{1 \leqslant k \leqslant n} |x_k| \end{array}$$

Wichtige Ungleichungen:
Höldersche Ungleichung

$$|x^T y| \leqslant \|x\|_p \|y\|_q, \text{ für } \frac{1}{p} + \frac{1}{q} = 1 \tag{1.32}$$

$$|x^T y| \leqslant \|x\|_2 \|y\|_2 \quad \text{Cauchy–Schwarz} \tag{1.33}$$

$$\left.\begin{array}{rcccl} \|x\|_2 & \leqslant & \|x_1\| & \leqslant & \sqrt{n}\|x\|_2 \\ \|x\|_\infty & \leqslant & \|x\|_2 & \leqslant & \sqrt{n}\|x\|_\infty \\ \|x\|_\infty & \leqslant & \|x\|_1 & \leqslant & n\|x\|_\infty \end{array}\right\} \text{ Normäquivalenz} \tag{1.34}$$

$f : \mathbb{R}^{m,n} \longrightarrow \mathbb{R}$ ist eine Matrix Norm, falls

$$\begin{array}{rcll} f(A) & \geqslant & 0 & \forall A \in \mathbb{R}^{m,n} \ (f(A) = 0 \Longleftrightarrow A = 0) \\ f(A+B) & \leqslant & f(A) + f(B) & \forall A, B \in \mathbb{R}^{m,n} \\ f(\alpha A) & = & |\alpha| f(A) & \forall \alpha \in \mathbb{R}, A \in \mathbb{R}^{m,n} \end{array} \tag{1.35}$$

Frobenius Norm:

$$\begin{array}{l} \|A\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2\right)^{\frac{1}{2}} \\ (\|\bullet\|_2 - \text{Norm auf } \mathbb{R}^{mn}) \end{array} \tag{1.36}$$

$\longrightarrow$ Übung 1.47.

p–Normen

$$\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \sup_{x \neq 0} \left\| A \frac{x}{\|x\|_p} \right\|_p = \max_{\|x\|_p = 1} \|Ax\|_p \tag{1.37}$$

$\longrightarrow$ Übung 1.50.

Die Matrix Norm ist abhängig von der Dimension. $\|\bullet\|_2$ auf $\mathbb{R}^{3\times 2}$ ist etwas anderes als $\|\bullet\|_2$ auf $\mathbb{R}^{5,6}$.

Konsistente Matrixnorm

$$\begin{array}{rcl} \|AB\|_p & \leqslant & \|A\|_p\|B\|_p \\ \|AB\|_F & \leqslant & \|A\|_2\|B\|_F \leqslant \|A\|_F\|B\|_F \end{array} \qquad \forall A \in \mathbb{R}^{m,n}, B \in \mathbb{R}^{n,q} \tag{1.38}$$

$\longrightarrow$ Übung 1.47.

Nicht alle Normen erfüllen die Konsistenzbedingung.

Ungleichungen und Gleichungen

$$\left.\begin{array}{rcl} \|A\|_2 & \leqslant & \|A\|_F \leqslant \sqrt{n}\|A\|_2 \\ \max\limits_{i,j}|a_{ij}| & \leqslant & \|A\|_2 \leqslant \sqrt{mn}\max\limits_{i,j}|a_{ij}| \\ \|A\|_1 & = & \max\limits_{1\leqslant j\leqslant n}\sum_{i=1}^m |a_{ij}| \\ \|A\|_\infty & = & \max\limits_{1\leqslant i\leqslant m}\sum_{j=1}^n |a_{ij}| \\ \frac{1}{\sqrt{n}}\|A\|_\infty & \leqslant & \|A\|_2 \leqslant \sqrt{m}\|A\|_\infty \\ \frac{1}{\sqrt{m}}\|A\|_1 & \leqslant & \|A\|_2 \leqslant \sqrt{n}\|A\|_1 \end{array}\right\} \tag{1.39}$$

Die Matrix 2–Norm ist nicht so leicht zu charakterisieren wie $\| \bullet \|_1, \| \bullet \|_\infty$.
$\longrightarrow$ Übung 1.48

Satz 1.40 *Sei $A \in \mathbb{R}^{m,n}$. Dann gibt es einen Vektor $z \in \mathbb{R}^n$ mit $\|z\|_2 = 1$, so dass $A^TAz = \mu^2 z$, wobei $\mu = \|A\|_2$, d.h. $\|A\|_2^2$ ist ein Eigenwert von A^TA.*

Beweis: Sei $z \in \mathbb{R}^n, \|z\|_2 = 1$ und $\|Az\|_2 = \|A\|_2$, dann maximiert z die Funktion

$$g(x) = \frac{1}{2}\frac{\|Ax\|_2^2}{\|x\|_2^2} = \frac{1}{2}\frac{x^TA^TAx}{x^Tx}$$

also gilt $\bigtriangledown g(z) = 0$ ($\bigtriangledown g$ Gradient von g)

$$\begin{array}{l} \left.\frac{\partial g(x)}{\partial x_i}\right|_{x=z} = \left[(z^Tz)\sum_{j=1}^n (A^TA)_{ij}z_j - (z^TA^TAz)z\right]/(z^Tz)^2 \\ \Longrightarrow A^TAz = (\underbrace{z^TA^TAz}_{\mu^2})z = \mu^2 z. \end{array}$$

□

Insbesondere gilt, dass $\|A\|_2^2$ der größte Eigenwert von A^TA ist.

Norminvarianz

$$\begin{array}{rcl} \|UAV\|_2 & = & \|A\|_2 \\ \|UAV\|_F & = & \|A\|_F \end{array} \text{ , } U \in \mathbb{R}^{m,m}, V \in \mathbb{R}^{n,n} \text{ orthogonal, } A \in \mathbb{R}^{m,n}. \tag{1.41}$$

Absolutbeträge für Matrizen

$A \in \mathbb{R}^{m,n}$, dann ist $|A| = B$ mit $b_{ij} = |a_{ij}|$. Wir setzen

$$B \leqslant A, \text{ falls } b_{ij} \leqslant a_{ij}, \ \forall i = 1:m, j = 1:n.$$

Für das Rechnen mit Beträgen gelten folgende Regeln

$$\left.\begin{array}{rcl} |A+B| & \leqslant & |A| + |B| \\ |AB| & \leqslant & |A|\,|B| \\ A & \leqslant & B.\ C, D \geqslant O \Longrightarrow CAD \leqslant CBD \\ \|A\|_p & \leqslant & \|\,|A|\,\|_p \\ \|A\| & = & \|\,|A|\,\|, \text{ für } \| \bullet \| = \| \bullet \|_1, \| \bullet \|_\infty, \| \bullet \|_F \\ |A| & \leqslant & |B| \Longrightarrow \|A\|_1 \leqslant \|B\|_1, \|A\|_\infty \leqslant \|B\|_\infty, \|A\|_F \leqslant \|B\|_F \end{array}\right\} \tag{1.42}$$

$\longrightarrow$ Übung 1.49.

Wir haben gesehen, dass sich beim Speichern oder Runden ergibt

$$[gl(A)]_{ij} = [gl(a_{ij})] = [a_{ij}(1+\varepsilon_{ij})] \text{ mit } |\varepsilon_{ij}| \leqslant \text{eps}$$

Dann folgt

$$|gl(A) - A| \leqslant \text{eps}\,|A|.$$

Dieses kann auch als Normungleichung umgeschrieben werden, $\|gl(A) - A\|_1 \leqslant \text{eps}\,\|A\|_1$. Heute werden vielfach für Fehleranalysen nicht Normabschätzungen sondern elementweise Abschätzungen (LAPACK) verwendet.

1.6 Übungen zu Kapitel 1

Übung 1.43 *Stelle folgende Dezimalzahlen im Dual–, Hexadezimalsystem sowie im Zwölfersystem dar (d.h. $p = 2, 16, 12$, für $p = 12$ verwende man A, B als Ziffern für $10, 11$)*

$$275, 0.1, 1/3, 12.125$$

Übung 1.44 *$a_1, \ldots, a_n$ seien Maschinenzahlen und es sei $f(a_1, \ldots, a_n) = \sum_{i=1}^n a_i$. Der Computer habe die Maschinengenauigkeit* eps *. Auf dem Computer wird f in der Form $\tilde{f}(a_1, \ldots, a_n) = (\ldots((a_1 \oplus a_2) \oplus a_3) \oplus \ldots) \oplus a_n$ berechnet.*

- *Mache eine Rückwärtsanalyse für f. D.h. $\tilde{f}(a_1, \ldots, a_n) = f(a_1(1+\delta_1), \ldots, a_n(1+\delta_n))$ und $|\delta_1|, \ldots, |\delta_n|$ sind geeignet abzuschätzen.*
- *Mache eine Vorwärtsanalyse für f. D.h. $\tilde{f}(a_1, \ldots, a_n) = f(a_1, \ldots, a_n) + \varepsilon$ und $|\varepsilon|$ ist geeignet abzuschätzen.*
- *Zeige, daß für den Fehler zwischen $\tilde{f}$ und f gilt:*

$$|\tilde{f}(a_1, \ldots, a_n) - f(a_1, \ldots, a_n)| \dot{\leqslant} \text{eps} \sum_{i=1}^n |n - i + 1|\,|a_i|.$$

Terme in der Größenordung $\text{eps}^2, \text{eps}^3, \ldots$ *dürfen als näherungsweise* 0 *vernachlässigt werden.*

Übung 1.45 *Ermittle in* **MATLAB** *die Realisierung von underflow/overflow sowie* $x_{\min}, x_{\max}$, eps .

Übung 1.46 *Gib für das Beispiel aus der Vorlesung zur Berechnung von $x^* = f(a_1, a_2) = a_1 - \sqrt{a_1^2 - a_2}$ einen besseren Algorithmus an. Begründung!*

Übung 1.47 *Sei $A \in \mathbb{R}^{m,n}$. Zeige:*

1. $\|A\|_2 \leqslant \|A\|_F \leqslant \sqrt{n}\|A\|_2$
2. $\|AB\|_F \leqslant \|A\|_2\,\|B\|_F \leqslant \|A\|_F\,\|B\|_F$.

Übung 1.48 *Sei $A \in \mathbb{R}^{m,n}$. Zeige:*

1. *Ist $B \in \mathbb{R}^{n,l}$ so gilt $\|AB\|_p \leqslant \|A\|_p \|B\|_p$*
   Tipp: Zeige die Behauptung zuerst für $l = 1$ und nutzte die Def.

2. $\|A\|_1 = \max\limits_{1 \leqslant j \leqslant n} \sum\limits_{i=1}^{m} |a_{ij}|$.
   Tipp: Zeige zunächst $\|Ax\|_1 \leqslant (\max \Sigma \cdots)\|x\|_1$ *und konstruiere dann ein* x *für welches Gleichheit gilt.*
3. $\|A\|_\infty = \max\limits_{1 \leqslant i \leqslant m} \sum\limits_{j=1}^{n} |a_{ij}|$.
4. *Ist* $\rho(A) = \max\{|\lambda| : \lambda$ *Eigenwert von* $A\}$ *eine Matrixnorm?*

Übung 1.49 *Sei* $A \in \mathbb{R}^{m,n}, B \in \mathbb{R}^{k,l}$. *Zeige:*

1. *Falls* $k = m, l = n$, *dann gilt* $|A + B| \leqslant |A| + |B|$.
2. *Falls* $k = n$, *dann gilt* $|AB| \leqslant |A|\,|B|$.
3. *Falls* $k = m, l = n$ *und* $A \leqslant B$, *dann gilt für alle* $C \in \mathbb{R}^{p,m}$ *mit* $C \geqslant O$: $CA \leqslant CB$.
4. *Falls* $k = m, l = n$ *und* $|A| \leqslant |B|$, *dann gilt* $\|A\|_\infty \leqslant \|B\|_\infty$.

Übung 1.50 *Für eine Matrix* $A \in \mathbb{R}^{m,n}$ *ist die Matrix* p*–Norm definiert durch*

$$\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p = 1} \|Ax\|_p$$

1. $\|A\|_p$ *ist eine Norm*
2. $\|Ax\|_p \leqslant \|A\|_p \|x\|_p$
3. *Ist* c *eine Konstante, so dass* $\|Ax\|_p \leqslant c\|x\|_p$ *für alle* x *gilt, so ist* $c \geqslant \|A\|_p$, *d.h.* $c = \|A\|_p$ *ist die kleinst mögliche Konstante, so dass* $\|Ax\|_p \leqslant c\|x\|_p$ *gilt.*

Kapitel 2

Lösung linearer Gleichungssysteme

Problem: Berechne Lösung von $AX = B$ wobei $A \in \mathrm{GL}_n(\mathbb{C}) = \mathbb{C}^{n,n}$ (oder $\mathrm{GL}_n(\mathbb{R}) = \mathbb{R}^{n,n}$) und $B \in \mathbb{C}^{n,m}$ (oder $\mathbb{R}^{m,n}$) ist. Im folgenden $\mathbb{K} = \mathbb{C}$ oder $\mathbb{K} = \mathbb{R}$. Wir betrachten zuerst direkte Verfahren, d.h. Verfahren die bei exakter Rechnung in endlich vielen Schritten enden.

2.1 Gauß–Elimination

Der immer noch in vielen Variationen am weitesten bekannte Algorithmus ist die Gauß–Elimination.

2.1.1 Lösung von Dreieckssystemen

Zur Vorbereitung betrachten wir zuerst einmal spezielle Dreiecksmatrizen und nur eine rechte Seite. Sei nun $L = [l_{ij}] \in \mathbb{K}^{n,n}$ untere $\triangle$–Matrix. Wir betrachten die Lösung von $Lx = b = [b_i]$ mit $b \in \mathbb{K}^n$. Wir erhalten sofort als Lösung

$$x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right) / l_{ii} \quad i = 1, \cdots, n \tag{2.1}$$

Dieses Verfahren heißt Vorwärts–Einsetzen und wird durch folgenden Algorithmus realisiert.

Algorithmus 2.2
Input: $L \in \mathbb{K}^{n,n}$ *untere* $\triangle$*–Matrix, nichtsingulär,* $b \in \mathbb{K}^n$.
Output: *Lösung von* $Lx = b$, *überschrieben auf* b.

$b(1) = b(1)/L(1,1)$;
FOR $i = 2 : n$
$\quad b(i) = (b(i) - L(i, 1 : i-1) * b(1 : i-1))/L(i,i)$;
END

! Achtung die Multiplikation $L(i, 1 : i-1) * b(1 : i-1)$ ist natürlich eine Schleife.
$\longrightarrow$ Übung 2.58
Kosten für Algorithmus 2.2: n^2 flops

Fehleranalyse für Algorithmus 2.2: Die berechnete Lösung von $\tilde{x}$ erfüllt

$$(L+F)\tilde{x} = b, \text{ wobei } |F| \leqslant n \cdot \text{ eps } \cdot |L| + \mathcal{O}(\text{ eps }^2). \tag{2.3}$$

Der analoge Algorithmus für obere Dreiecksmatrizen heißt Rückwärts–Einsetzen. Sei $U \in [u_{ij}] \in \mathbb{K}^{n,n}$ obere Dreiecksmatrix. Für die Lösung von $Ux = b = [b_i] \in \mathbb{K}^{n,n}$ erhalten wir

$$x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{jj} \quad i = n, n-1, \ldots, 1 \tag{2.4}$$

durch den folgenden Algorithmus.

Algorithmus 2.5
Input: $U \in \mathbb{K}^{n,n}$ *nichtsinguläre obere* $\triangle$*–Matrix* $b \in \mathbb{K}^n$.
Output: *Lösung von* $Ux = b$, *überschrieben auf* b.

$b(n) = b(n)/U(n,n)$;
FOR $i = n-1 : -1 : 1$
$\quad b(i) = (b(i) - U(i, i+1:n) * b(i+1:n))/U(i,i)$;
END

Es gilt wieder
Kosten Algorithmus 2.5: n^2 flops
Fehleranalyse Algorithmus 2.5: $\tilde{x}$ erfüllt

$$(U+F)\tilde{x} = b, \text{ wobei } |F| \leqslant n \cdot \text{ eps } \cdot |U| + \mathcal{O}(\text{ eps }^2). \tag{2.6}$$

$\longrightarrow$ Übung 2.58

Es gibt Varianten dieser Algorithmen für Parallel– und Vektorrechner (spaltenorientierte Versionen) und entsprechende Block–Versionen für mehrfache rechte Seiten. Die Algorithmen zum Vorwärts– und Rückwärts–einsetzen bilden die Basis für die Lösung von Gleichungssystemen mittels LR–Zerlegung.

2.1.2 LR–Zerlegung

Wir kommen nun zur LR–Zerlegung einer Matrix A als $A = LR$, mit L untere und R obere $\triangle$–Matrix. Falls man so eine Zerlegung hat, so löst man $Ax = b$ mittels Algorithmus 2.5 und 2.2 indem man $Ly = b, Rx = y$ nacheinander löst. Die LR–Zerlegung wird mittels des Gaußschen Eliminationsverfahrens erzeugt. Dazu verwenden wir sogenannte Gauß–Transformationen.
Sei $x \in \mathbb{K}^n$ mit $x_k \neq 0$. Sei

$$t \equiv t^{(k)} = \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ t_{k+1} \\ \vdots \\ t_n, \end{array} \right] \Bigg\} k \quad , \; t_i = \frac{x_i}{x_k}, i = k+1 : n.$$

Setze

$$M_k := I - t^{(k)} e_k^T \text{ (Dyade, äußeres Produkt, } e_k \text{ } k\text{–ter Einheitsvektor)}, \tag{2.7}$$

dann gilt

$$M_k x = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -t_{k+1} & 1 & & \\ & & \vdots & & \ddots & \\ & & -t_n & & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.8}$$

M_k heißt Gauß–Transformation.

Algorithmus 2.9

Input: $x \in \mathbb{K}^n, x_1 \neq 0$.

Output: *Vektor t der Länge $n-1$, so daß für die Gauß–Transformation M mit $M(2:n,1) = -t$ und $y = Mx$ gilt: $y(2:n) = 0$.*

```
FUNCTION    t = GAUSS(x)
            n = length(x);
            t = x(2:n)/x(1);
END GAUSS
```

Dieser Algorithmus benötigt $n-1$ flops. Multiplikation mit einer Gauß–Transformation

$$M_k C = (I - t^{(k)} e_k^T) C = C - t^{(k)} (e_k^T C). \tag{2.10}$$

Da $t(1:k) = 0$ wird nur $C(k+1:n,:)$ verändert. Die Multiplikation wird durch folgenden Algorithmus durchgeführt:

Algorithmus 2.11

Input: $C \in \mathbb{K}^{n,r}, M \in \mathbb{K}^{n,n}$ *Gauß–Transformation mit* $M(2:n,1) = -t$.

Output: *C überschrieben mit MC.*

```
FUNCTION        C = GAUSSAPP(C, t)
                n = size(C, 1);
                C(2:n,:) = C(2:n,:) − t ∗ C(1,:);
END GAUSSAPP
```

Dieser Algorithmus braucht $2(n-1)r$ flops.

Fehleranalyse für 2.11: Der berechnete Wert $\tilde{t}$ für t aus *GAUSS* erfüllt

$$\tilde{t} = t + e, \text{ wobei } |e| \leqslant \text{eps}\,|t|. \tag{2.12}$$

Damit erhält man für das Ergebnis von *GAUSSAPP*

$$gl\left((I - \tilde{t} e_1^T) C\right) = (I - t e_1^T) C + E, \tag{2.13}$$

wobei

$$|E| \leqslant 3\ \text{eps}\,(|C| + |t|\,|C(1,:)|) + \mathcal{O}(\text{eps}^2) \tag{2.14}$$

! Es ist offensichtlich, falls $|t|$ groß ist, dann werden die Fehler in der Aufdatierung sehr groß gegenüber $|C|$.

$\longrightarrow$ Übung 2.62
Mehrfache Anwendung von *GAUSSAPP* führt dann auf Dreiecksgestalt.
Im k–ten Schritt erhalten wir (falls alles glatt geht)

$$A^{(k-1)} := M_{k-1}\cdots M_2 M_1 A = \begin{bmatrix} a_{11}^{(k-1)} & \cdots & a_{1,k-1}^{(k-1)} & \cdots & \cdots & a_{1,n}^{(k-1)} \\ 0 & \ddots & & & & \vdots \\ \vdots & \ddots & a_{k-1,k-1}^{(k-1)} & \cdots & \cdots & a_{k-1,n}^{(k-1)} \\ \vdots & & 0 & a_{kk}^{(k-1)} & & a_{k,n}^{(k-1)} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & a_{n,k}^{(k-1)} & \cdots & a_{nn}^{(k-1)} \end{bmatrix}. \tag{2.15}$$

Wir fahren dann fort auf dem noch nicht reduzierten unteren Block. Die vollständige Dreiecksreduktion wird dann durch die folgende Schleife erzielt.

Algorithmus 2.16
$n = size(A, 1);$
$k = 1;$
WHILE $A(k,k) \neq 0$ *AND* $k < n$
$t = GAUSS(A(k:n, k));$
$A(k:n,:) = GAUSSAPP(A(k:n,:), t);$
$k = k + 1;$
END

! Die Elemente $A(k,k)$, die während des Algorithmus auf '0' überprüft werden müssen, heißen Pivots. Ihre relative Größe ist von größter Bedeutung für die Fehleranalyse.

Matrizentheoretisch formuliert gilt, wenn Algorithmus 2.16 mit $k = n$ endet, daß

$$M_{n-1}\cdots M_1 A =: R$$

mit R obere $\triangle$–Matrix ist.
Für jedes $M_k = I - t^{(k)} e_k^T$ gilt $M_k^{-1} = I + t^{(k)} e_k^T$, also folgt:
$A = M_1^{-1}\cdots M_{n-1}^{-1} R =: L \cdot R$. Alle M_k sind untere $\triangle$–Matrizen mit 1–Diagonale also auch L.

Eine Zerlegung $A = L \cdot R$ mit R obere $\triangle$–Matrix und L untere $\triangle$–Matrix mit 1–Diagonale heißt LR–Zerlegung (engl. LU–Decomposition).

Satz 2.17 (Existenz und Eindeutigkeit der LR–Zerlegung)
$A \in \mathbb{K}^{n,n}$ hat eine LR–Zerlegung genau dann, wenn

$$\det\left(A(1:k, 1:k)\right) \neq 0, \quad \textit{für } k = 1 : n-1. \tag{2.18}$$

Falls die LR–Zerlegung existiert und A nichtsingulär ist, so ist die LR–Zerlegung eindeutig und $\det A = r_{11}\cdots r_{nn}$.

Beweis: Falls A eine LR–Zerlegung

$$A = \begin{bmatrix} 1 & & & \\ l_{21} & \ddots & & \\ \vdots & \ddots & \ddots & \\ l_{n1} & \cdots & l_{n,n+1} & 1 \end{bmatrix} \begin{bmatrix} r_{11} & \cdots & \cdots & r_{1n} \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ & & & r_{n,n} \end{bmatrix}$$

besitzt, so gilt: alle $r_{ii},\ i = 1 : n-1$ sind $\neq 0$, da sie die Pivots in Algorithmus 2.16 sind.

$$\det\left(A(1:k,1:k)\right) = \det\left(L(1:k,1:k)\right)\cdot\det\left(R(1:k,1:k)\right) = 1\cdot\prod_{i=1}^{k} r_{ii} \neq 0.$$

Die Rückrichtung machen wir mit Induktion über k.
$k = 1$ ist klar. Angenommen $k-1$ Schritte sind ausgeführt. $A^{(k-1)} = M_{k-1}\cdots M_1 A$ und $a_{k,k}^{(k-1)}$ ist das k–te Pivot.

$$\underbrace{M_{k-1}\cdots M_1}_{M^{(k-1)}} A = \left[\begin{array}{ccc|ccc} 1 & & & & & \\ * & \ddots & & & & \\ \vdots & \ddots & 1 & & & \\ \hline \vdots & & * & 1 & & \\ \vdots & & \vdots & & \ddots & \\ * & \cdots & * & & & 1 \end{array}\right] A = A^{(k-1)} = \left[\begin{array}{ccc|ccc} a_{11}^{(k-1)} & \cdots & \cdots & \cdots & \cdots & * \\ & \ddots & & & & \vdots \\ & & a_{k-1,k-1}^{(k-1)} & \cdots & \cdots & * \\ \hline & & & a_{kk}^{(k-1)} & \cdots & * \\ & & & \vdots & & \vdots \\ & & & * & \cdots & * \end{array}\right].$$

Es folgt, daß

$$\begin{aligned} \det\left(A^{(k-1)}(1:k,1:k)\right) &= \prod_{i=1}^{k} a_{ii}^{(k-1)} \\ &= \underbrace{\det\left(M^{(k-1)}(1:k,1:k),\right)}_{=1}\cdot\det\left(A(1:k,1:k)\right). \end{aligned}$$

Also folgt aus $\det\left(A(1:k,1:k)\right) \neq 0$, daß $a_{k,k}^{(k-1)} \neq 0$ ist.

Zur Eindeutigkeit: Seien $A = L_1R_1 = L_2R_2$ zwei LR–Zerlegungen von A, A nichtsingulär, dann sind $L_i, R_i i = 1,2$ auch nichtsingulär. Also folgt

$$\underbrace{\begin{array}{ccc} L_2^{-1}L_1 & = & R_2R_1^{-1} \\ \left[\begin{smallmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{smallmatrix}\right] & = & \left[\begin{smallmatrix} \ddots & & \\ & \ddots & \\ & & \ddots \end{smallmatrix}\right] \end{array}}_{=I}$$

$\Longrightarrow L_1 = L_2, R_1 = R_2$.

$$\det A = \det L\cdot\det R = 1\cdot\det R = r_{11}\cdots r_{nn}.$$

□

! Die LR–Zerlegung läßt sich auch über anderen Körpern (Ringen) durchführen.

Einige praktische Details.

- Gauß–Transformation braucht nur auf Spalten $k : n$ angewendet werden, und selbst bei Spalte k kennen wir das Ergebnis. Also Änderung von Algorithmus 2.16

$$A(k:n,k+1:n) = \text{ GAUSSAPP}\left(A(k:n,k+1:n),t\right);$$

- Die Multiplikatoren, d.h. die wichtigen Elemente von t_k können auf den entstandenen Nullen von A gespeichert werden.

Damit erhält folgenden Algorithmus für die LR–Zerlegung.

Algorithmus 2.19 (Äußere–Produkt–Form der Gauß–Elimination)

Input: $A \in \mathbb{K}^{n,n}$ *mit* $A(1:k,1:k)$ *nichtsingulär* $k = 1:n-1$.

Output: *Faktorisierung* $M_{n-1}\cdots M_1 A = R$, *mit* R *obere* $\triangle$*–Matrix und* M_i *Gauß–Transformationen,* R *wird im oberen* $\triangle$ *von* A *und die Multiplikatoren aus* M_k *in* $A(k+1:n,k)$, *d.h.* $A(k+1:n,k) = -M_k(k+1:n,k)$.

```
FOR   k = 1 : n − 1
      t = GAUSS(A(k : n, k));
      A(k + 1 : n, k) = t;
      A(k : n, k + 1 : n) = GAUSSAPP(A(k : n, k + 1 : n), t);
END
```

Kosten $\frac{2n^3}{3}$ flops, jeder Durchgang durch die k–Schleife ist ein äußeres Produkt. Wir haben gesehen, daß wir die Multiplikatoren aus $t^{(k)}$ in A speichern können. Wie erhalten wir nun L (falls wir es benötigen)?
$\longrightarrow$ Übung 2.68.

$$\begin{aligned} L &= (M_{n-1}\cdots M_1)^{-1} = M_1^{-1}\cdots M_{n-1}^{-1} \\ &= (I + t^{(1)}e_1^T)\cdots(I + t^{(n-1)}e_{n-1}^T) = I + \sum_{k=1}^{n-1} t^{(k)}e_k^T. \end{aligned}$$

Also folgt $L(k+1:n,k) = t^{(k)}$. Die Lösung eines linearen Gleichungssystems folgt nun basierend auf der LR–Zerlegung wie oben beschrieben.

! Im wesentlichen enthält Algorithmus 2.19 3 Schleifen die ineinandergeschachtelt sind. Je nach Rechnerarchitektur ist es eine andere als die beschriebene Anordnung. Dies kann zu erheblichen Einsparungen führen. Als Beispiel sei hier die sogenannte $GAXPY$ LR–Zerlegung betrachtet:

$$\begin{array}{c} GAXPY \qquad (y = Ax + y \quad y \in R^m, A \in \mathbb{R}^{m,n}, x \in \mathbb{R}^n) \\ (\text{general } Ax \text{ plus } y) \end{array}$$

(Dies ist eine Routine von den Basic Linear Algebra Subroutines(BLAS), die auf allen Standardrechnern in sehr guter Weise implementiert sind, vektor/parallel und auf die man seine Codes aufbauen sollte.)

Betrachte die Schleifen von Algorithmus 2.19 ausgeschrieben.

```
FOR   k = 1 : n − 1
      A(k + 1 : n, k) = A(k + 1 : n, k)/A(k, k);
      FOR   j = k + 1 : n
            FOR   i = k + 1 : n
                  A(i, j) = A(i, j) − A(i, k) ∗ A(k, j);
            END
      END
END
```

In der veränderten Variante wird die Transformation mit M_k nicht direkt ausgeführt wie in Algorithmus 2.19 sondern $A(:,j)$ wird erst im Schritt j mit $M_{j-1}\cdots M_1 A(:,j)$ überschrieben und dann M_j berechnet. Genauer, sei $1 \leqslant j \leqslant n-1$ und $L(:,1:j-1), R(1:j-1,1:j-1)$

seien bekannt. Um die j–ten Spalten von L, R zu erhalten, setzen wir die j–ten Spalten von $A = LR$ gleich

$$\begin{aligned} A(1:j-1,j) &= L(1:j-1,1:j-1)R(1:j-1,j), \\ A(j:n,j) &= \sum_{k=1}^{j} L(j:n,k)R(k,j). \end{aligned}$$

Die erste Gleichung ist ein Gleichungssystem mit unterer $\triangle$–Matrix, daß gelöst werden muß. Die zweite Gleichung schreiben wir wie folgt.

$$\begin{aligned} v(j:n) &= A(j:n,j) - \sum_{k=1}^{j-1} L(j:n,k)R(k,j) \\ &= A(j:n,j) - L(j:n,1:j-1)R(1:j-1,j). \end{aligned}$$

Dann gilt $R(j,j) = v(j)$ und $L(j+1:n,j) = v(j+1:n)/R(j,j)$.
Also ist $L(j+1:n,j)$ nichts anderes als eine skalierte GAXPY Operation und wir erhalten:

FOR $j = 1:n$
Solve $L(1:j-1,1:j-1)R(1:j-1,j) = A(1:j-1,j)$;
$v(j:n) = A(j:n,j) - L(j:n,1:j-1)R(1:j-1,j)$
$R(j,j) = v(j)$;
$L(j+1:n,j) = v(j+1:n)/R(j,j)$;
END

Zusammengefaßt erhält man dann

Algorithmus 2.20 (GAXPY Form der Gauß–Elimination)
Input: $A \in \mathbb{K}^{n,n}$ *mit* $A(1:k,1:k)$ *nichtsingulär* $k = 1:n-1$
Output: *Faktorisierung* $A = LR$ *mit* L *untere* $\triangle$*–Matrix mit 1–Diagonale,* R *obere* $\triangle$*–Matrix. Falls* $i > j$*, so ist* $L(i,j)$ *in* $A(i,j)$ *gespeichert. Falls* $i \leqslant j$*, so ist* $R(i,j)$ *in* $A(i,j)$ *gespeichert.*

```
FOR  j = 1 : n
     % Solve L(1 : j − 1, 1 : j − 1)R(1 : j − 1, j) = A(1 : j − 1, j)
     FOR  k = 1 : j − 1
          FOR  i = k + 1 : j − 1
               A(i, j) = A(i, j) − A(i, k) ∗ A(k, j);
          END
     END
     % v(j : n) = A(j : n, j) − L(j : n, 1 : j − 1) ∗ R(1 : j − 1, j)
     FOR  k = 1 : j − 1
          FOR  i = j : n
               A(i, j) = A(i, j) − A(i, k) ∗ A(k, j);
          END
     END
     % L(j + 1 : n, j) = v(j + 1 : n)/v(j)
     A(j + 1 : n, j) = A(j + 1 : n, j)/A(j, j);
END
```

Kosten $\frac{2n^3}{3}$ flops.
Weitere Varianten (andere Schleifenorientierung) Crout, Prolittle, Blockvarianten, siehe Buch von Golub/Van Loan: 'Matrix Computations'.

2.1.3 Fehleranalyse der Gauß-Elimination

Anwendung der dargestellten Algorithmen zur Lösung von $Ax = b$. Fehleranalyse i.a. sehr schwierig, bei Gauß sehr viel bekannt, für andere Algorithmen teils sehr wenig.
Zuerst müssen wir nun eine Störungsanalyse für Gleichungssysteme machen. Betrachte das parametrisierte System

$$(A + \varepsilon F)x(\varepsilon) = b + \varepsilon f,\ x(0) = x,\ F \in \mathbb{R}^{n,n}, f \in \mathbb{R}^n.$$

Falls A nichtsingulär ist, so ist $x(\varepsilon)$ differenzierbar in einer Umgebung von 0 und es gilt:

$$\dot{x}(0) = A^{-1}(f - Fx).$$

Taylorentwicklung liefert

$$x(\varepsilon) = x + \varepsilon\dot{x}(0) + \mathcal{O}(\varepsilon^2).$$

Also folgt für jede Vektornorm und konsistente Matrixnorm

$$\frac{\|x(\varepsilon) - x\|}{\|x\|} \leqslant |\varepsilon|\|A^{-1}\| \left\{ \frac{\|f\|}{\|x\|} + \|F\| \right\} + \mathcal{O}(\varepsilon^2). \tag{2.21}$$

Für quadratische Matrizen definiere die Kondition $\kappa_{\|\bullet\|}(A) := \|A\|\|A^{-1}\|$, diese ist abhängig von der Norm, $\kappa(A) = \infty$ für A singulär. Mit der Konsistenzungleichung $\|b\| \leqslant \|A\|\|x\|$ folgt

$$\frac{\|x(\varepsilon) - x\|}{\|x\|} \leqslant \kappa(A)(\rho_A + \rho_b) + \mathcal{O}(\varepsilon^2), \tag{2.22}$$

wobei

$$\rho_A = \varepsilon\frac{\|F\|}{\|A\|}, \rho_b = \varepsilon\frac{\|f\|}{\|b\|}$$

die relativen Fehler in A, b sind.
Die Abschätzung (2.22) ist noch unbefriedigend, da sie auf „ε klein“ beruht, eine genaue Analyse erhält man mit den folgenden Resultaten.

Lemma 2.23 *Sei $F \in \mathbb{R}^{n,n}$ und $\| \bullet \|$ eine konsistente Norm. Falls $\|F\| < 1$, so ist $I - F$ nicht singulär und es gilt*

$$(I - F)^{-1} = \sum_{k=0}^{\infty} F^k \qquad \textit{Neumannsche Reihe} \tag{2.24}$$

und

$$\|(I - F)^{-1}\| \leqslant \frac{1}{1 - \|F\|}. \tag{2.25}$$

Beweis: Angenommen $I - F$ singulär $\Longrightarrow \exists x \neq 0$ mit

$$(I - F)x = 0 \Longrightarrow \|x\| = \|Fx\| \leqslant \|F\|\|x\| \Longrightarrow \|F\| \geqslant 1.$$

Widerspruch!

$$\left(\sum_{k=0}^{N} F^k\right)(I - F) = I - F^{N+1}.$$

Da $\|F\| < 1$ und $\|F^k\| \leqslant \|F\|^k$, so gilt $\lim\limits_{k\to\infty} F^k = 0$.

$$\begin{aligned} \Longrightarrow \quad \left(\lim_{N\to\infty} \sum_{k=0}^{N} F^k\right)(I-F) &= I \\ \Longrightarrow \qquad \lim_{N\to\infty} \sum_{k=0}^{N} F^k &= (I-F)^{-1} \end{aligned}$$

und

$$\|(I-F)^{-1}\| \leqslant \sum_{k=0}^{\infty} \|F\|^k = \frac{1}{1-\|F\|}.$$

□

Mit diesem Lemma erhält man

Lemma 2.26 *Sei*

$$\begin{aligned} Ax &= b, & A \in \mathbb{R}^{n,n}, A \text{ nichtsingulär}, 0 \neq b \in \mathbb{R}^n \\ (A+\triangle A)y &= b+\triangle b, & \triangle A \in \mathbb{R}^{n,n}, \triangle b \in \mathbb{R}^n, \end{aligned}$$

mit $\| \triangle A\| \leqslant \delta\|A\|, \| \triangle b\| \leqslant \delta\|b\|$. *Falls* $\delta\,\kappa(A) = r < 1$, *so ist* $A + \triangle A$ *nichtsingulär und*

$$\frac{\|y\|}{\|x\|} \leqslant \frac{1+r}{1-r}.$$

Beweis: $\|A^{-1} \triangle A\| \leqslant \delta\|A^{-1}\|\,\|A\| = r < 1. \Longrightarrow A + \triangle A$ nichtsingulär nach Lemma 2.23.
$(I + A^{-1} \triangle A)y = x + A^{-1} \triangle b$ ergibt

$$\begin{aligned} \|y\| &\leqslant \|(I + A^{-1} \triangle A)^{-1}\|\,(\|x\| + \delta\|A^{-1}\|\,\|b\|) \\ &\leqslant \frac{1}{1-r}(\|x\| + \delta\|A^{-1}\|\,\|Ax\|) \\ &\leqslant \frac{1}{1-r}\left(\|x\| + \underbrace{\delta\|A^{-1}\|\,\|A\|}_{r}\,\|x\|\right). \end{aligned}$$

□

Damit haben wir jetzt ein genaues Störungsresultat:

Satz 2.27 *Unter den Voraussetzungen von Lemma 2.26 gilt*

$$\frac{\|x-y\|}{\|x\|} \leqslant \frac{2r}{1-r}. \tag{2.28}$$

Beweis: $y - x = A^{-1} \triangle b - A^{-1} \triangle Ay$

$$\begin{aligned} \Longrightarrow \|y-x\| &\leqslant \delta\|A^{-1}\|\,\|b\| + \delta\|A^{-1}\|\,\|A\|\,\|y\| \\ &\leqslant \delta\|A^{-1}\|\,\|A\|\,\|x\| + \delta\|A^{-1}\|\,\|A\|\,\|y\| = r(\|x\| + \|y\|) \end{aligned}$$

und damit ergibt sich mit Lemma 2.26

$$\frac{\|y-x\|}{\|x\|} \leqslant r\left(1 + \frac{1+r}{1-r}\right).$$

□

Es folgt, daß der relative Fehler abhängig von der Kondition ist. Es kann allerdings sein, daß die Schranke in (2.28) eine grobe Überschätzung ist.

Beispiel 2.29

$$\begin{bmatrix} 1 & 0 \\ 0 & 10^{-6} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 10^{-6} \end{bmatrix}, \Longrightarrow x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = 10^6$. *Sei* $\triangle b = \begin{bmatrix} 10^{-7} \\ 0 \end{bmatrix}, \triangle A = 0$, *so ist* $\delta = 10^{-7}, r = 0.1$ *und (2.28) ergibt* $\frac{10^{-7}}{10^0} \leqslant \frac{2}{0.9} 0.1 = \frac{2}{9}$. *Jedoch für* $\triangle b = \begin{bmatrix} 0 \\ 10^{-7} \end{bmatrix}, \triangle A = 0$ *ist weiterhin* $\delta = 10^{-7}, r = 0.1$ *und (2.28) lautet* $\frac{10^{-1}}{10^0} \leqslant \frac{2}{9}$, *also fast Gleichheit.*

Eine verbesserte Fehleranalyse ist möglich, wenn man komponentenweise Störungsresultate verwendet.

Lemma 2.30 *Sei*

$$\begin{array}{rcll} Ax & = & b, & A \in \mathbb{R}^{n,n}, A \text{ nichtsingulär}, 0 \neq b \in \mathbb{R}^n \\ (A + \triangle A)y & = & b + \triangle b, & \triangle A \in \mathbb{R}^{n,n}, \triangle b \in \mathbb{R}^n, \end{array}$$

mit $|\triangle A| \leqslant \delta |A|, |\triangle b| \leqslant \delta |b|$. *Falls* $\delta \| \, |A^{-1}| \, |A| \, \| = r < 1$, *so ist* $A + \triangle A$ *nichtsingulär und*

$$\frac{\|y\|}{\|x\|} \leqslant \frac{1+r}{1-r},$$

wobei $\| \bullet \| = \| \bullet \|_\infty, \| \bullet \|_1, \| \bullet \|_F$.

$\longrightarrow$ Übung 2.65.

Satz 2.31 *Unter den Voraussetzungen von Lemma 2.30 gilt*

$$\frac{\|y - x\|}{\|x\|} \leqslant \frac{2r}{1-r}, \tag{2.32}$$

wobei $\| \bullet \| = \| \bullet \|_\infty, \| \bullet \|_1, \| \bullet \|_F$.

$\longrightarrow$ Übung 2.66.

Alle Resultate lassen sich auch für $\mathbb{C}$ zeigen. Komponentenweise Abschätzungen sind i.a. schwieriger aber oft aussagefähiger.

Beispiel 2.33 *Wir betrachten Beispiel 2.29. Hier ist* $\| \, |A^{-1}| \, |A| \, \|_\infty = 1$. *Sei* $\triangle b = \begin{bmatrix} 10^{-7} \\ 0 \end{bmatrix}, \triangle A = 0$, *so ist* $\delta = r = 10^{-7}$ *und (2.32) ergibt* $\frac{10^{-7}}{10^0} \leqslant \frac{2}{1-10^{-7}} 10^{-7} \approx 2 \cdot 10^{-7}$. *Für* $\triangle b = \begin{bmatrix} 0 \\ 10^{-7} \end{bmatrix}, \triangle A = 0$ *ist jetzt* $\delta = r = 10^{-1}$ *und (2.32) ergibt* $\frac{10^{-1}}{10^0} \leqslant \frac{2}{0.9} 10^{-1} = \frac{2}{9}$.

Es gibt Bestrebungen, die Analyse von Algorithmen ganz auf komponentenweise Abschätzungen zu basieren, dies ist noch im Fluß. Diese Störungssätze wenden wir nun an.
Zu Anfang betrachten wir nur gestörte Daten und exakte Rechnung:

$$gl(A) = A + E, gl(b) = b + e,$$

und angenommen

$$(A+E)\tilde{x} = b + e, \quad \text{wobei } \|E\|_\infty \leqslant \operatorname{eps} \|A\|_\infty, \|e\|_\infty \leqslant \operatorname{eps} \|b\|_\infty. \tag{2.34}$$

Störungssätze ergeben mit $\operatorname{eps} \kappa_\infty(A) \leqslant \frac{1}{2}$

$$\frac{\|x - \tilde{x}\|_\infty}{\|x\|_\infty} \leqslant 4 \operatorname{eps} \kappa_\infty(A). \tag{2.35}$$

Beides sind „bestmögliche“ Norm–schranken, d.h. keine allgemeine $\| \bullet \|_\infty$–Analyse kann bessere Schranken liefern.
Wir können konsequenterweise keinen Algorithmus als schlecht abtun, der ein ungenaues Ergebnis für eine Matrix mit $\operatorname{eps} \kappa_\infty(A) \approx 1$ liefert. Für schlecht konditionierte Probleme kann also auch ein guter Algorithmus schlechte Ergebnisse liefern.
Nun zum Gauß–Verfahren. (Äußere Produktversion, Analyse gilt auch für GAXPY.)

Satz 2.36 *Sei A eine $n \times n$ Matrix von Maschinenzahlen. Falls kein 0–Pivot während der Ausführung von Algorithmus 2.19 auftritt, dann erfüllen die berechneten Faktoren $\tilde{L}, \tilde{R}$*

$$\tilde{L}\tilde{R} = A + H \tag{2.37}$$

mit

$$|H| \leqslant 3(n-1) \operatorname{eps} (|A| + |\tilde{L}||\tilde{R}|) + \mathcal{O}(\operatorname{eps}^2) \tag{2.38}$$

Beweis: Induktion über n. $n = 1 : \checkmark$.
Angenommen Resultat sei richtig für alle $(n-1) \times (n-1)$ Maschinenzahl–Matrizen. Sei

$$A = \begin{bmatrix} \alpha & w^T \\ v & B \end{bmatrix} \begin{matrix} \}1 \\ \}n-1 \end{matrix},$$

so wird im ersten Schritt von Algorithmus 2.19 $\tilde{z} = gl(v/\alpha)$ und $\tilde{A}_1 = gl(B - \tilde{z}w^T)$ berechnet. Also folgt

$$\tilde{z} = \frac{1}{\alpha} v + f, \text{ mit } |f| \leqslant \operatorname{eps} \frac{|v|}{\alpha} \tag{2.39}$$

und

$$\tilde{A}_1 = B - \tilde{z}w^T + F, \text{ mit } |F| \leqslant 2 \operatorname{eps} (|B| + |\tilde{z}|\,|w|^T) + \mathcal{O}(\operatorname{eps}^2). \tag{2.40}$$

Dann wird die LR–Zerlegung von $\tilde{A}_1$ berechnet. Mit Induktion folgt $\tilde{A}_1 = \tilde{L}_1\tilde{R}_1$ mit

$$\tilde{L}_1\tilde{R}_1 = \tilde{A}_1 + H_1, \text{ wobei } |H_1| \leqslant 3(n-2) \operatorname{eps} (|\tilde{A}_1| + |\tilde{L}_1|\,|\tilde{R}_1|) + \mathcal{O}(\operatorname{eps}^2). \tag{2.41}$$

Also gilt

$$\begin{aligned} \tilde{L}\tilde{R} &= \begin{bmatrix} 1 & 0 \\ \tilde{z} & \tilde{L}_1 \end{bmatrix} \begin{bmatrix} \alpha & w^T \\ 0 & \tilde{R}_1 \end{bmatrix} \\ &= A + \begin{bmatrix} 0 & 0 \\ \alpha f & H_1 + F \end{bmatrix} = A + H. \end{aligned}$$

Aus (2.40) folgt

$$|\tilde{A}_1| \leqslant (1 + 2\,\mathrm{eps}\,)(|B| + |\tilde{z}|\,|w|^T) + \mathcal{O}(\,\mathrm{eps}^{\,2}).$$

Mit (2.40), (2.41) folgt

$$|H_1 + F| \leqslant 3(n-1)\,\mathrm{eps}\,(|B| + |\tilde{z}|\,|w|^T + |\tilde{L}_1|\,|\tilde{R}_1|) + \mathcal{O}(\,\mathrm{eps}^{\,2})$$

und da $|\alpha f| \leqslant \mathrm{eps}\,|v|$, folgt

$$|H| \leqslant 3(n-1)\,\mathrm{eps}\,\left\{ \begin{bmatrix} |\alpha| & |w|^T \\ |v| & |B| \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ |\tilde{z}| & |\tilde{L}_1| \end{bmatrix} \begin{bmatrix} |\alpha| & |w|^T \\ 0 & |\tilde{R}_1| \end{bmatrix} \right\} + \mathcal{O}(\,\mathrm{eps}^{\,2}).$$

□

Nun haben wir eine Analyse über die LR–Zerlegung, jetzt müssen wir noch analysieren, was die $\triangle$–Löser machen.

Satz 2.42 *Seien $\tilde{L}, \tilde{R}$ die berechneten LR–Faktoren aus Algorithmus 2.16 oder 2.19. Bei Verwendung von Algorithmus 2.2 bzw. 2.5 zur Lösung von $\tilde{L}y = b$ und $\tilde{R}x = \tilde{y}$ ergibt sich $(A + E)\tilde{x} = b$ mit*

$$|E| \leqslant n\,\mathrm{eps}\,(3|A| + 5|\tilde{L}|\,|\tilde{R}|) + \mathcal{O}(\,\mathrm{eps}^{\,2}). \tag{2.43}$$

Beweis: Aus (2.3) und (2.6) ergibt sich

$$\begin{array}{ll} (\tilde{L} + F)\tilde{y} = b & |F| \leqslant n\,\mathrm{eps}\,|\tilde{L}| + \mathcal{O}(\,\mathrm{eps}^{\,2}), \\ (\tilde{R} + G)\tilde{x} = \tilde{y} & |G| \leqslant n\,\mathrm{eps}\,|\tilde{R}| + \mathcal{O}(\,\mathrm{eps}^{\,2}). \end{array}$$

Also

$$(\tilde{L} + F)(\tilde{R} + G)\tilde{x} = (\tilde{L}\tilde{R} + F\tilde{R} + \tilde{L}G + FG)\tilde{x} = b$$

$\tilde{L}\tilde{R} = A + H$ aus Satz 2.36 mit

$$|H| \leqslant 3(n-1)\,\mathrm{eps}\,(|A| + |\tilde{L}|\,|\tilde{R}|) + \mathcal{O}(\,\mathrm{eps}^{\,2}).$$

Setze $E := H + F\tilde{R} + \tilde{L}G + FG$, so gilt

$$\begin{array}{rcl} |E| & \leqslant & |H| + |F|\,|\tilde{R}| + |\tilde{L}|\,|G| + \mathcal{O}(\,\mathrm{eps}^{\,2}) \\ & \leqslant & 3n\,\mathrm{eps}\,(|A| + |\tilde{L}|\,|\tilde{R}|) + 2n\,\mathrm{eps}\,(|\tilde{L}|\,|\tilde{R}|) + \mathcal{O}(\,\mathrm{eps}^{\,2}). \end{array}$$

□

Wäre nicht der Term $|\tilde{L}|\,|\tilde{R}|$ in der Abschätzung, welcher groß sein kann, so wäre der Algorithmus rückwärts stabil. Da wir jedoch nichts gegen kleine Pivots machen können, falls diese auftauchen, kann $|\tilde{L}|, |\tilde{R}|$ sehr groß werden, und der Algorithmus ist NICHT rückwärts stabil.
$\longrightarrow$ Übung
Ausweg ist Pivotisierung.

2.1.4 Partielle Pivotisierung (Spaltenpivotisierung)

Um zu vermeiden, daß 0–Pivots oder sehr kleine Pivots auftauchen, wird jeweils in der momentanen Spalte, das betragsmäßig maximale Element gesucht und durch eine Zeilenvertauschung (Permutation) in die Diagonalposition gebracht.

Grundidee:

FOR $k = 1 : n-1$
Bestimme Permutationsmatrix P_k, so daß für $z = P_k A^{(k-1)} e_k$ gilt: $|z(k)| = \|z(k:n)\|_\infty$.
Setze $\hat{A}^{(k-1)} = P_k A^{(k-1)}$.
Bestimme Gauß Transformation M_k, so daß für $v = M_k \hat{A}^{(k-1)} e_k$ gilt: $v(k+1:n) = 0$.
Setze $A^{(k)} = M_k P_k A^{(k-1)}$
END

Diese Vorgehensweise heißt Spaltenpivotisierung oder partielle Pivotisierung und garantiert, daß alle Multiplikatoren vom Betrag $\leqslant 1$ sind.

$$|(P_k M_{k-1} \cdots M_1 P_1 A)_{kk}| = \max_{k \leqslant i \leqslant n} |(P_k M_{k-1} \cdots M_1 P_1 A)_{ik}|\,,\ k = 1 : n-1.$$

Falls das Pivot $= 0$ ist, so kann man einfach den Schritt der Elimination weglassen, da dann alle Elemente in dieser Spalte $= 0$ sind.

Man erhält dann den folgenden Algorithmus.

Algorithmus 2.44 (Gauß–Elim. mit partieller Pivotisierung, Äußere Produktversion)
Input: $A \in \mathbb{K}^{n,n}$.
Output: *Gauß–Transformationen $M_1, \ldots, M_{k-1}$ und Permutationen $P_1, \ldots, P_{n-1}$, so daß $M_{n-1} P_{n-1} \cdots M_1 P_1 A = R$ obere $\triangle$–Matrix.*
Keiner der Multiplikatoren hat Betrag > 1. $A(k+1:n,k)$ wird durch $-M_k(k+1:n,k)$, $k = 1 : n-1$ überschrieben, $A(1:k,k)$ wird durch $R(1:k,k)$, $k = 1 : n$ überschrieben. Im Pivotvektor $piv(1:n-1)$ werden die Vertauschungen gespeichert. P_k vertauscht die Zeilen k und $piv(k)$, $k = 1 : n-1$.

```
FOR   k = 1 : n-1
      Bestimme μ (k ⩽ μ ⩽ n) mit |A(μ,k)| = ‖A(k:n,k)‖∞
      A(k,k:n) ⟷ A(μ,k:n)
      piv(k) = μ;
      IF    A(k,k) ≠ 0
            t = GAUSS(A(k:n,k));
            A(k+1:n,k) = t;
            A(k:n,k+1:n) = GAUSSAPP(A(k:n,k+1:n),t);
      END
END
```

Kosten Für die Bestimmung aller μ's: $\mathcal{O}(n^2)$ Vergleiche, $\frac{2n^3}{3}$ flops.
Für das lineare Gleichungssystem $Ax = b$ macht man also

- $y = M_{n-1} P_{n-1} \cdots M_1 P_1 b$
- Löse $Rx = y$.

Sämtliche Informationen in A und piv.

Beispiel 2.45

$$
\begin{aligned}
A &= \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix} \\
P_1 &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \\
P_1 A &= \begin{bmatrix} 6 & 18 & -12 \\ 2 & 4 & -2 \\ 3 & 17 & 10 \end{bmatrix} \qquad piv(1) = 3 \\
M_1 &= \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{bmatrix} \qquad M_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & -2 & 2 \\ 0 & 8 & 16 \end{bmatrix} \\
P_2 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad piv(2) = 3 \\
M_2 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{4} & 1 \end{bmatrix} \qquad R = \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & 0 & 6 \end{bmatrix}
\end{aligned}
$$

Was passiert mit L?

Satz 2.46 *Man verwende Gauß–Elimination mit partieller Pivotisierung zur Berechnung von*

$$M_{n-1} P_{n-1} \cdots M_1 P_1 A = R \tag{2.47}$$

mit Algorithmus 2.44, so gilt

$$PA = LR,$$

mit $P = P_{n-1} \cdots P_1$ (Permutationsmatrix), L untere $\triangle$–Matrix mit 1–Diagonale, $|l_{ij}| \leqslant 1$. Die k–te Spalte von L ist unterhalb der Diagonalen eine permutierte Version des k–ten Gauß-Vektors.
Speziell für $M_k = I - t^{(k)} e_k^T$ gilt

$$L(k+1:n,k) = g(k+1:n),\ g = P_{n-1} \cdots P_{k+1} t^{(k)}.$$

Beweis: Aus (2.47) folgt

$$\tilde{M}_{n-1} \cdots \tilde{M}_1 PA = R \quad \text{mit}$$

$$
\begin{aligned}
\tilde{M}_{n-1} &= M_{n-1}, \\
\tilde{M}_k &= P_{n-1} \cdots P_{k+1} M_k P_{k+1} \cdots P_{n-1},\ k \leqslant n-2.
\end{aligned}
$$

Da P_j nur Zeilen j und $\mu \geqslant j$ vertauscht, so folgt $P_j(1:j-1, 1:j-1) = I_{j-1}$. Also ist $\tilde{M}_k$ eine Gauß–Transformation mit Gauß–Vektor $\tilde{t}^{(k)} = P_{n-1} \cdots P_{k+1} t^{(k)}$. □

Durch die einfache Ersetzung von

$$
\begin{aligned}
A(k:k:n) &\longleftrightarrow A(\mu, k:n) \text{ durch} \\
A(k, 1:n) &\longleftrightarrow A(\mu, 1:n) \text{ in Algorithmus 2.44}
\end{aligned}
$$

gilt dann wieder $A(i,j)$ enthält $L(i,j)$ für $i > j$ nach Ende des Algorithmus.

$\longrightarrow$ Übung 2.76: Analoge GAXPY Version.

Fehleranalyse: Da Permutationen rundungsfehlerfrei durchgeführt werden können, kann man analog zeigen, daß die berechnete Lösung $\tilde{x}$ $(A+E)\tilde{x} = b$ erfüllt mit

$$|E| \leqslant n \operatorname{eps} \left(3|A| + 5\tilde{P}^T|\tilde{L}|\,|\tilde{R}|\right) + \mathcal{O}(\operatorname{eps}^2), \tag{2.48}$$

wobei $\tilde{P}, \tilde{L}, \tilde{R}$ die berechneten P, L, R sind.

Durch die Pivotisierung folgt

$$\begin{aligned} &\|\tilde{L}\|_\infty \leqslant n, \text{ also folgt} \\ &\|E\|_\infty \leqslant n \operatorname{eps} \left(3\|A\|_\infty + 5n\|\tilde{R}\|_\infty\right) + \mathcal{O}(\operatorname{eps}^2). \end{aligned}$$

Problem: Schranke für $\|\tilde{R}\|_\infty$.
Definiere den Wachstumsfaktor ρ durch

$$\rho = \max_{i,j,k} \frac{|\tilde{a}_{ij}^{(k)}|}{\|A\|_\infty}, \tag{2.49}$$

wobei $\tilde{A}^{(k)}$ die berechnete Version von $A^{(k)}$ ist. Dann folgt

$$\|E\|_\infty \leqslant 8n^3\rho\|A\|_\infty \operatorname{eps} + \mathcal{O}(\operatorname{eps}^2). \tag{2.50}$$

Die Schranke hängt also wesentlich von ρ ab (n^3 ist i.a. vernachlässigbar in der Praxis, da nicht auftretend).
ρ ist typisch von der Ordnung 10, kann aber 2^{n-1} sein.
$\longrightarrow$ Übung 2.71.

Im allgemeinen ist Gauß mit partieller Pivotisierung zuverlässig und kann relativ sorglos verwendet werden.

2.1.5 Vollständige Pivotisierung

Um den Wachstumsfaktor zu verkleinern gibt es eine Variante, in der das Pivot aus der gesamten Matrix $A^{(k-1)}(k:n, k:n)$ ausgewählt wird, d.h. wir bestimmen die Zerleung

$$M_{n-1}P_{n-1}\cdots M_1P_1AQ_1\cdots Q_{n-1} = R$$

mit Permutationsmatrizen P_i, Q_i, die so bestimmt werden, daß

$$\left|(P_kA^{(k-1)}Q_k)_{kk}\right| = \max_{k\leqslant i,j\leqslant n} \left|(P_kA^{(k-1)}Q_k)_{ij}\right|.$$

Satz 2.51 *Bei Verwendung von Gauß–Elimination mit vollständiger Pivotisierung zur Berechnung von*

$$M_{n-1}P_{n-1}\cdots M_1P_1AQ_1\cdots Q_{n-1} = R \tag{2.52}$$

gilt

$$PAQ = LR$$

mit $P = P_{n-1}\cdots P_1$, $Q = Q_1\cdots Q_{n-1}$ und L ist untere $\triangle$–Matrix mit 1–Diagonale und $|l_{ij}| \leqslant 1$. Für $M_k = I - t^{(k)}e_k^T$ gilt

$$L(k+1:n,k) = g(k+1:n) \text{ mit } g = P_{n-1}\cdots P_{k+1}t^{(k)}.$$

Beweis: Analog zu Satz 2.46. □

Algorithmus 2.53 (Gauß–Elim. mit vollständiger Pivotisierung)

Input: $A \in \mathbb{K}^{n,n}$.

Output: *LR–Zerlegung von PAQ mit P, Q Produkte von elementaren Permutationsmatrizen, $A(1:k,k)$ wird durch $R(1:k,k)$, $k = 1:n$ und $A(k+1:n,k)$ durch $L(k+1:n,k)$, $k = 1:n-1$ überschrieben. P_k vertauscht Zeilen k und $p(k)$, Q_k vertauscht Spalten k und $q(k)$.*

FOR $k = 1:n-1$
  Bestimme μ, λ, *so daß* $|A(\mu,\lambda)| = \max\{|A(i,j)| : i,j = k:n\}$
  $A(k,1:n) \longleftrightarrow A(\mu,1:n)$
  $A(1:n,k) \longleftrightarrow A(1:n,\lambda)$
  $p(k) = \mu$;
  $q(k) = \lambda$;
  IF $A(k,k) \neq 0$
    $t = GAUSS(A(k:n,k))$;
    $A(k+1:n,k) = t$;
    $A(k:n,k+1:n) = GAUSSAPP(A(k:n,k+1:n),t)$;
  END
END

Kosten $\frac{2n^3}{3}$ flops und Vergleiche. Vergleiche nicht vernachlässigbar.

- Vollständige Pivotisierung erlaubt LR–Zerlegung für $\operatorname{rank} A = r < n$.
- In exakter Arithmetik gilt für Algorithmus 2.53

$$\left|a_{ij}^{(k)}\right| \leqslant k^{\frac{1}{2}}(2 \cdot 3^{\frac{1}{2}} \cdots k^{\frac{1}{k-1}})^{\frac{1}{2}} \max|a_{ij}|.$$

  Die Schranke ist sehr langsam wachsend mit k. Mit der empirischen Tatsache, daß $\rho \approx 10$, kann man aussagen, daß Gauß–Elimination mit vollständiger Pivotisierung rückwärts stabil ist, d.h. die Methode löst exakt

$$(A+E)\hat{x} = b$$

  für kleines E. Wilkinsons Vermutung Schranke $= n$ ist nicht richtig.

2.1.6 Abschätzung der Genauigkeit

Das Residuum der berechneten Lösung von $Ax = b$ ist der Vektor $b - A\tilde{x}$. Ein kleines Residium bedeutet daß $A\tilde{x}$ eine gute Näherung von b ist. Wenn man annimmt, daß $(A+E)\tilde{x} = b, \|E\|_\infty \approx \text{eps}\ \|A\|_\infty$, so folgt $\|b - A\tilde{x}\|_\infty \approx \text{eps}\ \|A\|_\infty \|\tilde{x}\|_\infty$.

Heuristik I Gauß–Elimination erzeugt eine Lösung $\tilde{x}$ mit relativ kleinem Residuum.

! Jedoch kleine Residuen implizieren nicht hohe Genauigkeit, (nur bei kleiner Kondition).

$$\frac{\|\tilde{x} - x\|_\infty}{\|x\|_\infty} \approx \text{eps}\, \kappa_\infty(A).$$

Heuristik II Falls $\text{eps} \approx 10^{-d}$ und $\kappa_\infty(A) \approx 10^q$, dann erzeugt Gauß–Elimination eine Lösung mit ungefähr $d - q$ korrekten Dezimalstellen.

Beispiel 2.54

$$\begin{bmatrix} .986 & .579 \\ .409 & .237 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} .235 \\ .107 \end{bmatrix},\ \kappa(A)_\infty \approx 700,\ x = \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$$

eps	$\tilde{x}_1$	$\tilde{x}_2$	$\frac{\|\tilde{x}-x\|}{\|x\|_\infty}$	$\frac{\|b-A\tilde{x}\|_\infty}{\|A\|_\infty\|\tilde{x}\|_\infty}$
10^{-3}	2.11	-3.17	$5 \cdot 10^{-2}$	$2.0 \cdot 10^{-3}$
10^{-4}	1.986	-2.975	$8 \cdot 10^{-3}$	$1.5 \cdot 10^{-4}$
10^{-5}	2.0019	-3.0032	$1 \cdot 10^{-3}$	$2.1 \cdot 10^{-6}$
10^{-6}	2.00025	-3.00094	$3 \cdot 10^{-4}$	$4.2 \cdot 10^{-7}$

2.1.7 Skalierung

Eine weitere Verbesserung ist durch Skalierung der Daten zu erreichen.

$$Ax = b \Longleftrightarrow D_1^{-1}AD_2y = D_1^{-1}b,\ y = D_2^{-1}x$$

Falls man für D_1, D_2 Diagonalmatrizen aus Maschinenzahlen der Form

$$\text{diag}\,(P^{r_1}, P^{r_2}, \ldots, P^{r_n})$$

nimmt, so kann die Skalierung ohne Rundungsfehler in $\mathcal{O}(n^2)$ flops durchgeführt werden. Dann gilt

$$\frac{\|D_2^{-1}(\tilde{x}-x)\|_\infty}{\|D_2^{-1}x\|_\infty} = \frac{\|\tilde{y}-y\|_\infty}{\|y\|_\infty} \approx \text{ eps } \kappa_\infty(D_1^{-1}AD_2). \tag{2.55}$$

Wenn man also $\kappa_\infty(D_1^{-1}AD_2)$ gegen $\kappa_\infty(A)$ verkleinert, erwartet man eine Verbesserung des Resultats.

Varianten:

a) Zeilenskalierung: $D_2 = I$, D_1 so, daß alle Zeilen $\approx$ gleiche ∞–Norm haben.

b) Zeilen–Spalten–Gleichgewichtung: Wähle D_1, D_2 so, daß alle Zeilen und Spalten ∞–Norm im Intervall $\left[\frac{1}{p}, 1\right]$ haben (p Basis der Maschine).

Beispiel 2.56

$$\begin{bmatrix} 10 & 100000 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 100000 \\ 2 \end{bmatrix} \Longleftrightarrow \begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Bei dreistelliger Arithmetik, $p = 10$ *ergibt das erste System* $\tilde{x} = \begin{bmatrix} 0.00 \\ 1.00 \end{bmatrix}$ *und das zweite* $\begin{bmatrix} 1.00 \\ 1.00 \end{bmatrix}$. *Exakte Lösung* $x = \begin{bmatrix} 1.0001... \\ 0.9999... \end{bmatrix}$.

2.1.8 Iterative Verbesserung

Angenommen, wir haben $Ax = b$ mittels Gauß-Elimination mit partieller Pivotisierung gelöst, $PA = LR$, und wollen jetzt die Lösungsgenauigkeit verbessern. Wir könnten folgendes ausführen:

$$\begin{cases} r = b - A\tilde{x} \quad \text{(doppelt genau)}\ , \\ \text{Löse } Ly = Pr, \\ \text{Löse } Rz = y, \\ \text{Setze } x = \tilde{x} + z, \end{cases} \tag{2.57}$$

so folgt bei exakter Rechnung

$$Ax = A\tilde{x} + Az = (b - r) + r = b.$$

Wenn man (2.57) allerdings einfach so ausführt, ist das Ergebnis nicht genauer als $\tilde{x}$. Dies ist zu erwarten, denn $\tilde{r} = gl(b - A\tilde{x})$ hat im allgemeinen kaum richtige signifikante Stellen (siehe Heuristik I). Also

$$\tilde{z} = gl(A^{-1}r) = A^{-1} \cdot \text{Rauschen} = \text{Rauschen}.$$

! Um eine Verbesserung zu erhalten, sollte man daher $r = b - Ax$ in doppelter Genauigkeit rechnen (doppelt so viele Stellen wie sonst). Dieser Prozeß kann iteriert werden.

Heuristik III Bei Maschinengenauigkeit $\text{eps} = 10^{-d}$ und $\kappa_\infty(A) \approx 10^q$, hat nach k Schritten von (2.57) (mit doppelter Genauigkeit für das Residuum) das berechnete x ungefähr $\min\{d, k(d-q)\}$ korrekte Stellen.

Kosten $\mathcal{O}(n^2)$ pro Schritt.
Bei Rechnung von $PA = LR$ in einfacher Genauigkeit und $\text{eps} \cdot \kappa_\infty(A) \leqslant 1$, ist die Lösung korrekt in einfacher Genauigkeit nach einer Iteration. Also kaum Verteuerung und Verbesserung der Lösung. Problem: maschinenabhängig, da an Genauigkeit orientiert.
$\longrightarrow$ Übung 2.75: Konditionsschätzer

2.1.9 Übungen zu Kapitel 2.1

Übung 2.58 *Schreibe einen* **MATLAB** *-Algorithmus zur Lösung eines Gleichungssystems $Lx = b$ mit einer unteren Dreiecksmatrix L (Algorithmus 2.2 der Vorlesung). Verfahre analog bei $Ux = b$ wobei U obere Dreiecksmatrix ist (Algorithmus 2.5 der Vorlesung).*

Übung 2.59 *Schreibe einen* **MATLAB** *-Algorithmus zur Berechnung der Determinante einer $n \times n$–Matrix mit Hilfe des Laplaceschen Entwicklungssatzes. Der Laplacesche Entwicklungssatz zur Determinantenentwicklung (nach der ersten Zeile) lautet:*
$\det A = \sum_{j=1}^{n} (-1)^{j+1} a_{1j} \det A(2:n, [1:j-1, j+1:n])$. *Diese Formel soll natürlich rekursiv ohne* 'det' *aus* **MATLAB** *angewendet werden.*

Übung 2.60 *Vergleiche folgende vier Verfahren in* **MATLAB** *zur Lösung von $Ax = b, A \in \mathbb{R}^{n,n}, b \in \mathbb{R}^n$. Messe die Rechenzeit für $n = 4, 6, 8, 10$ und zufällig erzeugte A, b.*

1. *den '\' aus* **MATLAB**. *(Hinter diesem steckt der Gauß-Alg.)*
2. *Einen selbstgeschrieben Gauß-Algorithmus entsprechend der Vorlesung*
3. *Die Cramersche Regel mit Hilfe der eingebautem* 'det'*–Funktion*
4. *Die Cramersche Regel mit Hilfe des Laplaceschen Entwicklungssatzes*

Die Cramersche Regel für $Ax = b$ ist wie folgt definiert:
$x_i = \det[A(:, 1:i-1), b, A(:, i+1:n)] / \det A,\ i = 1, \ldots, n$
Der Laplacesche Entwicklungssatz zur Determinantenentwicklung (nach der ersten Zeile) lautet:
$\det A = \sum_{j=1}^{n} (-1)^{j+1} a_{1j} \det A(2:n, [1:j-1, j+1:n])$. *Diese Formel soll natürlich rekursiv ohne* 'det' *aus* **MATLAB** *angewendet werden.*

Übung 2.61 *Die exakte Lösung des Gleichungssystems*

$$\begin{pmatrix} 1 & 0.99 \\ 0.99 & 0.98 \end{pmatrix} x = \begin{pmatrix} 1.99 \\ 1.97 \end{pmatrix}$$

ist $x = \binom{1}{1}$. *Mit* $y = \binom{1.94}{0.050459}$ *gilt aber auch*

$$\begin{pmatrix} 1 & 0.99 \\ 0.99 & 0.98 \end{pmatrix} y = \begin{pmatrix} 1.98995441 \\ 1.97004982 \end{pmatrix}.$$

Erkläre dieses Verhalten anhand der Störungstheorie der Vorlesung.

Übung 2.62 *Schreibe ein* **MATLAB** *–Programm, welches den Gauß–Algorithmus der Vorlesung ohne Pivotisierung auf folgendes Gleichungssystem (Hilbertmatrix, in* **MATLAB 'hilb(n)'***) anwendet.*

$$Hx = b,\ H = \left(\frac{1}{i+j-1} \right)_{i,j=1,\ldots,n},\ b = H\,(1, \ldots, 1)^T.$$

Vergleiche die Lösung des Algorithmus mit der exakten Lösung $x = (1, \ldots, 1)^T$ *sowie der von* **MATLAB** *mittels* $\backslash$ *berechneten Lösung für* $n = 2, 3, 4, \ldots, 30$. *Was stellst Du fest? Erklärung!*

Übung 2.63 *Eine Matrix* $A = (a_{ij})_{i,j=1,\ldots,n}$ *heißt strikt diagonaldominant, wenn gilt*

$$|a_{jj}| - \sum_{\substack{i=1 \\ i \neq j}}^{n} |a_{ij}| > 0, \text{ für alle } j = 1, \ldots, n.$$

Zeige:

1. *Ist A strikt diagonaldominant, dann ist A nicht singulär.*
2. *Ist A strikt diagonaldominant, dann gilt das auch für alle Hauptabschnittsmatrizen.*
3. *Ist A strikt diagonaldominant, dann besitzt A eine LR–Zerlegung.*

Übung 2.64 *Sei*

$$A = \begin{pmatrix} a_{11} & a_{12} & & \\ a_{21} & \ddots & \ddots & \\ & \ddots & \ddots & a_{n-1,n} \\ & & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

tridiagonal. Wieviel flops $(+, -*, /)$ *braucht der Gauß-Algorithmus, falls nicht permutiert werden muß und der Algorithmus durchgeht? Wie teuer sind Vorwärts– und Rückwärtseinsetzen?*

Übung 2.65 *Zeige Lemma 2.30 aus der Vorlesung: Sei*

$$\begin{aligned} Ax &= b, & A \in \mathbb{R}^{n,n}, 0 \neq b \in \mathbb{R}^n \\ (A + \triangle A)y &= b + \triangle b, & \triangle A \in \mathbb{R}^{n,m}, \triangle b \in \mathbb{R}^n, \end{aligned}$$

mit $|\triangle A| \leqslant \delta |A|, |\triangle b| \leqslant \delta |b|$. *Falls* $\delta \,\| \, |A^{-1}| \, |A| \, \|_\infty = r < 1$, *so ist* $A + \triangle A$ *nichtsingulär und*

$$\frac{\|y\|_\infty}{\|x\|_\infty} \leqslant \frac{1+r}{1-r}.$$

Tip: Verfahre analog zum Beweis des entsprechenden Lemmas 2.26 über Normen.

Übung 2.66 *Zeige Satz 2.31 aus der Vorlesung: Unter den Voraussetzungen von Lemma 2.30 gilt*

$$\frac{\|y-x\|}{\|x\|} \leqslant \frac{2r}{1-r},$$

wobei $\|\bullet\| = \|\bullet\|_\infty, \|\bullet\|_1, \|\bullet\|_F$. *Tip: Verfahre analog zum Beweis des entsprechenden Satzes 2.27 über Normen.*

Übung 2.67 *Sei* $1 > \varepsilon > 0$,

$$A = \begin{pmatrix} \varepsilon & -1 & & \\ & \ddots & \ddots & \\ & & \ddots & -1 \\ & & & \varepsilon \end{pmatrix} \in \mathbb{R}^{n,n}.$$

1. *Zeige, es existiert eine singuläre Matrix* B *mit* $\|B-A\|_\infty \leqslant \varepsilon^n$. *D.h. der Abstand von* A *zu den singulären Matrizen ist höchstens* ε^n. *Tip: man füge in der Position* $(n,1)$ *einen geeigneten Eintrag ein, so dass bei anschließender Determinantenbildung eine Null entsteht.*
2. *Berechne* A^{-1}
3. *Bestimme für* $\|\bullet\|_\infty$ *die Normen von* A, A^{-1}, *die Konditionszahl sowie die Skeel-Konditionszahl* $\| \, |A| \, |A^{-1}| \, \|$.
4. *Wende für* $\varepsilon \leqslant \frac{1}{4}$ *und* $\delta = \varepsilon^n/4$, $b = (1,\ldots,1)^T$, $\Delta b = \delta \cdot b$ *und* $\Delta A = 0$ *die Sätze 2.27, 2.31 der Vorlesung an. Wie scharf sind die Ergebnisse?*

Übung 2.68 *Sei*

$$L = \begin{pmatrix} 1 & & & 0 \\ l_{21} & \ddots & & \\ \vdots & \ddots & \ddots & \\ l_{n1} & \cdots & l_{n,n-1} & 1 \end{pmatrix}, \; l_k = \left.\begin{pmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1,k} \\ \vdots \\ l_{n,k} \end{pmatrix}\right\} k \quad k = 1,\ldots,n-1.$$

Zeige:

1. $(I + l_k e_k^T)(I + l_m e_m^T) = I + l_k e_k^T + l_m e_m^T$ *für alle* $k \leqslant m$. *Hier bezeichnen* $e_1,\ldots,e_n$ *die Einheitsvektoren.*
2. $L = (I + l_1 e_1^T) \cdots (I + l_{n-1} e_{n-1}^T)$.
3. $L^{-1} = (I - l_{n-1} e_{n-1}^T) \cdots (I - l_1 e_1^T)$.

Übung 2.69 *Zähle die genaue Anzahl* flops *in der GAXPY-Version des Gauß–Algorithmus (Algorithmus 2.20 der Vorlesung). Zeige dass* $\frac{2}{3}n^3 + \mathcal{O}(n^2)$ flops *benötigt werden.*

Übung 2.70 *Schreibe* **MATLAB** *–Programme, welche den Gauß–Algorithmus der Vorlesung mit partieller Pivotisierung sowie vollständiger Pivotisierung realisieren. Verwende als Testbeispiel* $Hx = b$ *wieder die Hilbertmatrix* H, $x = (1,\ldots,1)^T$ *für* $n = 2,3,4,\ldots,30$.

Übung 2.71 *Bestimme den Wachstumsfaktor* $\rho = \max\limits_{i,j,k} \dfrac{\left|a_{ij}^{(k)}\right|}{\|A\|_\infty}$ *aus der Vorlesung bei exakter Arithmetik und partieller Pivotisierung für die Matrix*

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & 1 \\ -1 & 1 & \ddots & \vdots & 1 \\ -1 & -1 & \ddots & 0 & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ -1 & \cdots & \cdots & -1 & 1 \end{pmatrix}.$$

Übung 2.72 *Sei* $A \in \mathbb{R}^{n,n}$, $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ *und nehme an, daß* A *und* $A_{11} \in \mathbb{R}^{k,k}$ *nicht singulär seien. Das Schur–Komplement ist gegeben durch* $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$. *Zeige:*

1. *A läßt sich schreiben als*
$$A = \begin{pmatrix} I & O \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & O \\ O & S \end{pmatrix} \begin{pmatrix} I & A_{11}^{-1}A_{12} \\ O & I \end{pmatrix}.$$
2. *S ist nicht singulär.*
3. *A^{-1} läßt sich schreiben als* $A^{-1} = \begin{pmatrix} * & * \\ * & S^{-1} \end{pmatrix}$ *($*$ heißt, die Einträge interessieren uns hier nicht).*
4. *Ist A symmetrisch, so ist S auch symmetrisch.*
5. *Ist A positiv definit d.h. $x^TAx > 0$, $\forall x \neq 0$, so ist S auch positiv definit, d.h. $y^TSy > 0$, $\forall y \neq 0$.*
6. *Ist A invers nicht negativ, dann ist auch S invers nicht negativ (A heißt invers nicht negativ, falls A^{-1} nur nicht negative Einträge hat).*

Übung 2.73 *Sei* $A \in \mathbb{R}^{n,n}$, $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ *Falls* $A_{11} \in \mathbb{R}^{k,k}$ *nicht singulär ist, dann ist das Schur–Komplement gegeben durch* $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$. *Zeige:*

1. *Ist A strikt diagonaldominant, dann (ist A_{11} nicht singulär und) S ist strikt diagonaldominant. Was bedeutet das für den Gauß–Algorithmus mit partieller Pivotisierung?*
2. *Ist A eine M–Matrix, dann ist auch S eine M–Matrix. (A heißt M–Matrix, falls A invers nicht negativ ist und alle Einträge von A außerhalb der Diagonalen kleiner oder gleich 0 sind).*
   Tip: Aus der Definition M–Matrix folgt bereits, daß die Diagonaleinträge von A positiv sind. Warum?

Übung 2.74 1. *Schreibe ein* **MATLAB** *–Programm, welches den Gauß–Algorithmus der Vorlesung mit vollständiger Pivotisierung sowie Zeilenskalierung und iterativer Verbesserung realisiert. Verwende als Testbeispiel $Hx = b$ wieder die Hilbertmatrix H, $x = (1, \ldots, 1)^T$ für $n = 2, 3, 4, \ldots, 30$. (Siehe Übung 2.62).*

2. *Untersuche die Genauigkeitsverbesserung mittels iterativer Verbesserung dadurch, dass Daten und Rechenoperationen mit zufälligen Werten der Form $(1+\varepsilon)$ gestört werden, wobei $|\varepsilon| \leqslant \sqrt{\text{eps}}$ ist (einfache statt doppelte Genauigkeit).*
   Vergleiche die iterative Verbesserung mit und ohne gestörte Berechnung des Residuums.

Übung 2.75 *Gegeben sei eine nichtsinguläre untere Dreiecksmatrix* $L \in \mathbb{R}^{n,n}$.

1. *Nehme an, dass $y \in \mathbb{R}^n$ gegeben ist und x die Gleichung $Lx = y$ erfüllt. Zeige: falls $c\|y\|_p \leqslant \|x\|_p$ gilt, so ist $\|L^{-1}\|_p \geqslant c$.*

2. *Nutze (1) zur Entwicklung eines Konditionsschätzers. Wähle ein y der Form $y^\top = (\pm 1 \quad \cdots \quad \pm 1)$ und betrachte folgende Variante des Vorwärtseinsetzens*

   $p = 0 \in \mathbb{R}^n$
   for $k = 1 : n$
   Wähle das Vorzeichen von $y(k)$
   $x(k) = (y(k) - p(k))/L(k,k)$
   $p(k+1:n) = p(k+1:n) + L(k+1:n,k)x(k)$
   end
   $\kappa = \|x\|_\infty \|L\|_\infty$

   (a) *Schreibe ein MATLAB–Programm, in dem y so gewählt wird, dass für $k = 1, \ldots, n$, $|x(k)|$ sukzessive maximiert wird.*

   (b) *Schreibe ein MATLAB–Programm, in dem y so gewählt wird, dass für $k = 1, \ldots, n$, $|x(k)| + \|p(k+1:n) + L(k+1:n,k)x(k)\|_1$ sukzessive maximiert wird.*

   (c) *Vergleiche beide Varianten in MATLAB mit der eingebauten Funktion **condest** sowie der exakten Kondition **cond**. Was stellst du fest?*

Übung 2.76 *Schreibe ein* **MATLAB** *–Programm für die GAXPY–Version des Gauß–Algorithmus der Vorlesung mit partieller Pivotisierung. Verwende als Testbeispiel $Hx = b$ die Hilbertmatrix H, $x = (1, \ldots, 1)^\top$ für $n = 2, 3, 4, \ldots, 30$.*

Übung 2.77 *Untersuche in* **MATLAB** *die Genauigkeitsverbesserung mittels iterativer Verbesserung bei der GAXPY–Version des Gauß–Algorithmus mit partieller Pivotisierung dadurch, dass Daten und Rechenoperationen mit zufälligen Werten der Form $(1 + \varepsilon)$ gestört werden, wobei $|\varepsilon| \leqslant \sqrt{\text{eps}}$ ist (einfache statt doppelte Genauigkeit).*
Vergleiche die iterative Verbesserung mit und ohne gestörte Berechnung des Residuums. Verwende als Testbeispiel $Hx = b$ wieder die Hilbertmatrix H, $x = (1, \ldots, 1)^T$ für $n = 2, 3, 4, \ldots, 30$.

2.2 Der Cholesky–Algorithmus, Bandmatrizen

Ein guter numerischer Algorithmus sollte spezielle Eigenschaften des mathematischen Problems berücksichtigen. Der Gauß–Algorithmus ist auf allgemeine lineare Gleichungssysteme zugeschnitten. Nun sind aber viele der Anwendungsproblem aus Ingenierwissenschaft oder Physik, Chemie so, daß die Gleichungssysteme eine spezielle Struktur haben, wie zum Beispiel symmetrische, Hermitesche Matrizen, Bandstruktur etc. Wir betrachten nun zuerst Hermitesch, postitiv definite Systeme

$$Ax = b \ \text{ mit } \ A = A^*, \ \text{ d.h. } a_{ij} = \overline{a}_{ij}, \tag{2.78}$$

$$\text{und } x^* A x > 0, \ \forall x \in \mathbb{C}^n \setminus \{0\}$$

und untersuchen, wie wir den Gauß–Algorithmus verbessern können.

Satz 2.79 (Cholesky Zerlegung)
Sei $A \in \mathbb{K}^{n,n}$ ($\mathbb{K} = \mathbb{C}$ oder $\mathbb{K} = \mathbb{R}$), Hermitesch positiv definit. Dann existiert eine untere $\triangle$–Matrix $G \in \mathbb{K}^{n,n}$ mit positiven Diagonalelementen, so daß

$$A = GG^*.$$

Beweis: Da A positiv definit ist, sind alle Hauptabschnittsmatrizen

$$A(1:k, 1:k),\ k = 1, \ldots, n$$

positiv definit, haben also Determinante $\neq 0$. Also existiert eine eindeutige LR–Zerlegung $A = LR$ mit L untere $\triangle$–Matrix mit 1–Diagonale, und R hat Diagonalelemente > 0 (gleich den Pivots). Also kann man R schreiben als $D\tilde{R}$, wobei $\tilde{R}$ 1–Diagonale hat. Sei $D = \operatorname{diag}(d_1, \ldots, d_n)$. Die Pivots sind gerade die Determinanten der Hauptabschnittsmatrizen, also reell positiv (alle Eigenwerte sind reell, positiv), also ist $d_i > 0,\ \forall i = 1, \ldots, n$. Wir haben $A = LD\tilde{R} = LD^{\frac{1}{2}}D^{\frac{1}{2}}\tilde{R}$. $D^{\frac{1}{2}} = \operatorname{diag}(d_1^{\frac{1}{2}}, \ldots, d_n^{\frac{1}{2}})$.
Also folgt

$$D^{-\frac{1}{2}}L^{-1}AL^{-*}D^{-\frac{1}{2}} = D^{\frac{1}{2}}\tilde{R}L^{-*}D^{-\frac{1}{2}}.$$

Links steht eine Hermitesche Matrix, rechts eine obere $\triangle$–Matrix mit 1–Diagonale. Also muß rechts die Einheitsmatrix stehen. Also gilt

$$D^{\frac{1}{2}}\tilde{R} = D^{\frac{1}{2}}L^* \Longleftrightarrow \tilde{R} = L^*.$$

Setze $G := LD^{\frac{1}{2}}$. □

Damit können wir den Gauß–Algorithmus verbessern und erhalten den Cholesky–Algorithmus.

Algorithmus 2.80 (Cholesky Zerlegung GAXPY–Version)
Input: *Hermitesch positiv definite Matrix $A \in \mathbb{K}^{n,n}$.*
Output: *Untere $\triangle$–Matrix $G \in \mathbb{K}^{n,n}$ mit positiver Diagonale, so daß $A = GG^*$. Für $i \leqslant j$ überschreibt $G(i,j)$ den Wert von $A(i,j)$.*

FOR $j = 1 : n$
  IF $j > 1$
    $A(j:n, j) = A(j:n, j) - A(j:n, 1:j-1) * A(j, 1:j-1)^*$;
  END
  $A(j:n, j) = A(j:n, j)/\sqrt{A(j,j)}$;
END

Kosten: $n^3/3$ flops.
Fehleranalyse: Analog wie bei Gauß–Zerlegung.

Wie im Fall der LR–Zerlegung kann man auch die Cholesky–Zerlegung anders aufbauen:

$$A = \begin{bmatrix} \alpha & v^* \\ v & B \end{bmatrix} = \begin{bmatrix} \beta & 0 \\ v/\beta & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & B - vv^*/\alpha \end{bmatrix} \begin{bmatrix} \beta & v^*/\beta \\ 0 & I_{n-1} \end{bmatrix}$$

$\beta = \sqrt{\alpha}$. Wenn die Cholesky Zerlegung von $B - vv^*/\alpha = G_1G_1^*$ gegeben ist, dann erhält man die Zerlegung, $A = GG^*$ mit

$$G = \begin{bmatrix} \beta & 0 \\ v/\beta & G_1 \end{bmatrix}.$$

Algorithmus 2.81 (Cholesky–Zerlegung: Äußere Produkte Version)

Input: *Hermitesch positiv definite Matrix* $A \in \mathbb{K}^{n,n}$.

Output: *Untere* $\triangle$*–Matrix* $G \in \mathbb{K}^{n,n}$ *mit positiver Diagonale, so daß* $A = GG^*$. *Für* $i \geqslant j$ *überschreibt* $G(i,j)$ *jeweils* $A(i,j)$.

FOR $k = 1 : n$
$A(k,k) = \sqrt{A(k,k)}$;
$A(k+1:n,k) = A(k+1:n,k)/A(k,k)$;
FOR $j = k+1 : n$
$A(j:n,j) = A(j:n,j) - A(j:n,k) * A(j,k)$;
END
END

Kosten: $n^3/3$ flops.
Fehleranalyse Analog wie bei Gauß–Zerlegung.

Beachte: Die j–Schleife berechnet den unteren $\triangle$–Anteil von

$$A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - A(k+1:n,k)A(k+1:n,k)^*$$

Pivotisierung: Um Symmetrie zu erhalten, sollte man nur auf der Diagonalen pivotisieren. Also Permutationen PAP^* verwenden, die die Diagonalelemente vertauschen.

2.2.1 Verallgemeinerung für nicht positiv definite Systeme.

LDL^*–Zerlegung D blockdiagonal mit 1×1 oder 2×2 Blöcken. Hier muß man i.a. permutieren, d.h. man erzeugt $P^*AP = LDL^*$.
$\longrightarrow$ Übung.

Lösung des Gleichungssystems dann wie bei Gauß mit Vorwärts– und Rückwärtseinsetzen.

2.2.2 Bandsysteme

Eine weitere häufig vorkommende spezielle Struktur ist die Bandstruktur. Eine Matrix heißt $p-q$ Bandmatrix mit unterer Bandbreite p und oberer Bandbreite q falls $a_{ij} = 0$ für $i > j+p$ und $j > i+q$.

Beispiel 2.82

$$A = \begin{bmatrix} * & * & * & 0 & 0 \\ * & * & * & * & 0 \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

untere Bandbreite 1
obere Bandbreite 2
1–2 Bandmatrix.

Satz 2.83 $A \in \mathbb{K}^{n,n}$ *habe eine LR–Zerlegung* $A = LR$. *Falls* A *eine* $p-q$*–Bandmatrix ist, so hat* L *untere Bandbreite* p *und* R *obere Bandbreite* q.

Beweis: Mit vollständiger Induktion:

$$A = \begin{bmatrix} \alpha & w^* \\ v & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/\alpha & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & B - vw^*/\alpha \end{bmatrix} \begin{bmatrix} \alpha & w^* \\ 0 & I_{n-1} \end{bmatrix}$$

$B - vw^*/\alpha$ ist $p-q$ Bandmatrix da nur die ersten q Komponenten von w und die ersten p Komponenten von v ungleich 0 sind. Sei L_1R_1 die LR–Zerlegung von $B - vw^*/\alpha$. Nach I.V. und der Sparsität von v, w folgt daß

$$L = \begin{bmatrix} 1 & 0 \\ v/\alpha & L_1 \end{bmatrix} \quad \text{und} \quad R = \begin{bmatrix} \alpha & w^* \\ 0 & R_1 \end{bmatrix}$$

die gewünschten Brandbreiten haben und es gilt $A = LR$. □

! Es folgt also, daß man die Bandstruktur optimal ausnutzen kann vorausgesetzt, die LR–Zerlegung existiert!

Algorithmus 2.84 (Band Gauß–Elimination, äußere Produkte Version)
Input: $A \in \mathbb{K}^{n,n} p-q$*–Bandmatrix, welches eine LR–Zerlegung besitzt.*
Output: *Zerlegung* $A = LR$, $A(i,j)$ *überschrieben durch* $L(i,j)$ *für* $i > j$ *und* $R(i,j)$ *sonst.*

```
FOR   k = 1 : n − 1
      FOR   i = k + 1 : min(k + p, n)
            A(i, k) = A(i, k)/A(k, k);
      END
      FOR   j = k + 1 : min(k + q, n)
            FOR   i = k + 1 : min(k + p, n)
                  A(i, j) = A(i, j) − A(i, k) ∗ A(k, j);
            END
      END
END
```

Kosten: für $n \gg p, q$: $\approx 2npq$ flops
Fehleranalyse: wie normaler Gauß.
Cholesky für Band analog.
$\longrightarrow$ Übung .

Spezielle Speichertechniken möglich. Dann kann man auch den $\triangle$–Löser verbessern.

Algorithmus 2.85 (Band–Vorwärts–Auflösen)
Input: $L \in \mathbb{K}^{n,n}$ *untere* $\triangle$*–Matrix mit 1–Diagonale und unterer Bandbreite* $p, b \in \mathbb{K}^n$.
Output: b *überschrieben mit der Lösung von* $Lx = b$.

```
FOR   j = 1 : n
      FOR   i = j + 1 : min(j + p, n)
            b(i) = b(i) − L(i, j) ∗ b(j);
      END
END
```

Kosten: für $n \gg p$: $\approx 2np$ flops .

Algorithmus 2.86 (Band–Rückwärts–Auflösen)

Input: $R \in \mathbb{K}^{n,n}$ *obere* $\triangle$*–Matrix mit oberer Bandbreite* $q, b \in \mathbb{K}^n$.
Output: b *überschrieben mit der Lösung von* $Rx = b$.

```
FOR   j = n : −1 : 1
      b(j) = b(j)/u(j, j);
      FOR   i = max(1, j − q) : j − 1
            b(i) = b(i) − L(i, j) ∗ b(j);
      END
END
```

Tridiagonalmatrizen werden als 3 Vektoren gespeichert, oder bei Hermitesch positiv definiten Matrizen als 2 Vektoren.

Algorithmus 2.87 (Symmetrisch, positiv definiter tridiagonal Löser)
Input: $A \in \mathbb{R}^{n,n}$ *symmetrisch, positiv definit, tridiagonal,* $b \in \mathbb{R}^n$. A *gespeichert als* $d(1:n), e(1:n-1)$.
Output: *Lösung von* $Ax = b$ *überschrieben auf* b.

```
FOR   k = 2 : n
      t = e(k − 1);
      e(k − 1) = t/d(k − 1);
      d(k) = d(k) − t ∗ e(k − 1);
END
FOR   k = 2 : n
      b(k) = b(k) − e(k − 1) ∗ b(k − 1);
END
b(n) = b(n)/d(n);
FOR   k = n − 1 : −1 : 1
      b(k) = b(k)/d(k) − e(k) ∗ b(k + 1);
END
```

Kosten: $8n$ flops.
$\longrightarrow$ Übung 2.94

2.2.3 Übungen zu Kapitel 2.2

Übung 2.88 *Sei*

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n,n}.$$

Zeige:

1. *A besitzt eine LU–Zerlegung*
2. *partielle Pivotisierung führt zu keiner Änderung.*
3. *Es besteht die Beziehung* $U = DL^\top$. *Was bedeutet das für die LU–Zerlegung?*

Übung 2.89 *Sei* $A \in \mathbb{R}^{n,n}$, *symmetrisch positiv definit,* $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ *und nehme an, dass* $A_{11} \in \mathbb{R}^{k,k}$ *ist. Das Schur–Komplement ist gegeben durch* $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$. *Zeige:*

1. *A läßt sich schreiben als*

$$A = L \begin{pmatrix} A_{11} & O \\ O & S \end{pmatrix} L^\top, \text{ wobei } L = \begin{pmatrix} I & O \\ A_{21}A_{11}^{-1} & I \end{pmatrix}.$$

2. *Ist $M \in \mathbb{R}^{n,m}$ mit rang $M = m$, dann ist $M^\top AM$ symmetrisch positiv definit.*
3. *S ist symmetrisch positiv definit*

Übung 2.90 *Sei $A \in \mathbb{R}^{n,n}$ symmetrisch positv definit und $A = LL^\top$ die Cholesky–Zerlegung von A. Zeige:*

1. *Für $i = 1, \ldots, n$ gilt*

$$l_{ii}^2 \geqslant \min_{x \neq 0} \frac{x^\top Ax}{x^\top x} = \frac{1}{\|L^{-\top}\|_2^2}.$$

2. *Für $i = 1, \ldots, n$ gilt*

$$\|L^\top\|_2^2 = \max_{x \neq 0} \frac{x^\top Ax}{x^\top x} \geqslant l_{ii}^2.$$

3. *Für die Konditionszahl in der 2–Norm gilt*

$$cond_2\,(L^\top) \geqslant \max_{1 \leqslant i,k \leqslant n} \frac{|l_{ii}|}{|l_{kk}|}.$$

Übung 2.91 *Schreibe einen* **MATLAB** *–Algorithmus zur Berechnung der Cholesky–Zerlegung. Vergleiche den Algorithmus bezüglich Rechengenauigkeit und Rechenzeit mit der Funktion 'chol' und dem '\' aus* **MATLAB** *für $n = 10, 20, 30, 40, \ldots, 100$ und zufällig erzeugten positiv definiten Matrizen $A = MDM^T, M = rand(n), D = diag(1, 2, 4, 9, \ldots, n^2)$.*

Übung 2.92 *Sei $A \in \mathbb{R}^{n,n}, A = (a_{ij})_{i,j=1,\ldots,n}$, $A = A^T$. Zeige folgende Äquivalenzen:*

1. *A ist positiv definit, d.h. $x^T Ax > 0$, $\forall x \neq 0$.*
2. *Alle Eigenwerte von A sind positiv.*
3. *Alle Hauptabschnittsmatrizen $(a_{ij})_{i,j=1,\ldots,k}$, $k \leqslant n$ sind positiv definit.*
4. *Alle Hauptabschnittsdeterminanten $\det\left((a_{ij})_{i,j=1,\ldots,k}\right)$, $k \leqslant n$ sind positiv.*

Übung 2.93 *Sei $A \in \mathbb{R}^{n,n}$ und sei $F \in \mathbb{R}^{n,r}$ mit $r \leqslant n$. Zeige:*

1. *Ist A positiv definit (d.h. $x^T Ax > 0$ für alle $x \neq 0$), dann ist $F^T AF$ positiv definit genau dann, wenn F vollen Rang hat.*
2. *Welchen Rang haben Matrizen der Form $\begin{pmatrix} X \\ I_r \end{pmatrix}, \begin{pmatrix} I_r \\ Y \end{pmatrix}$?*
3. *Ist A positiv definit, dann ist auch A^{-1} positiv definit.*

Übung 2.94 *1. Schreibe einen* **MATLAB** *–Algorithmus zur Berechnung der Cholesky–Zerlegung. Vergleiche den Algorithmus bezüglich Rechengenauigkeit und Rechenzeit mit der Funktion 'chol' und dem '\' aus* **MATLAB** *für $n = 10, 20, 30, 40, \ldots, 100$ und zufällig erzeugten positiv definiten Matrizen $A = MDM^T, M = rand(n), D = diag(1, 2, 4, 9, \ldots, n^2)$.*

2. *Schreibe einen* **MATLAB** *–Algorithmus zur Berechnung der Cholesky–Zerlegung für tridiagonale Matrizen. Vergleiche den Algorithmus bezüglich Rechengenauigkeit und Rechenzeit mit der Funktion 'chol' und dem '\' aus* **MATLAB** *für* $n = 10, 20, 30, 40, \ldots, 100$ *anhand der Matrix*

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}.$$

2.3 Householder und Gram–Schmidt–Orthogonalisierung

Wir haben bei der Fehleranalyse von Gauß gesehen, daß die Größe $|\tilde{L}|, |\tilde{R}|$ der berechneten Matrizen $\tilde{L}, \tilde{R}$ in die Abschätzung des Fehlers eingeht (2.38). Nun kann natürlich $|\tilde{L}|$ sehr groß sein, wenn wir ohne Pivotisierung arbeiten. Außerdem haben wir gesehen, daß die Lösung von Gleichungssystemen mit Gauß auch bei Pivotisierung nicht numerisch rückwärts stabil ist, wegen der Wachstumsfaktoren. In diesem Kapitel betrachten wir nun Methoden, die auf Zerlegungen mit orthogonalen (unitären) Matrizen beruhen und bei denen wir numerische Stabilität zeigen können.

2.3.1 Householder Matrizen(Spiegelungen)

Sei $v \in \mathbb{K}^n \backslash \{0\}$. Eine $n \times n$ Matrix der Form

$$P = I - \frac{2vv^*}{v^*v} \tag{2.95}$$

heißt Householder Matrix.

Eine Multiplikation mit P spiegelt einen Vektor x an der Hyperebene $\text{span}(v)^\perp$. Householder Matrizen sind Hermitesch und unitär (im reellen symmetrisch und orthogonal). Sie sind Rang 1 Modifikationen der Identität. Sie werden verwendet, um bestimmte Komponenten eines Vektors zu eliminieren. (Analog wie bei Gauß–Transformationen).

Angenommen $x \in \mathbb{K}^n \setminus \{0\}$ und wir wollen v bestimmen, so daß $Px = ce_1$ (P wie in (2.95))

$$Px = \left(I - \frac{2vv^*}{v^*v}\right) x = x - \left(\frac{2v^*x}{v^*v}\right) v$$

$Px \in \text{span}\{e_1\} \Longrightarrow v \in \text{span}\{x, e_1\}$. Setze $v = x + \alpha e_1$, dann gilt

$$v^*x = x^*x + \alpha x_1, v^*v = x^*x + 2\,\text{Real}\,(\alpha x_1) + |\alpha|^2.$$

Also

$$Px = \left(1 - 2\frac{x^*x + \alpha x_1}{x^*x + 2\,\text{Real}\,(\alpha x_1) + |\alpha|^2}\right) x - 2\alpha\frac{v^*x}{v^*v} e_1$$

Damit der Koeffizient 0 ist, brauchen wir also nur $\alpha = e^{i\varphi}\|x\|_2 = e^{i\varphi}(x^*x)^{\frac{1}{2}}$. Dann gilt

$$v = x + e^{i\varphi}\|x\|_2 e_1 \Longrightarrow Px = \left(I - 2\frac{vv^*}{v^*v}\right) x = -e^{i\varphi}\|x\|_2 e_1.$$

2.3.2 Berechnung der Householder–Matrix

Wichtige Details:

1. Wahl des Vorzeichens. Falls $x \approx \alpha e_1$, dann hat $v = x - \text{sign}(x_1)\|x\|_2 e_1$ sehr kleine Norm $\Longrightarrow$ Auslöschung großer relativer Fehler in $\beta = 2/v^*v$. Wähle daher immer $v = x + \text{sign}(x_1)\|x\|_2 e_1$, dann gilt $\|v\|_2 \geqslant \|x\|_2$ und P ist fast perfekt unitär.
2. Normalisierung von v, auf $v(1) = 1$. Dies vereinfacht eine ganze Menge von Algorithmen, die auf Householder–Vektoren beruhen.

Algorithmus 2.96 (Erzeugung des Householder Vektors)
Input: $x \in \mathbb{K}^n$.
Output: $v \in \mathbb{K}^n$ *mit* $v(1) = 1$, *so daß* $\left(I - \frac{2vv^*}{v^*v}\right) x = \alpha e_1$.

```
FUNCTION   v = HOUSE(x)
           n = LENGTH(x);
           μ = ||x||_2;
           v = x;
           IF    μ ≠ 0
                 β = x(1) + sign(x(1)) * μ;
                 v(2:n) = v(2:n)/β;
           END
           v(1) = 1;
END HOUSE
```

Kosten: $\approx 3n$ flops.
Fehler: Berechnetes $\tilde{P}$ erfüllt $\|\tilde{P} - P\|_2 = \mathcal{O}(\text{ eps })$.

2.3.3 Anwendung von Householder–Matrizen

Bei der Multiplikation mit Householder–Matrizen kann an einigen Stellen die Struktur ausgenutzt werden.

$$PA = \left(I - \frac{2vv^*}{v^*v}\right) A = A + vw^*$$

mit $w = \beta A^* v$ und $\beta = \frac{-2}{v^*v}$.
Matrix–Vektor–Multiplikation und äußere–Produkt Aufdatierung.

Algorithmus 2.97 (Vormultiplikation mit Householder–Matrix)
Input: $A \in \mathbb{K}^{m,n}, v \in \mathbb{K}^m, v(1) = 1$.
Output: A *überschrieben mit* $PA = \left(I - \frac{2vv^*}{v^*v}\right) A$.

```
FUNCTION   A = ROWHOUSE(A, v)
           β = -2/v^* * v;
           w = β * A^* * v;
           A = A + v * w^*;
END ROWHOUSE
```

Kosten: $4mn$ flops.
Fehler: $gl(\tilde{P}A) = P(A + E)$, $\|E\|_2 = \mathcal{O}(\text{ eps }\|A\|_2)$

Algorithmus 2.98 (Nachmultiplikation mit Householder–Matrix)
Input: $A \in \mathbb{K}^{m,n}, v \in \mathbb{K}^n, v(1) = 1.$
Output: A *überschrieben mit* $AP = A\left(I - \frac{2vv^*}{v^*v}\right).$

FUNCTION $A = COLHOUSE(A, v)$
$\beta = -2/v^* * v;$
$w = \beta * A * v;$
$A = A + w * v^*;$
END COLHOUSE

Kosten: $4mn$ flops.
Fehler: $gl(A\tilde{P}) = (A + E)P, \ \|E\|_2 = \mathcal{O}(\text{eps}\,\|A\|_2).$

Diese 3 Algorithmen werden verwendet um bestimmte Teile einer Matrix zu eliminieren.

Beispiel 2.99 *Überschreibe* $A \in \mathbb{R}^{m,n}(m \geqslant n)$ *mit* $B = Q^T A$ *wobei* Q *orthogonal, so daß* $B(j+1:m, j) = 0$ *für ein* j *mit* $1 \leqslant j \leqslant n$. *Sei weiter angenommen, daß* $A(j:m, 1:j-1) = 0$ *und wir wollen den nichtrivialen Teil von* v *in* $A(j+1:m, j)$ *speichern.*

$$\begin{aligned} v(j:m) &= HOUSE(A(j:m, j)); \\ A(j:m, j:n) &= ROWHOUSE(A(j:m, j:n), v(j:m)); \\ A(j+1:m, j) &= v(j+1:m); \end{aligned}$$

Mathematisch gesehen haben wir damit mit der $m \times m$ *Householder–Matrix*

$$P = \begin{bmatrix} I_{j-1} & 0 \\ 0 & \tilde{P} \end{bmatrix} = I - \frac{zvv^T}{v^Tv}$$

multipliziert. Fehleranalyse (Wilkinson) sagt, daß Berechnung und Anwendung von Householder Matrizen numerisch rückwärts stabil sind.

Faktorisierite Speicherung. Analog zur Gauß–Transformationen speichert man meistens die Produkte von P_i's als v_i's ab, wie im Beispiel.

2.3.4 Die QR–Zerlegung.

Die QR–Zerlegung einer Matrix $A \in \mathbb{K}^{m,n}(m \geqslant n)$ ist gegeben durch

$$A = QR$$

mit $Q \in \mathbb{K}^{m,m}$ unitär, $R \in \mathbb{K}^{m,n}$ obere $\triangle$–Matrix $R = \begin{bmatrix} \diagdown & \\ O & \end{bmatrix}$.
Falls A vollen Rang hat, dann bilden die ersten n Spalten von Q eine Orthonormalbasis für Bild (A). Mit Hilfe der QR–Zerlegung kann man also Orthonormalbasen für Mengen von Vektoren bestimmen.

Lösung von Gleichungssystemen.

$$\begin{aligned} Ax &= b \qquad\quad A \in \mathbb{K}^{n,n} \\ QRx &= b \iff c = Q^H b, \ Rx = c \end{aligned}$$

Anwendung von Q^H auf b nur Matrix–Vektor Multiplikation oder in faktorisierter Householder Form, mehrere innere Produkte. $Rx = c$ durch rückwärts Auflösen.

Lösung von Ausgleichproblemen als nächstes:

Wie erhält man eine QR–Zerlegung?
Nacheinander Anwendung von Householder–Matrizen.

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}, \; P_1A = \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{bmatrix}, \; P_2P_1A = \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{bmatrix}, \quad \text{usw.}$$

Algorithmus 2.100 (Householder QR–Zerlegung)

Input: $A \in \mathbb{K}^{m,n}$ *mit* $m \geqslant n$.

Output: *Householder–Matrizen* $P_1, \ldots, P_n$, *so daß für* $Q = P_1 \cdots P_n, Q^H A = R$ *obere* $\triangle$*–Matrix. Der obere* $\triangle$*–Teil von* A *wird durch* R *überschrieben und die Komponenten* $j+1:m$ *des* j*–ten Householdervektors werden auf* $A(j+1:m, j)$, $j < m$ *überschrieben.*

```
FOR   j = 1 : n
      v(j : m) = HOUSE(A(j : m, j));
      A(j : m, j : n) = ROWHOUSE(A(j : m, j : n), v(j : m));
      IF    j < m
            A(j + 1 : m, j) = v(j + 1 : m);
      END
END
```

Kosten: $2n^2(m - \frac{n}{3})$ flops.
Fehleranalyse: $\tilde{Q}^*(A + E) = \tilde{R}$, wobei berechnetes $\tilde{Q}$ exakt unitär und $Q^*\tilde{Q} = I + F$, $\|F\|_2 \approx \text{eps}$, $\|E\|_2 \approx \text{eps}\,\|A\|_2$.
A sieht am Ende des Algorithmus wie folgt aus

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & \cdots & r_{1,n} \\ v_2^{(1)} & r_{22} & & & \vdots \\ v_3^{(1)} & v_3^{(2)} & \ddots & & \vdots \\ \vdots & \vdots & \ddots & r_{n-1,n-1} & \vdots \\ \vdots & \vdots & & v_n^{(n-1)} & r_{n,n} \\ \vdots & \vdots & & \vdots & v_{n+1}^{(n)} \\ \vdots & \vdots & & \vdots & \vdots \\ v_m^{(1)} & v_m^{(2)} & \cdots & v_m^{(n-1)} & v_m^{(n)} \end{bmatrix}$$

Die j–te Komponente von $v^{(j)}$ ist jeweils 1. Falls Q benötigt wird, kostet das

$$4(m^2n - mn^2 + \frac{n^3}{3}) \text{ flops}.$$

2.3.5 Eigenschaften der QR–Zerlegung

Der obige Algorithmus beweist die Existenz.

Satz 2.101 *Ist $A = QR$ eine QR–Zerlegung einer Matrix $A \in \mathbb{K}^{m,n}$ von vollem Rang und $Q = [q_1, \ldots, q_n], A = [a_1, \ldots, a_n]$, dann ist*

$$\text{span}\,\{a_1, \cdots, a_k\} = \text{span}\,\{q_1, \cdots, q_k\},\ k = 1 : n.$$

Für $Q_1 = Q(1 : m, 1 : n), Q_2 = Q(1 : m, n + 1 : m)$ gilt:

$$\begin{aligned} \text{Bild}\,(A) &= \text{Bild}\,(Q_1), \\ \text{Bild}\,(A)^{\perp} &= \text{Bild}\,(Q_2) \end{aligned}$$

und $A = Q_1 R_1$ mit $R_1 = R(1 : n, 1 : n)$.

Beweis:

$$a_k = \sum_{i=1}^{k} r_{ik} q_i \in \text{span}\,\{q_1, \ldots, q_k\}$$

$\Longrightarrow \text{span}\,\{a_1, \ldots, a_k\} \subseteq \text{span}\,\{q_1, \ldots, q_k\}$

$$\text{rank}\,(A) = n \Longrightarrow \dim(\,\text{span}\,\{a_1, \ldots, a_k\}) = k, \forall k \leqslant n$$

$\Longrightarrow \text{span}\,\{a_1, \ldots, a_k\} = \text{span}\,(q_1, \ldots, q_k)$.
Rest trivial. □

Satz 2.102 *$A \in \mathbb{K}^{m,n}$ habe vollen Rang. Die „dünne“ QR–Zerlegung*

$$A = Q_1 R_1 \text{ mit } Q_1 \in \mathbb{K}^{m,n}, R \in \mathbb{K}^{n,n}$$

obere $\triangle$–Matrix $r_{ii} \in \mathbb{R}, r_{ii} > 0$, ist eindeutig. $G = R_1^*$ ist der Cholesky Faktor von A^*A.

Beweis: $A^*A = (Q_1 R_1)^*(Q_1 R_1) = R_1^* R_1$.
R_1 ist eindeutig und da A vollen Rang hat auch $Q_1 = AR_1^{-1}$. □

2.3.6 Gram–Schmidt Orthogonalisierung

Klassischer Gram–Schmidt (CGS): Die $Q_1 R_1$ Zerlegung wie in Satz 2.102 kann man natürlich auch über Gram–Schmidt erhalten. Sei $\text{rank}\,(A) = n$. Der klassische Gram–Schmidt ist dann gegeben durch

$$\begin{aligned} r_{ik} &= q_i^* a_k,\ i = 1 : k - 1 \\ q_k &= (a_k - \sum_{i=1}^{k-1} r_{ik} q_i)/r_{kk}. \end{aligned}$$

Numerisch gesehen ist dieser Algorithmus sehr schlecht, da die Orthogonalität der q_k durch Rundungsfehler zerstört wird. Eine leichte Umsortierung ergibt den sogenannten modifizierten

Gram–Schmidt (MGS), der numerisch wesentlich besser ist.
Definiere $A^{(k)} \in \mathbb{K}^{m,n-k+1}$ durch

$$A - \sum_{i=1}^{k-1} q_i r_i^* = \sum_{i=k}^{n} q_i r_i^* = [0\ A^{(k)}].$$

Mit $A^{(k)} = [\underbrace{z}_{1}\ \underbrace{B}_{n-k}]$ gilt: $r_{kk} = \|z\|_2,\ q_k = z/r_{kk}$ und $[r_{k,k+1}, \dots r_{k,n}] = q_k^* B$.
Damit können wir

$$A^{(k+1)} = B - q_k [r_{k,k+1}, \dots, r_{k,n}]$$

als äußeres Produkt bestimmen und weitermachen.

Algorithmus 2.103 (Modifizierter Gram–Schmidt)

Input: $A \in \mathbb{K}^{m,n}$ *Rang* $(A) = n$.
Output: *Zerlegung* $A = Q_1 R_1, Q_1 \in \mathbb{K}^{m,n}$ *mit orthonormalen Spalten,* $R_1 \in \mathbb{K}^{n,n}$ *obere* $\triangle$*–Matrix.*

```
FOR  k = 1 : n
     R(k,k) = ‖A(1:m,k)‖_2;
     Q(1:m,k) = A(1:m,k)/R(k,k);
     FOR  j = k+1 : n
          R(k,j) = Q(1:m,k)^* * A(1:m,j);
          A(1:m,j) = A(1:m,j) − Q(1:m,k) * R(k,j);
     END
END
```

Kosten: $2mn^2$ flops.
Fehler: $\tilde{Q}_1^* \tilde{Q}_1 = I + E, \|E\|_2 \approx \text{eps}\, \kappa_2(A)$
Zum Vergleich bei Householder $\|E\|_2 = \text{eps}$.

Kostenvergleich: Zur Berechnung einer Orthonormalbasis für Bild(A) ist der modifizierte Gram–Schmidt schneller als QR.

2.3.7 Übungen zu Kapitel 2.3

Übung 2.104 *Zeige:*

1. *Für jedes* $V \in \mathbb{R}^{n,m}$, *Rang* $V = m$ *ist* $S = I - 2V(V^TV)^{-1}V^T$ *symmetrisch und orthogonal und es ist* $S^2 = I$.
2. S *ist eine Spiegelung am orthogonalen Komplement von Bild* V.
3. S *hat nur Eigenwerte -1* m*–fach, 1* $n - m$*–fach.*
4. $P = I - V(V^TV)^{-1}V^T$ *ist symmetrisch und erfüllt* $P^2 = P$.
5. P *ist eine Projektion auf das orthogonalen Komplement von Bild* V.
6. P *hat nur Eigenwerte 0* m*–fach, 1* $n - m$*–fach.*
7. *Jede orthogonale Matrix obere Dreiecksmatrix ist bereits eine Diagonalmatrix.*
8. *Jede orthogonale Matrix läßt sich als Produkt von Elementarspiegelungen* S *(d.h.* $m = 1$ *in (a)) schreiben.*

Übung 2.105 *Sei* $G = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$ *orthogonal und nehme an, dass wir für vorgegebenes* $v = \binom{v_1}{v_2}$, c, s *so bestimmen können, dass* $Gv = \binom{*}{0}$ *ist. Zeige:*

(a) Ist A eine obere Hessenbergmatrix (d.h. $a_{i,j} = 0$ *für alle* $i > j + 1$*), dann besitzt A eine QR–Zerlegung, die in* $3n^2 + \mathcal{O}(n)$ flops *berechnet werden kann (ohne explitzite Berechnung von Q). Welches Belegungsmuster hat Q?*

(b) Ist A eine beliebige $n \times n$*–Matrix, dann braucht die QR–Zerlegung mit Hilfe von G,* $2n^3 + \mathcal{O}(n^2)$ flops *(ohne explitzite Berechnung von Q).*
Ist das besser oder schlechter als die QR–Zerlegung mit Hilfe von Householder–Transformationen?

Übung 2.106 *Zeige:*

(a) Ist eine orthogonale Matrix gleichzeitig eine obere Dreiecksmatrix, dann ist sie bereits diagonal. Wie sehen die Diagonaleinträge aus? Gilt ein analoges Ergebnis auch für unitäre Matrizen?

(b) Jede orthogonale Matrix lässt sich als Produkt von Elementarspiegelungen (Householdertransformationen) schreiben. Gilt das auch, wenn man Elementarspiegelungen durch Rotationen ersetzt?

Übung 2.107 *Betrachte für* $c > 0, s = \sqrt{1 - c^2}$ *die Matrix*

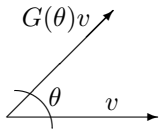
$$T_n = \operatorname{diag}\left(1, s, \ldots, s^{n-1}\right) \begin{pmatrix} 1 & -c & \cdots & -c \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -c \\ 0 & \cdots & 0 & 1 \end{pmatrix} \in \mathbb{R}^{n,n}.$$

Zeige:

1. $\max_{1 \leqslant k \leqslant n} \|T_n(:,k)\|_2 = \|T_n(:,1)\|_2$.
2. $\sigma_n(T_n) \equiv \min_{\|x\|_2 = 1} \|T_n x\|_2 \leqslant \frac{s^{n-1}}{c(c+1)^{n-2}}$.
3. *Was bedeutet das für den QR–Algorithmus mit Spaltenpivotisierung (d.h. in der Restmatrix wird die Spalte mit maximaler 2–Norm mit der führenden Spalte vertauscht) und für die Rangbestimmung von* T_n*? (Betrachte z.B.* $n = 100, c = 0.2$*)*

Übung 2.108 *Sei* $G(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \equiv \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$. *Zeige:*

1. *die Abbildung G beschreibt eine Drehung um den Winkel* θ *gegen den Uhrzeigersinn in der Ebene.*



2. *Bestimme für vorgegebenes* $v = \binom{v_1}{v_2}$, c, s *so, dass* $Gv = \binom{*}{0}$ *ist. Wie müssen* c, s *bestimmt werden, damit ein Höchstmaß an numerischer Stabilität gesichert ist? Begründung!*

Übung 2.109 *Schreibe einen* **MATLAB** *–Algorithmus zur Berechnung der QR–Zerlegung mittels Householder–Transformationen.*
Vergleiche den Algorithmus bezüglich Rechengenauigkeit und Rechenzeit mit der Funktion 'qr' aus **MATLAB** *für* $n = 10, 20, 30, 40, \ldots, 100$ *und zufällig erzeugten Matrizen*

Übung 2.110 *Seien $A \in \mathbb{R}^{n,m}, v, w \in \mathbb{R}^n$. Vergleiche den Rechenaufwand folgender Algorithmen zur Berechnung von $B = (I - vw^T)A$.*

1. $P := I - vw^T, B := PA$
2. $z := w^T A, B := A - vz$

Unterscheidet sich der Rechenaufwand wesentlich?

Übung 2.111 *Sei $Q = [q_1, \ldots, q_n] \in \mathbb{R}^{m,n}$ mit $Q^T Q = I$.*

1. *Was bedeutet das für $q_1, \ldots, q_n$?*
2. *Zeige:*
$$I - QQ^T = \prod_{i=1}^{n} (I - q_i q_i^T),$$
dabei ist die Reihenfolge, in der das Produkt gebildet wird, beliebig.
3. *Sei $a \in \mathbb{R}^n$. Zeige $(I - QQ^T)a$ steht senkrecht auf $q_1, \ldots, q_n$*
4. *Berechne $b = (I - QQ^T)a$ auf zwei Weisen*

   1.
   ```
   for j = 1 : n
       R_j  :=  q_j^T a
   end
   b := a
   for j = 1 : n
       b  :=  b − q_j ∗ R_j
   end
   ```

   2.
   ```
   b := a
   for j = 1 : n
       R_j  :=  q_j^T b
       b    :=  b − q_j ∗ R_j
   end
   ```

   Nach einer etwaigen Normierung von b am Ende, welcher der beiden Algorithmen entspricht dem klassischen und welcher dem modifizierten Gram-Schmidt-Verfahren. Wie kann man das hinsichtlich Teil (b) interpretieren?

Übung 2.112 *Seien A und $B = QA$ gegeben, wobei Q orthogonal. Nehme an, dass $\nu_k = \| (a_{i,k})_{i=1,\ldots,m} \|_2^2$, für $k = 1, \ldots, n$ gegeben ist. Wie lassen sich $\nu'_k = \| (b_{i,k})_{i=2,\ldots,m} \|_2^2$, für $k = 2, \ldots, n$ möglichst schnell aus $\nu_2, \ldots, \nu_n$ berechnen? Welche Auswirkungen hat das auf Anzahl* flops *bei der QRP–Zerlegung?*

Übung 2.113 *Schreibe einen* **MATLAB** *–Algorithmus zur Berechnung*

1. *der QRP–Zerlegung (Spaltenpivotisierung) mittels Householder–Transformationen (hierzu wird in jedem Schritt vor der Elimination die Spalte mit der größten 2–Norm in die führende Position gebracht. D.h., ist $A(j:m, j:n)$ die Restmatrix im j–ten Schritt, so berechne μ mit $\|A(:,\mu)\|_2 = \max_{j \leqslant l \leqslant n} \|A(:,l)\|_2$ und vertausche $A(:,j)$ mit $A(:,\mu)$),*
2. *der QR–Zerlegung mittels klassischem Gram–Schmidt Verfahren.*
3. *der QR–Zerlegung mittels modifiziertem Gram–Schmidt Verfahren.*

Untersuche die Algorithmen für das lineare Ausgleichsproblem $\|Ax - b\|_2 \stackrel{!}{=} \min$ *und vergleiche sie bezüglich Rechengenauigkeit und Rechenzeit mit mit dem Cholesky–Verfahren für die Normalengleichungen* $A^T Ax = A^T b$. *Zur Generieung der Beispielaufgaben kann auf ein entsprechendes* **MATLAB** *-Scriptfile im WWW zurückgegriffen werden.*

Übung 2.114 *Sei*

$$A = \begin{pmatrix} * & & & & & & & * \\ * & & * & & & & & \\ * & & & & & * & & \\ * & & * & & & & & \\ * & & & & * & & & \\ * & * & & & & & & \\ * & & & * & & & & \\ * & * & * & * & * & * & * & * \end{pmatrix}$$

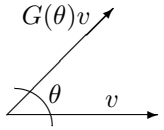
Bestimme Permutationen U, V, *so dass bei einer QR–Zerlegung von* UAV *möglichst wenig Einträge in* R *entstehen.*

Übung 2.115 *Zeige:*

1. *Für jedes* $V \in \mathbb{C}^{n,m}$, *Rang* $V = m$ *ist* $S = I - 2V(V^H V)^{-1} V^H$ *Hermitesch und unitär und es ist* $S^2 = I$.
2. S *ist eine Spiegelung am orthogonalen Komplement von Bild* V.
3. S *hat nur die Eigenwerte -1 m–fach, 1 n − m–fach.*
4. *Sei* $x \in \mathbb{C}^n \setminus \{0\}$ *und schreibe die erste Komponente* x_1 *von* x *in Polarkoordinaten als* $x_1 = |x_1|\ e^{i\theta}$. *Setze* $v = x + e^{i\theta}\|x\|_2\ e_1$. *Dann ist* $S = I - \frac{2}{v^H v} v v^H$ *eine Spiegelung mit* $Sx = -e^{i\theta}\|x\|_2\ e_1$.

Übung 2.116 *Sei* $G(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \equiv \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$. *Zeige:*

1. *die Abbildung* G *beschreibt eine Drehung um den Winkel* θ *gegen den Uhrzeigersinn in der Ebene.*



2. *Für vorgegebenes* $v = \binom{v_1}{v_2}$ *bewirkt die Wahl*

$$c = \frac{v_1}{\sqrt{v_1^2 + v_2^2}},\ s = \frac{-v_2}{\sqrt{v_1^2 + v_2^2}},$$

dass $Gv = \binom{*}{0}$ *ist (Givens–Rotation). Wie sollten* c, s *berechnet werden, um ein Höchstmaß an numerischer Stabilität zu sichern? Begründung!*

3. *Sei* $v \in \mathbb{R}^n$. *Dann lässt sich* v *durch Anwendung von* $n - 1$ *eingebetteten Givensrotationen aus Teil 2 auf ein Vielfaches von* e_1 *transformieren.*

Übung 2.117 *Schreibe einen* **MATLAB** *–Algorithmus zur Berechnung der QR–Zerlegung mittels Givens–Rotationen aus Aufgabe 2.*
Vergleiche den Algorithmus bezüglich Rechengenauigkeit und Rechenzeit mit der Funktion 'qr' aus **MATLAB** *für* $n = 10, 20, 30, 40, \ldots, 100$ *und zufällig erzeugten Matrizen*

Übung 2.118 *Sei* $G = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$ *orthogonal und nehme an, dass wir für vorgegebenes* $v = \binom{v_1}{v_2}$, c, s *so bestimmen können, dass* $Gv = \binom{*}{0}$ *ist. Zeige:*
Ist A eine obere Hessenbergmatrix (d.h. $a_{i,j} = 0$ *für alle* $i > j+1$*), dann besitzt A eine QR–Zerlegung, die in* $3n^2 + \mathcal{O}(n)$ flops *berechnet werden kann (ohne explitzite Berechnung von Q).*
Welches Belegungsmuster hat Q?

2.4 Ausgleichsprobleme

Bei vielen wissenschaftlichen Versuchen geht es darum, aus Messungen die Werte von Konstanten $x_1, \ldots, x_n$ zu bestimmen. Oft kann man x_i nicht direkt messen sondern nur eine leichter zugängliche Größe y, die als Funktion von x und anderen Parametern z abhängt, $y = f(z, x_1, \ldots, x_n)$.

Beispiel 2.119 *Bestimmung von g.*
Fallversuche $h = g\frac{t^2}{2}$, *h – Fallhöhe, t – Fallzeit, g – Gravitation.*

Um x_i *zu bestimmen, führe* $m > n$ *verschiedene Experimente durch und erhalte*

$$y_k = f(z_k, x_1, \ldots, x_n),\ k = 1, \ldots, m.$$

Dies ergibt überbestimmtes Gleichungssystem. Dies ist i.a. nicht lösbar. Daher nur beste Approximation möglich. Zum Beispiel minimiere:

$$\sum_{k=1}^{m} \left(y_k - f(z_k, x_1, \ldots, x_n)\right)^2, \quad \textit{kleinste Quadrate (least square)}, \ (\| \bullet \|_2) \tag{2.120}$$

oder

$$\max_{1 \leqslant k \leqslant m} \left|y_k - f(z_k, x_1, \ldots, x_n)\right|, \ (\| \bullet \|_\infty).$$

für (2.120) ergibt sich die notwendige Bedingung für Minimierung durch

$$\frac{\partial}{\partial x_i} \left(\sum_{k=1}^{m} (y_k - f_k(x_1, \ldots, x_n))^2 \right) = 0,\ i = 1 : n, \tag{2.121}$$

wobei $f_k(x_1, \ldots, x_n) := f(z_k, x_1, \ldots, x_n)$.

Dies sind die sogenannten Normalengleichungen. Ein wichtiger Spezialfall, das lineare Ausgleichsproblem, liegt vor falls $f_k(x_1, \ldots, x_n)$ linear in $x_1, \ldots, x_n$, d.h.

$$\begin{bmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{bmatrix} = Ax,\ A \in \mathbb{K}^{m,n}. \tag{2.122}$$

Dann sind die Normalengleichungen

$$\operatorname{grad}_x \left((y - Ax)^*(y - Ax)\right) = 2A^*Ax - 2A^*y = 0. \tag{2.123}$$

Also man hat das lineare Gleichungssystem

$$A^*Ax = A^*y \tag{2.124}$$

zur Lösung von

$$\min_{x \in \mathbb{K}^n} \|Ax - y\|_2. \tag{2.125}$$

Satz 2.126 *Das lineare Ausgleichsproblem (2.125) hat mindestens eine Lösung x_0. Ist x_1 eine weitere Lösung, so gilt $Ax_0 = Ax_1$. Das Residuum $r = y - Ax_0$ ist eindeutig bestimmt und erfüllt $A^*r = 0$. Jede Lösung x_0 erfüllt auch (2.124) und umgekehrt.*

Beweis: $\mathbb{K}^m = \underbrace{\text{Bild}\,(A)}_{L} \oplus \underbrace{\text{Bild}\,(A)^\perp}_{L^\perp}$.
Daher gibt es eine eindeutige Zerlegung

$$y = s + r, \text{ mit } s \in L, r \in L^\perp \tag{2.127}$$

und es gibt ein x_0 mit $A^*x_0 = s$. Da $A^*r = 0$, so folgt $A^*y = A^*s = A^*Ax_0$. Also löst x_0 (2.124) die Normalengleichungen.
Umgekehrt sei x_1 Lösung von (2.124). Dann hat y die Zerlegung $y = s + r$ mit $s = Ax_1, r = y - Ax_1,\ s \in L, r \in L^\perp$. Wegen der Eindeutigkeit der Zerlegung folgt $Ax_1 = Ax_0$.
Weiter erfüllt jede Lösung x_0 von (2.124) das Ausgleichsproblem (2.125). Denn für beliebiges x setze $z = Ax - Ax_0,\ r = y - Ax_0$, dann gilt wegen $r^*z = 0$:

$$\|y - Ax\|_2^2 = \|r - z\|_2^2 = \|r\|_2^2 + \|z\|_2^2 \geqslant \|r\|_2^2 = \|y - Ax_0\|_2^2.$$

□

Falls A vollen Rang hat, so ist A^*A positiv definit. Nun könnte man einfach folgenden Algorithmus verwenden:

Algorithmus 2.128 (Normalengleichung)
Input: $A \in \mathbb{K}^{m,n}$ *mit* $\text{rank}\,(A) = n$, *und* $b \in \mathbb{K}^m$.
Output: *Lösung x des Minimierungsproblems (2.125).*

Berechne das untere Dreieck von $C = A^*A$.
$d = A^*b$.
Berechne Cholesky Zerlegung von $C = GG^*$.
Löse $Gy = d$ und $G^*x = y$.

Kosten: $(m + n/3)n^2$ flops.
Fehleranalyse: Angenommen, keine Fehler bei $C = A^*A$ und $d = A^*b$. Dann gilt wegen der Analyse der Cholesky Zerlegung:

$$(A^*A + E)\tilde{x} = A^*b,$$

mit $\|E\|_2 \approx \text{eps}\,\|A^*\|_2\|A\|_2 \approx \text{eps}\,\|A^*A\|_2$., d.h. wir erwarten

$$\frac{\|x - \tilde{x}\|_2}{\|x\|_2} \approx \text{eps}\,\kappa_2(A^*A) = \text{eps}\,\kappa_2(A)^2,$$

! d.h. das Quadrat der Kondition geht ein. **Schlecht!**

Beispiel 2.129

$$A = \begin{bmatrix} 1 & 1 \\ 10^{-3} & 0 \\ 0 & 10^{-3} \end{bmatrix},\ b = \begin{bmatrix} 2 \\ 10^{-3} \\ 10^{-3} \end{bmatrix},\ x = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ \kappa_2(A) = 1.4 \cdot 10^3.$$

Mit 6–stelliger Arithmetik ist $A^*A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ *singulär. Mit 7–stelliger Arithmetik folgt*

$$\tilde{x} = \begin{bmatrix} 2.00001 \\ 0 \end{bmatrix} \Longrightarrow \frac{\|\tilde{x} - x\|_2}{\|x\|_2} \approx \underbrace{\text{eps}}_{10^{-6}}\underbrace{\kappa_2(A)^2}_{10^6}.$$

Normalengleichung kann also zu sehr schlechten Ergebnissen führen. Verwende daher QR–Zerlegung. Bilde

$$\begin{aligned} Q^*A &= R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix}, \ R_1 \text{ obere } \triangle\text{–Matrix} \\ Q^*b &= \begin{bmatrix} c \\ d \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix} \end{aligned}$$

Dann gilt

$$\|Ax-b\|_2^2 = \|Q^*Ax - Q^*b\|_2^2 = \|R_1x - c\|_2^2 + \|d\|_2^2,$$

da Q unitär, für alle $x \in \mathbb{K}^n$. Falls $\operatorname{rank}(A) = \operatorname{rank}(R_1) = n$, so gilt für die Lösung $R_1x = c$. Also hat man:

Algorithmus 2.130 (QR Lösung des Ausgleichsproblem)
Input: $A \in \mathbb{K}^{m,n}$ *mit* $\operatorname{rank}(A) = n$, *und* $b \in \mathbb{K}^m$.
Output: $x \in \mathbb{K}^n$, *so daß* $\|Ax-b\|_2 \stackrel{!}{=} \min$.

Verwende Algorithmus 2.100, um A mit seiner QR–Zerlegung zu überschreiben.
FOR $j = 1:n$
$v(j) = 1;$
$v(j+1:m) = A(j+1:m, j);$
$b(j:m) = ROWHOUSE(b(j:m), v(j:m));$
END
Löse $R(1:n, 1:n)x = b(1:n)$ *mit Rückwärts–Einsetzen.*

Kosten: $2n^2(m - \frac{n}{3})$ flops.
Fehleranalyse: Das berechnete $\tilde{x}$ löst $\min \|(A+\delta A)x - (b+\delta b)\|_2$ wobei

$$\begin{aligned} \|\delta A\|_F &\leqslant (6m-3n+41)n \operatorname{eps} \|A\|_F + \mathcal{O}(\operatorname{eps}^2) \\ \|\delta b\|_2 &\leqslant (6m-3n+40)n \operatorname{eps} \|b\|_2 + \mathcal{O}(\operatorname{eps}^2) \end{aligned}$$

! **Vorsicht!** Probleme tauchen auf, wenn $\kappa_2(A) = \kappa_2(R) \approx \frac{1}{\text{eps}}$, oder natürlich wenn $\operatorname{Rang}(A) < n$.

Abhilfe schafft hier die sogenannte Singulärwertzerlegung $A = U\Sigma V^*$ mit U, V unitär, $\Sigma = \left[\begin{array}{ccc} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \\ \hline & 0 & \end{array}\right]$. Dies kann zu diesem Zeitpunkt hier nicht gemacht werden.

2.5 Iterative Verfahren

Für viele der Gleichungssysteme, die in der Praxis auftauchen, gilt $n \gg 10000$ und typischerweise $a_{ij} = 0$ für $> 90\%$ der Einträge. Wenn man derartige Probleme mit Gauß oder QR löst, kann es passieren, das in den Faktoren L, Q, R alle Einträge $\neq 0$ sind. Extrembeispiel Gauß für die Matrix

$$\begin{bmatrix} * & * & \cdots & * \\ * & * & & 0 \\ \vdots & & \ddots & \\ * & 0 & & * \end{bmatrix}.$$

Die Matrix läuft total voll. Es gibt sparse Varianten von Gauß/Cholesky/QR. Dies ist Thema einer Spezialvorlesung. Eine wichtige Idee, die insbesondere im Zusammenhang mit Vektorrechnern und Parallelrechnern von größter Bedeutung ist, ist die Konstruktion von Verfahren, die die Matrix an sich nicht verändern, und wenn möglich auf Matrix * Matrix oder Matrix * Vektor–Operationen aufgebaut sind. Das führt auf iterative Verfahren. Im wesentlichen betrachten wir hier 2 Ansätze.

2.5.1 Splitting–Verfahren

Zerlege A additiv

$$A = M - N, \tag{2.131}$$

wobei M^{-1} eine Approximation an A^{-1} ist, die leicht invertierbar ist, z.B. diagonal, blockdiagonal, $\triangle$–Matrix,... Ersetze $Ax = b$ durch

$$Mx = Nx + b \Longleftrightarrow x = M^{-1}Nx + M^{-1}b. \tag{2.132}$$

Dies ist eine Fixpunktgleichung.

Kanonisches Fixpunktverfahren:

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b, \text{ mit gegebenen } x^{(0)}. \tag{2.133}$$

Satz 2.134 *Das Fixpunktverfahren (2.133) konvergiert für alle Startwerte $x^{(0)}$ gegen die Lösung von $Ax = b$ genau dann, wenn der Spektralradius $\rho(M^{-1}N) < 1$ ist, wobei $\rho(T) = \max\{|\lambda| : \lambda \text{ Eigenwert von } T\}$.*

Beweis: Banachscher Fixpunktsatz oder:
Sei $e^{(k)} = x - x^{(k)}$ der Fehler in der k–ten Iterierten, so gilt:

$$\begin{aligned} e^{(k+1)} &= x - x^{(k+1)} \\ &= (M^{-1}Nx + M^{-1}b) - (M^{-1}Nx^{(k)} + M^{-1}b) \\ &= M^{-1}N(x - x^{(k)}) \\ &= M^{-1}Ne^{(k)} \\ &= (M^{-1}N)^k e^{(0)}. \end{aligned}$$

Wenn Konvergenz für alle $x^{(0)}$ (also auch für alle $e^{(0)}$) gelten soll, dann ist dies äquivalent zu $\lim\limits_{k\to\infty}(M^{-1}N)^k = 0$.
Sei $M^{-1}N = PJP^{-1}$ die Transformation auf Jordan–Normalform, d.h.

$$J = \begin{bmatrix} J_1 & & \\ & \ddots & \\ & & J_k \end{bmatrix}, \; J_i = \begin{bmatrix} \lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix},$$

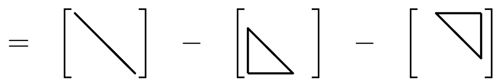
so gilt $\lim\limits_{k\to\infty}(M^{-1}N)^k = 0 \Longleftrightarrow \lim\limits_{k\to\infty} J^k = 0$.
J ist obere $\triangle$–Matrix mit Eigenwerten auf der Diagonale.

Also muß $(\lambda_i)^k \longrightarrow 0 \Longrightarrow |\lambda_i| < 1 \Longrightarrow \rho(J) < 1 \Longrightarrow \rho(M^{-1}N) < 1$.
Umgekehrt, falls $\rho(J) < 1$, so folgt $\lim\limits_{k\to\infty} J^k = 0$. Man schaue sich dann die Form von J^k an.

□

Mögliche Wahlen für M, N. Setze

$$A \;=\; D \;-\; L \;-\; U$$



D Diagonale, L unteres △, U oberes △.
$M = D,\ N = L + U$ ergibt das Verfahren

$$x^{(k+1)} = D^{-1}(L+U)x^{(k)} + D^{-1}b \tag{2.135}$$

Algorithmus 2.136 (Jacobi oder Gesamtschrittverfahren)

Input: *$A \in \mathbb{K}^{n,n}, b \in \mathbb{K}^n,\ a_{ii} \neq 0, i = 1,\dots,n$ und Startvektor $x^{(0)} = [x_1^{(0)},\dots,x_n^{(0)}]^T$ sowie Abbruchgrenze δ.*

Output: *Näherung $\tilde{x}$ für Lösung $Ax = b$*

FOR $k = 0,1,2,\dots$
 FOR $i = 1:n$

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) / a_{ii};$$

 END
 IF $\|x^{(k+1)} - x^{(k)}\| < \delta$, *STOP*
END

Kosten: im wesentlichen ein Matrix–Vektor Produkt (unter Ausnutzung der Sparsität), 1 Vektoraddition pro Iterationsschritt.
Fehleranalyse: Wie bei Matrix * Vektor, Vektor+Vektor. Verfahrensfehler abhängig von $\delta, \rho(D^{-1}(L+U))$.

$M = D - L,\ N = U$ ergibt das Verfahren

$$x^{(k+1)} = (D-L)^{-1}Ux^{(k)} + (D-L)^{-1}b \tag{2.137}$$

Algorithmus 2.138 (Gauß–Seidel oder Einzelschrittverfahren)

Input: *$A \in \mathbb{K}^{n,n}, b \in \mathbb{K}^n,\ a_{ii} \neq 0, i = 1,\dots,n$ und Startvektor $x^{(0)} = [x_1^{(0)},\dots,x_n^{(0)}]^T$ sowie Abbruchgrenze δ.*

Output: *Näherung $\tilde{x}$ für Lösung $Ax = b$*

FOR $k = 0,1,2\dots$
 FOR $i = 1:n$

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) / a_{ii};$$

 END
 IF $\|x^{(k+1)} - x^{(k)}\| < \delta$, *STOP*
END

Kosten: im wesentlichen wie Jacobi, nur schwerer als Vektoroperation zu implementieren.

Fehleranalyse. Fehler abhängig von Fehler in Lösung von $(D-L)x = c$ sowie Verfahrensfehler abhängig von $\rho((D-L)^{-1}U), \delta$.

Beide Verfahren sind einfach, aber konvergieren i.a. sehr langsam, insbesondere bei den Problemen die in der Praxis auftauchen. In Spezialfällen gilt: Falls Jacobi konvergiert, so konvergiert Gauß–Seidel schneller, falls Jacobi divergiert, so divergiert Gauß–Seidel schneller (Satz von Stein/Rosenberg).

Satz 2.139 (Ostrowski/Reich) *Sei $A \in \mathbb{R}^{n,n}$ symmetrisch positiv definit, so konvergiert das Gauß–Seidel Verfahren für alle $x^{(0)}$.*

Beweis: Siehe Stoer 2, Golub/Van Loan. □

2.5.2 Konvergenzbeschleunigung

Eine Möglichkeit, die Konvergenz zu beschleunigen ist die sukzessive Überrelaxation (SOR).

$M = D - \omega L, N = (1-\omega)D + \omega U, \omega \in (0,2)$, $(M-N) = \omega(D-L-U)$ ergibt Verfahren

$$x^{(k+1)} = (D-\omega L)^{-1}((1-\omega)D + \omega U)x^{(k)} + (D-\omega L)^{-1}\omega b. \tag{2.140}$$

Algorithmus 2.141 (SOR)

Input: $A \in \mathbb{K}^{n,n}, b \in \mathbb{K}^n$, $a_{ii} \neq 0, i = 1, \ldots, n$ *und Startvektor* $x^{(0)} = [x_1^{(0)}, \ldots, x_n^{(0)}]^T$ *sowie Abbruchgrenze* δ.

Output: *Näherung* $\tilde{x}$ *für Lösung* $Ax = b$

FOR $k = 0, 1, 2 \ldots$
  FOR $i = 1 : n$

$$x_i^{(k+1)} = \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii} + (1-\omega) x_i^{(k)};$$

  END
  IF $\|x^{(k+1)} - x^{(k)}\| < \delta$, *STOP*
END

Kosten und Fehler wie oben ($\omega = 1$: Gauß–Seidel).

In einigen Spezialfällen von strukturierten Matrizen aus partiellen Differentialgleichungen gibt es Konvergenzbeweise, optimales ω, etc. Weitere Beschleunigungsmethoden: Tschebyschow–Beschleunigung, semiiterative Verfahren, unvollständige Dreieckszerlegungen.

2.5.3 Konjugierte Gradienten

Grundidee: Betrachte quadratisches Funktional

$$\Phi(x) = \frac{1}{2} x^T A x - x^T b, \; A \in \mathbb{R}^{n,n} \text{ symmetrisch, positiv definit, } b \in \mathbb{R}^n.$$

Minimum von Φ ist der Wert $-b^T A^{-1} b/2$. Dieser wird erreicht bei $x = A^{-1}b$. Also ist Minimierung von $\Phi(x)$ äquivalent zur Lösung von $Ax = b$.

Methode des Steilsten Abstiegs: In jedem Schritt lege die Richtung des steilsten Abstiegs fest und gehe in diese Richtung so weit wie möglich. Beim Punkt x_c angelangt, ist die Richtung des steilsten Abstiegs die Richtung des negativen Gradienten.

$$-\nabla\Phi|_{x_c} = b - Ax_c = r_c, \quad \text{Residium von } x_c. \tag{2.142}$$

Falls $r_c \neq 0$, dann gibt es ein α, so daß $\Phi(x_c + \alpha r_c) < \Phi(x_c)$. Wähle

$$\alpha = \frac{r_c^T r_c}{r_c^T A r_c}. \tag{2.143}$$

Das ergibt Minimum von $\Phi(c_x + \alpha r_c)$ über α. Man hätte damit folgende Methode des steilsten Abstiegs.

$k = 0; x_0 = 0; r_0 = b;$
WHILE $r_k \neq 0$
$k = k + 1;$
$\alpha_k = r_{k-1}^T r_{k-1} / r_{k-1}^T A r_{k-1};$
$x_k = x_{k-1} + \alpha_k r_{k-1};$
$r_k = b - Ax_k;$
END

Sehr langsame Konvergenz wenn $\kappa_2(A)$ groß. Besser ist es, die Suchrichtungen so zu wählen, daß die Residuen senkrecht aufeinanderstehen, so daß man in exakter Rechnung in $\leqslant n$ Schritten fertig ist.

Algorithmus 2.144 (Konjugierte Gradienten–Verfahren)
Input: $A \in \mathbb{R}^{n,n}$, *symmetrisch, positiv definit,* $b \in \mathbb{R}^n$, $k_{\max}$ *(Maximum an Iterationsschritten),* $\varepsilon > 0$ *Toleranz.*
Output: *Lösung von* $Ax = b$.

$k = 0;\ x_0 = 0;\ r_0 = b;\ \rho_0 = \|r_0\|_2^2;$
WHILE $\sqrt{\rho_k} > \varepsilon\sqrt{\rho_0}$ *AND* $k < k_{\max}$
$k = k + 1;$
IF $k = 1$
$p = r;$
ELSE
$\beta = \rho_{k-1}/\rho_{k-2};$ % $(= r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2})$
$p = r + \beta p;$ % $(= p_k)$
END
$w = Ap;$
$\alpha = \rho_{k-1}/p^T w;$ % $(= r_{k-1}^T r_{k-1} / p_k^T A p_k)$
$x = x + \alpha p;$ % $(= x_k)$
$r = r - \alpha w;$ % $(= r_k)$
$\rho_k = \|r\|_2^2;$ % $(= r_k^T r_k)$.
END

Kosten: 1 Matrix * Vektor–Multiplikation + 2 Skalarprodukte + 2 GAXPY pro Schritt.
Fehleranalyse: Rundungsfehler zerstören Konvergenz in n Schritten und Orthogonalität der Residuen. Bemerkung:

- Für die Vektoren $p \equiv p_k$, $r \equiv r_k$ aus Algorithmus 2.144 gilt: $p_k^T A p_l = 0, r_k^T r_l = 0$ für alle $k > l \geqslant 0$.

- Für die Funktion Φ gilt mit $p_1, \ldots, p_k$ aus Algorithmus 2.144:

$$\min_{\alpha_1,\ldots,\alpha_k \in \mathbb{R}} \Phi(x + \sum_{l=1}^{k} \alpha_l p_l) = \min_{\alpha_k \in \mathbb{R}} \left(\min_{\alpha_1,\ldots,\alpha_{k-1} \in \mathbb{R}} \Phi(x + \sum_{l=1}^{k} \alpha_l p_l) \right).$$

$\rightarrow$ Übung 2.153.

Satz 2.145 *Sei $A \in \mathbb{R}^{n,n}$ symmetrisch positiv definit und $b \in \mathbb{R}$. Für die Iterierten in Algorithmus 2.144 gilt:*

$$\begin{aligned} \|x - x_k\|_A &\equiv \sqrt{(x - x_k)^T A (x - x_k)} \\ &\leqslant 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k . \end{aligned} \tag{2.146}$$

Beweis: Siehe Golub/Van Loan. □

Genauigkeit ist oft beser als (2.146) vorhersagt. Je näher $\kappa_2(A)$ an 1, je schneller die Konvergenz. Konvergenzbeschleunigung durch Präkonditionierung.
Löse

$$(C^{-1}AC^{-T})(C^T x) = C^{-1}b \tag{2.147}$$

anstatt $Ax = b$, wobei C nicht singulär und so gewählt, daß $\kappa_2(C^{-1}AC^{-T})$ möglichst nahe an 1 liegt. Best mögliche Wahl:

$$C^{-1} = A^{-\frac{1}{2}} \Longrightarrow \kappa_2(C^{-1}AC^{-T}) = 1.$$

Das ist natürlich genau so schwierig wie ursprüngliches Problem.
In vielen Anwendungen wie etwa Partielle Differentialgleichungen, Finite Element Methoden hat man natürliche Präkonditionierer. Präkonditioniertes Kongugiertes Gradientenverfahren für symmetrisch positiv definite Systeme i.a. das Beste.

2.5.4 Übungen zu Kapitel 2.5

Übung 2.148 *Sei $A \in \mathbb{R}^{n,n}$ und $\varepsilon > 0$. Zeige:*

1. *Für jede nicht singuläre Matrix $X \in \mathbb{R}^{n,n}$ und jede Matrixnorm $\| \bullet \|$ ist $|||A||| := \|X^{-1}AX\|$ wieder eine Matrixnorm.*
2. *Ist $A = \begin{pmatrix} \lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{pmatrix}$, dann existiert zu jedem $\varepsilon > 0$ eine Diagonalmatrix D, so dass $\|D^{-1}AD\|_\infty \leqslant \rho(A) + \varepsilon$.*
3. *Es existiert zu A eine Norm $\| \bullet \|$, so dass $\|A\| \leqslant \rho(A) + \varepsilon$.*
4. *Sei $\rho(A) < 1$. Dann konvergiert für jeden Startwert $x_0 \in \mathbb{R}^n$ die Iteration $x_{k+1} = Ax_k$.*

Übung 2.149 *Eine Matrix $A \in \mathbb{R}^{n,n}$ heisst M–Matrix, falls es eine nichtnegative Matrix $N \in \mathbb{R}^{n,n}$ und ein $s > \rho(N)$ gibt, mit $A = sI - N$ ($\rho(N) = \max\{|\lambda| : \lambda$ Eigenwert von $N\}$ ist der Spektralradius von N). Zeige: A ist nicht singulär und A^{-1} ist nicht negativ.*

Übung 2.150 *Sei*

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n,n}.$$

Zeige

1. *Die Eigenwerte von A sind $\lambda_j = 2 - 2\cos\frac{j\pi}{n+1}$, und die zugehörigen Eigenvektoren lauten $v_j = (v_{ij})_{i=1,\dots,n} = \left(\sin\frac{ij\pi}{n+1}\right)_{i=1,\dots,n}$ für alle $j = 1, \dots, n$.*
   Hinweis: $\sin(x+y) = \sin x \cos y + \cos x \sin y$, $\cos(-x) = \cos x$, $\sin(-x) = -\sin x$.
2. *Für jede mit der Matrixmultiplikation verträgliche Norm $\| \bullet \|$ gilt: $\rho(A) \leqslant \|A\|$.*
3. *In der Standard–Zerlegung $A = D - L - U$ ist das Einzelschrittverfahren definiert durch $M = D - L, N = U$ und die Iterationsmatrix $T_E = M^{-1}N$. Zeige: $T_E = (I + D^{-1}L + \cdots + (D^{-1}L)^{n-1})D^{-1}U$.*
4. *Bestimme/schätze den Spektralradius von Gesamt–/Einzelschrittverfahren. Was kann man über die Konvergenzgeschwindigkeit des cg–Verfahrens sagen?*

Übung 2.151 *Sei $A \in \mathbb{R}^{n,n}$, $A = D - L - U$ Standardzerlegung, d.h. D ist Diagonale von A, $-L$ striktes unteres Dreieck und $-U$ striktes oberes Dreieck von A. Sei $\omega \in \mathbb{R}$ und betrachte die Zerlegung $\omega A = M - N$ mit $M = D - \omega L$, $N = (1-\omega)D + \omega U$.*
Zeige: Ist $\rho(M^{-1}N) < 1$, so ist $0 < \omega < 2$.

Übung 2.152 *Schreibe* **MATLAB** *–Programme für das Jacobi–Verfahren, Gauß–Seidel–Verfahren, sowie das cg–Verfahren ohne Vorkonditionierung. Vergleiche die Verfahren bezüglich Rechenzeit und Iterationszahl anhand des Beispiels $Ax = b$,*

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}, \; b = \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{pmatrix},$$

$x^{(0)} = 0$, $n = 10, 100, 1000$. *Als Abbruchkriterium nehme man $\|r_{k+1}\|_2 = \|b - Ax_{k+1}\|_2 < 10^{-8}$, $k_{\max} = 600$.*

Übung 2.153 *Sei $A \in \mathbb{R}^{n,n}$ symmetrisch, positiv definit und $b, x_0 \in \mathbb{R}^n$. Betrachte das cg–Verfahren ohne Vorkonditionierung aus der Vorlesung (zur besseren Analyse mit Indizes).*

$k = 0$, $r_0 = b - Ax_0$, $\rho_0 = r_0^\top r_0$
while $\sqrt{\rho_k} > \varepsilon\sqrt{\rho_0}$ ***and*** $k < k_{\max}$
  $k = k + 1$
  if $k = 1$
    $p_1 = r_0$
  else
    $\beta_k = \rho_{k-1}/\rho_{k-2}$
    $p_k = r_{k-1} + \beta_k p_{k-1}$
  end
  $w_k = Ap_k$, $\alpha_k = \rho_{k-1}/w_k^\top p_k$
  $x_k = x_{k-1} + \alpha_k p_k$
  $r_k = r_{k-1} - \alpha_k w_k$
  $\rho_k = r_k^\top r_k$
end

Zeige:

1. $p_k^\top A p_l = 0$, *für alle* $k > l \geqslant 0$.
   Tipp: Zeige durch Induktion über k, $r_{k-1}^\top r_{l-1} = 0, p_k^\top A p_l = 0$, *für alle* $k > l > 0$.
2. *Nehme an, es sei* $p_k^\top A p_l = 0$, *für alle* $k > l > 0$, *dann ist*

$$\min_{v_1,\ldots,v_k \in \mathbb{R}} \Phi(x_0 + \sum_{l=1}^{k} p_l v_l) = \min_{v_k \in \mathbb{R}} \left(\min_{v_1,\ldots,v_{k-1} \in \mathbb{R}} \Phi(x_0 + \sum_{l=1}^{k} p_l v_l) \right),$$

   wobei $\Phi(x) = \frac{1}{2} x^\top A x - x^\top b$. *Was passiert mit dem cg–Verfahren nach maximal n Schritten?*

Übung 2.154 *Sei* $A \in \mathbb{R}^{n,n}$ *symmetrisch, positiv definit und* $b \in \mathbb{R}^n$. *Definiere* $\Phi : \mathbb{R}^n \to \mathbb{R}$ *durch* $\Phi(x) = \frac{1}{2} x^\top A x - x^\top b$. *Zeige:*

1. Φ *besitzt ein eindeutiges Minimum in* $x = A^{-1} b$
2. *Sei* $x_0 \in \mathbb{R}^n$ *und* $P = [p_1, \ldots, p_k] \in \mathbb{R}^{n,k}$ *so, dass* $P^\top A P = D = \operatorname{diag}(d_1, \ldots, d_k)$ *mit von Null verschiedenen* $d_1, \ldots, d_k$. *Dann besitzt die Funktion* $\Psi : \mathbb{R}^k \to \mathbb{R}$, $\Psi(v) = \Phi(x_0 + Pv)$ *ein eindeutiges Minimum in* $v = D^{-1} P^\top (b - A x_0) = \left(\frac{p_i^\top (b - A x_0)}{d_i} \right)_{i=1,\ldots,k}$

Übung 2.155 *Sei* $A \in \mathbb{R}^{n,n}$ *symmetrisch, positiv definit und* $b, x_0 \in \mathbb{R}^n$. *Betrachte das cg–Verfahren ohne Vorkonditionierung aus der Vorlesung (zur besseren Analyse mit Indizes).*

$k = 0$, $r_0 = b - A x_0$, $\rho_0 = r_0^\top r_0$
while $\sqrt{\rho_k} > \varepsilon \sqrt{\rho_0}$ ***and*** $k < k_{\max}$
  $k = k + 1$
  if $k = 1$
    $p_1 = r_0$
  else
    $\beta_k = \rho_{k-1} / \rho_{k-2}$
    $p_k = r_{k-1} + \beta_k p_{k-1}$
  end
  $w_k = A p_k$
  $\alpha_k = \rho_{k-1} / w_k^\top p_k$
  $x_k = x_{k-1} + \alpha_k p_k$
  $r_k = r_{k-1} - \alpha_k w_k$
  $\rho_k = r_k^\top r_k$
end

Zeige:

1. $\operatorname{span}\{r_0, \ldots, r_{k-1}\} = \operatorname{span}\{p_1, \ldots, p_k\}$
2. $\operatorname{span}\{r_0, \ldots, r_{k-1}\} = \operatorname{span}\{r_0, A r_0, \ldots, A^{k-1} r_0\}$

für alle $k = 1, \ldots, n$, *sofern der Algorithmus nicht vorher abbricht.*

Kapitel 3

Lösung nichtlinearer Gleichungssysteme

Neben den linearen Gleichungssystemen führen viele Anwendungsprobleme auf die Lösung von nichtlinearen Gleichungssystemen. Gegeben

$$\begin{array}{ccc} f : E & \longrightarrow & F \qquad (\text{z.B. hier } E = F = \mathbb{R}^n) \\ \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} & \longmapsto & \begin{bmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{bmatrix} \end{array}$$

Gesucht: Vektor $[x_1, \cdots, x_n]^T$ so daß $f_i(x_1, \cdots, x_n) = 0$, für alle $i = 1, \cdots, n$.
E, F können auch unendlich dimensionale lineare Räume sein, etwa Räume von Funktionen und f kann auch ein Differentialoperator sein. Hier betrachten wir nur den Fall $E = F = \mathbb{R}^n$. Im allgemeinen ist die Lösung nur durch Näherungsverfahren zu bestimmen. Nullstellenprobleme sind eng verwandt mit Minimierungsproblemen wie

$$\text{Bestimme für } h : \mathbb{R}^n \longrightarrow \mathbb{R} \quad \min_{x \in \mathbb{R}^n} h(x). \tag{3.1}$$

In üblicher Weise muß zur Bestimmung des Minimums

$$\text{grad } (h(x)) = \begin{bmatrix} \frac{\partial h}{\partial x_1} \\ \vdots \\ \frac{\partial h}{\partial x_n} \end{bmatrix} = 0$$

berechnet werden, welches wiederum ein Nullstellenproblem ist. Neben Optimierungsproblemen treten Nullstellenbestimmungen in fast allen Anwendungen auf. Gewöhnliche oder partielle Differentialgleichungen, Ausgleichsrechnung, ...

3.1 Fixpunktverfahren

Wandle das Nullstellenproblem in ein Fixpunktproblem um.

$$x = \Phi(x). \tag{3.2}$$

Dies ist im allgemeinen auf viele Arten möglich, die äquivalent oder nicht äquivalent sein können.

Beispiel 3.3 $e^x - \sin x = 0$
Mögliche Fixpunktgleichungen (nicht alle äquivalent)

$$\begin{array}{lrcl} a) & x & = & e^x - \sin x + x \\ b) & x & = & \sin x - e^x + x \\ c) & x & = & \arcsin(e^x), \quad \textit{für } x < 0 \\ d) & x & = & \ln(\sin x), \quad \textit{für } (-2n\pi, 2n\pi + \pi), n = 1, 2, 3, \ldots \end{array}$$

Alle haben unterschiedliche numerische Eigenschaften.
$\longrightarrow$ *Übung 3.50*

Die Idee des Fixpunktverfahrens ist es, die Folge

$$x^{(i+1)} = \Phi(x^{(i)}), \quad i = 0, 1, 2, \ldots \tag{3.4}$$

mit gegebenem Startwert x_1 zu erzeugen.

Satz 3.5 (Banachscher Fixpunktsatz) *Sei $D \subseteq D_\Phi \subseteq \mathbb{R}^m$ ein abgeschlossenes Gebiet und Φ eine kontrahierende Abbildung von D in sich, d.h.*

a) $\Phi : D \longrightarrow D$

b) Φ *kontrahierend, d.h.* $\exists\, \alpha < 1$ *und eine Norm* $\| \bullet \|$*, so daß*

$$\|\Phi(x) - \Phi(y)\| \leqslant \alpha \|x - y\|, \quad \forall x, y \in D. \tag{3.6}$$

Dann gilt:

i) Es gibt genau einen Fixpunkt $\hat{x}$ von (3.2) in D.

ii) Die Fixpunktiteration (3.4) konvergiert für jeden Startwert $x^{(0)} \in D$ gegen $\hat{x}$.

iii) Es gelten die Fehlerabschätzungen

$$\|\hat{x} - x^{(n)}\| \leqslant \frac{\alpha^n}{1-\alpha} \|x^{(1)} - x^{(0)}\| \quad \textit{a priori} \tag{3.7}$$

$$\|\hat{x} - x^{(n)}\| \leqslant \frac{\alpha}{1-\alpha} \|x^{(n)} - x^{(n-1)}\| \quad \textit{a posteriori} \tag{3.8}$$

Beweis: $x^{(0)} \in D \Longrightarrow x^{(n)} = \Phi(x^{(n-1)}) \in D$, für alle $n = 1, 2, \ldots$

$$\begin{array}{rcl} \|x^{(n+1)} - x^{(n)}\| & = & \|\Phi(x^{(n)}) - \Phi(x^{(n-1)})\| \\ & \leqslant & \alpha \|x^{(n)} - x^{(n-1)}\| \leqslant \alpha^2 \|x^{(n-1)} - x^{(n-2)}\| \\ & \leqslant & \alpha^n \|x^{(1)} - x^{(0)}\| \end{array}$$

$$\begin{array}{rcl} \|x^{(n+k)} - x^{(n)}\| & \leqslant & \displaystyle\sum_{i=1}^{k} \|x^{(n+i)} - x^{(n+i-1)}\| \\ & \leqslant & \displaystyle\sum_{i=1}^{k} \alpha^{n+i-1} \|x^{(1)} - x^{(0)}\| \\ & \leqslant & \alpha^n \|x^{(1)} - x^{(0)}\| \displaystyle\sum_{i=1}^{k} \alpha^{i-1} \\ & \leqslant & \dfrac{\alpha^n}{1-\alpha} \|x^{(1)} - x^{(0)}\| \end{array}$$

$\Longrightarrow$ Folge konvergiert, da Cauchy–Folge und $\mathbb{R}^m$ vollständig.
Φ ist stetig. Dies folgt bereits aus der Kontraktionseigenschaft (3.6). Somit ist der Grenzwert Fixpunkt von Φ, wegen

$$\hat{x} \equiv \lim_{n\mapsto\infty} x^{(n+1)} = \lim_{n\mapsto\infty} \Phi(x^{(n)}) = \Phi(\lim_{n\mapsto\infty} x^{(n)}) = \Phi(\hat{x}).$$

Eindeutigkeit. Seien $\hat{x}, \bar{x}$ Fixpunkte von Φ.

$$\|\hat{x} - \bar{x}\| = \|\Phi(\hat{x}) - \Phi(\bar{x})\| \leqslant \alpha\|\hat{x} - \bar{x}\|.$$

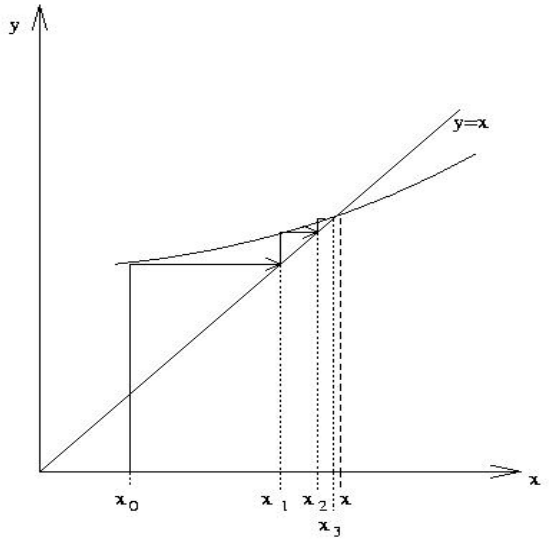
Wegen $\alpha < 1$ kann nur $\hat{x} = \bar{x}$ gelten.

$$\begin{aligned} \|x^{(n+k)} - x^{(n)}\| &\leqslant \frac{\alpha^n}{1-\alpha}\|x^{(1)} - x^{(0)}\| \\ \overset{k\to\infty}{\Longrightarrow} \|\hat{x} - x^{(n)}\| &\leqslant \frac{\alpha^n}{1-\alpha}\|x^{(1)} - x^{(0)}\| \Longleftrightarrow (3.7) \end{aligned}$$

$$n = 1 \Longrightarrow \|\hat{x} - x^{(1)}\| \leqslant \frac{\alpha}{1-\alpha}\|x^{(1)} - x^{(0)}\|$$

ersetze $x^{(1)}$ durch $x^{(n)}, x^{(0)}$ durch $x^{(n-1)}$ so folgt (3.8). □

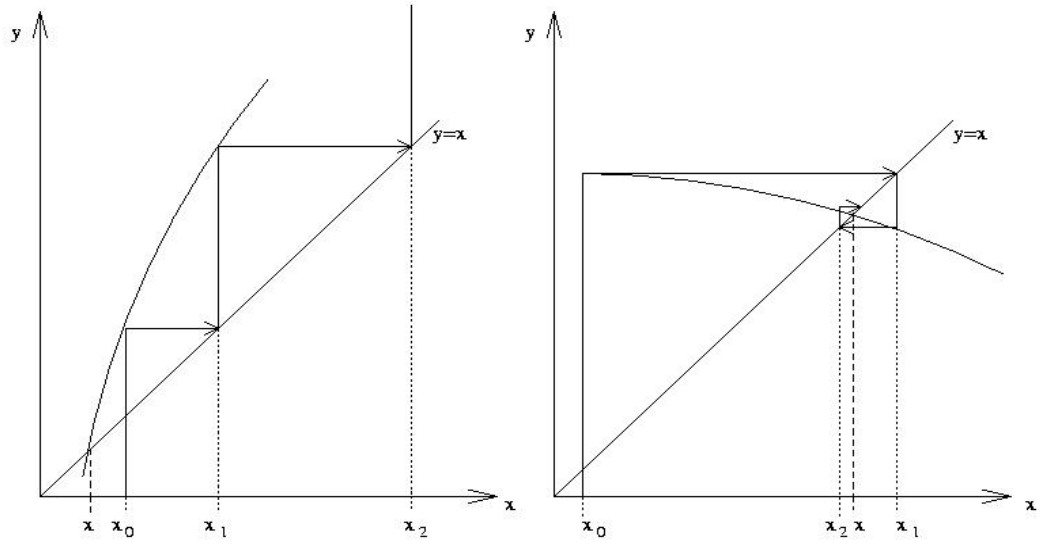
Graphik 3.9 *Graphische Veranschaulichung der Konvergenz $m = 1$.*



Definition 3.10 *Ein Fixpunkt $\hat{x}$ heißt anziehend, wenn die Iteration (3.4) für alle genügend nahe bei $\hat{x}$ gelegenen Startwerte $x^{(0)} \neq \hat{x}$ gegen $\hat{x}$ konvergiert.*

Ein Fixpunkt $\hat{x}$ heißt abstoßend, wenn es eine offene Kugel U_ε um $\hat{x}$ gibt, so daß für alle $x^{(0)} \in U_\varepsilon \setminus \{\hat{x}\}$ ein m existiert, so daß $x^{(m)} \notin U_\varepsilon$.

Graphik 3.11 *Abstoßender Fixpunkt / Anziehender Fixpunkt.*



Satz 3.12 *Sei $D \subseteq D_\Phi \subseteq \mathbb{R}^m$ abgeschlossen und konvex und nehme an, daß*
$D\Phi = \begin{bmatrix} \frac{\partial \Phi_1(x)}{\partial x_1} & \cdots & \frac{\partial \Phi_1(x)}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial \Phi_m(x)}{\partial x_1} & \cdots & \frac{\partial \Phi_m(x)}{\partial x_m} \end{bmatrix}$ *existiert und stetig ist. Falls es eine Norm gibt, so daß* $\|D\Phi(x)\| \leqslant \alpha < 1$, *so ist Φ in D kontrahierend mit Lipschitzkonstante α.*

Beweis: $\Longrightarrow$ Übung 3.49. □

Ist $D\Phi(\hat{x})$ stetig und $\|D\Phi(\hat{x})\| < 1$ so ist $\hat{x}$ anziehend. Ein Fixpunkt ist im allgemeinen nicht anziehend, falls einer der Eigenwerte von $D\Phi(x)$ vom Betrag größer 1 ist.
$\Longrightarrow$ Übung.

Definition 3.13 *Eine Iterationsfolge $\{x^{(i)}\}_{i=0}^{\infty}$, die für $p \geqslant 1$*

$$\|x^{(i+1)} - \xi\| \leqslant c\|x^{(i)} - \xi\|^p \quad i = 0, 1, 2 \cdots \tag{3.14}$$

und $c < 1$ für $p = 1$ erfüllt, heißt Folge von mindestens p–ter Ordnung.

Korollar 3.15 *Das Fixpunktverfahren (3.4) sofern es konvergiert, ist konvergent von mindestens 1. Ordnung (linear konvergent).*

Kosten für Iteration (3.4): eine Auswertung von $\Phi(x)$ pro Iteration, Anzahl der Iterationen aus Fehlerschätzung ablesbar.
Versuche höhere Konvergenzordnung zu bekommen.

3.2 Das Newtonverfahren

Idee: Taylorentwicklung von Nullstelle $\hat{x}$ von $f(x)$, für $x^{(0)}$ aus Umgebung.

$$0 = f(\hat{x}) = f(x^{(0)}) + Df(x^{(0)})(\hat{x} - x^{(0)}) + \underbrace{\mathcal{O}(\|\hat{x} - x^{(0)}\|^2)}_{\text{weglassen}}. \tag{3.16}$$

Ersetze durch

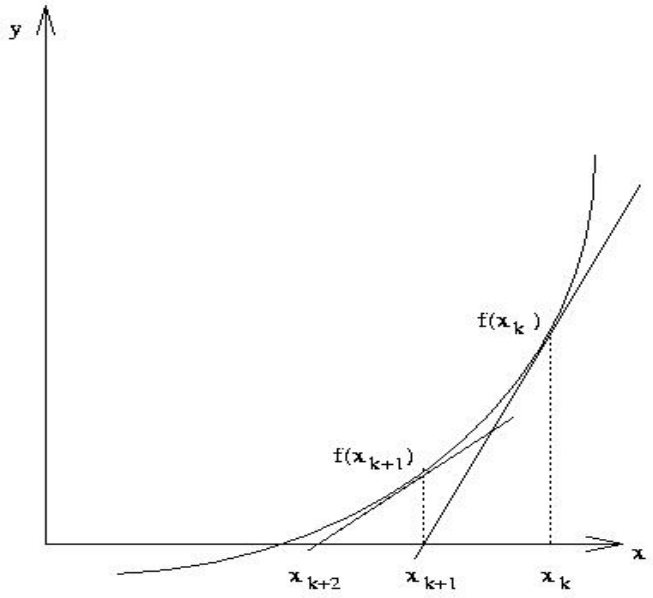
$$0 = f(x^{(0)}) + Df(x^{(0)})(x^{(1)} - x^{(0)}) \tag{3.17}$$

Angenommen $Df(x^{(0)})$ nichtsingulär, so folgt

$$Df(x^{(0)})(x^{(1)} - x^{(0)}) = -f(x^{(0)})$$

hat eindeutige Lösung $\delta^{(1)} = x^{(1)} - x^{(0)}$ und wir erhalten $x^{(1)} = x^{(0)} + \delta^{(1)}$.
$\longrightarrow$ Übung.

Graphik 3.18 *Newtonschritt*



Satz 3.19 *Sei $D \subseteq \mathbb{R}^n$ offen und sei D_0 konvex mit $\bar{D}_0 \subseteq D$. $f : D \to \mathbb{R}^n$ sei für alle $x \in D_0$ differenzierbar und für alle $x \in D$ stetig. Sei $x^{(0)} \in D_0$, so dass es Konstanten $r, \alpha, \beta, \gamma, h$ gibt mit*

$$\begin{aligned} S_r(x^{(0)}) &:= \left\{x|\ \|x - x^{(0)}\| < r\right\} \subseteq D_0 \\ h &= \alpha\beta\gamma/2 < 1 \\ r &= \alpha/(1-h), \end{aligned}$$

und $f(x)$ erfülle

$$\|Df(x) - Df(y)\| \leqslant \gamma\|x - y\|,\ \forall x, y \in D_0. \tag{3.20}$$

$$Df(x)^{-1} \text{ existiere } \forall x \in D_0 \text{ und sei } \|Df(x)^{-1}\| \leqslant \beta,\ \forall x \in D_0, \tag{3.21}$$

sowie

$$\|Df(x^{(0)})^{-1}f(x^{(0)})\| \leqslant \alpha. \tag{3.22}$$

Dann gilt

a) Ausgehend von $x^{(0)}$ *ist jedes* $x^{(i)}$*, welches durch*

$$x^{(i+1)} = x^{(i)} - Df(x^{(i)})^{-1}f(x^{(i)}),\ i = 0, 1, 2, \ldots$$

gegeben ist, wohldefiniert und es ist $x^{(i)} \in S_r(x^{(0)}),\ \forall\, i = 0, 1, 2, \ldots$

b) $\lim\limits_{i\to\infty} x^{(i)} = \hat{x}$ *existiert und es ist* $f(\hat{x}) = 0, \hat{x} \in \overline{S_r(x^{(0)})}$.

c) Für alle $i \geqslant 0$ *gilt* $\|x^{(i)} - \hat{x}\| \leqslant \alpha\dfrac{h^{2^i-1}}{1-h^{2^i}}$

Das Newton–Verfahren ist also mindestens quadratisch konvergent.

Beweis: a) Da $Df(x)^{-1}$ für alle $x \in D_0$ existiert, so ist $x^{(i+1)}$ für alle i dann wohldefiniert, wenn $x^{(i)} \in S_r(x^{(0)})$ für alle $i \geqslant 0$. Wegen Voraussetzung (3.22) gilt dies für $x^{(0)}, x^{(1)}$.
I.V. Sei nun $x^{(j)} \in S_r(x^{(0)})$ für $j = 0, 1, \ldots, k, k \geqslant 1$, so folgt aus (3.21)

$$\begin{aligned}\|x^{(k+1)} - x^{(k)}\| &= \| - Df(x^{(k)})^{-1}f(x^{(k)})\| \leqslant \beta\|f(x^{(k)})\| \\ &= \beta\|f(x^{(k)}) - f(x^{(k-1)}) - Df(x^{(k-1)})(x^{(k)} - x^{(k-1)})\|,\end{aligned}$$

da $f(x^{(k-1)}) + Df(x^{(k-1)})(x^{(k)} - x^{(k-1)}) = 0$.
Wende das folgende Lemma an: □

Lemma 3.23 *$Df(x)$ existiere für alle $x \in D_0, D_0 \subseteq \mathbb{R}^n$, D_0 konvex und es gebe ein γ, so dass*

$$\|Df(x) - Df(y)\| \leqslant \gamma\|x - y\|,\ \forall\, x, y \in D_0.$$

Dann gilt $\forall x, y \in D_0$ die Abschätzung

$$\|f(x) - f(y) - Df(y)(x - y)\| \leqslant \frac{\gamma}{2}\|x - y\|^2$$

Beweis: Sei $\varphi : [0,1] \longrightarrow \mathbb{R}^n$ gegeben durch $\varphi(t) := f(y + t(x - y))$. φ ist differenzierbar $\forall t \in [0,1], \forall x, y \in D_0$ und $\varphi'(t) = Df(y + t(x - y))(x - y)$

$$\Longrightarrow \|\varphi'(t) - \varphi'(0)\| \leqslant \|Df(y + t(x - y)) - Df(y)\|\,\|x - y\| \leqslant \gamma t\|x - y\|^2.$$

Nun ist

$$P \equiv f(x) - f(y) - Df(y)(x - y) = \varphi(1) - \varphi(0) - \varphi'(0) = \int_0^1 (\varphi'(t) - \varphi'(0))\,dt,$$

also folgt

$$\begin{aligned}\|P\| &\leqslant \int_0^1 \|\varphi'(t) - \varphi'(0)\|\,dt \\ &\leqslant \gamma\|x - y\|^2 \int_0^1 t\,dt \\ &\leqslant \frac{\gamma}{2}\|x - y\|^2.\end{aligned}$$

□

Weiter im Beweis von Satz 3.19. Anwendung des Lemma liefert

$$\begin{aligned} \|x^{(k+1)} - x^{(k)}\| &\leqslant \frac{\beta\gamma}{2}\|x^{(k)} - x^{(k-1)}\|^2 \qquad (3.24) \\ \Longrightarrow \|x^{(k+1)} - x^{(k)}\| &\leqslant \alpha h^{2^k-1}. \qquad (3.25) \end{aligned}$$

Diese letzte Abschätzung folgt aus (3.22) für $k = 0$ und mit Induktion für $k + 1$ aus (3.24). Aus (3.25) folgt

$$\begin{aligned} \|x^{(k+1)} - x^{(0)}\| &\leqslant \|x^{(k+1)} - x^{(k)}\| + \cdots + \|x^{(1)} - x^{(0)}\| \\ &\leqslant \alpha(1 + h + h^3 + h^7 + \cdots h^{2^k-1}) < \frac{\alpha}{1-h} = r, \end{aligned}$$

also $x^{(k+1)} \in S_r(x_0)$.

b) Aus (3.25) folgt, dass $x^{(k)}$ Cauchy–Folge ist, denn

$$\begin{aligned} \|x^{(n+k)} - x^{(n)}\| &\leqslant \sum_{i=1}^{k} \|x^{(n+i)} - x^{(n+i-1)}\| \\ &\leqslant \alpha \sum_{i=1}^{k} h^{2^{n+i-1}-1} \\ &\leqslant \alpha h^{2^n-1}(1 + h^{2^n} + (h^{2^n})^2 + \cdots) \\ &\leqslant \frac{\alpha h^{2^n-1}}{1-h^{2^n}} < \varepsilon, \qquad (3.26) \end{aligned}$$

für genügend großes $n \geqslant N(\varepsilon)$, weil $0 < h < 1$.

$$\Longrightarrow \lim_{k\to\infty} x^{(k)} = \hat{x} \in \overline{S_r(x_0)}.$$

Aus (3.26) folgt:

$$\|\hat{x} - x^{(n)}\| = \lim_{m\to\infty} \|x^{(m)} - x^{(n)}\| \leqslant \frac{\alpha h^{2^n-1}}{1-h^{2^n}}.$$

Noch zu zeigen ist, dass $\hat{x}$ Nullstelle von f in $\overline{S_r(x_0)}$ ist. Da $x^{(k)} \in S_r(x^{(0)})\ \forall\, k \geqslant 0$, folgt

$$\|Df(x^{(k)}) - Df(x^{(0)})\| \leqslant \gamma\|x^{(k)} - x^{(0)}\| < \gamma r.$$

$$\Longrightarrow \|Df(x^{(k)})\| \leqslant \gamma r + \|Df(x^{(0)})\| =: \kappa.$$

Aus der Iterationsgleichung folgt:

$$\|f(x^{(k)})\| \leqslant \kappa\|x^{(k+1)} - x^{(k)}\|.$$

$$\Longrightarrow \lim_{k\to\infty} \|f(x^{(k)})\| = 0$$

und wegen f stetig gilt $\lim_{k\to\infty} f(x^{(k)}) = f(\hat{x})$. □

Man kann unter noch stärkeren Voraussetzungen mit dem Satz von Newton–Kantorovich zeigen, dass $\hat{x}$ die einzige Nullstelle in $S_r(x^{(0)})$ ist.
Voraussetzung dafür, dass das Newton–Verfahren konvergiert, ist, dass der Startwert

genügend nahe bei der gesuchten Lösung liegt. Um globale Konvergenz zu erzielen, wird das Newton–Verfahren durch Einführung eines Parameters modifiziert. Setze

$$x^{(k+1)} = x^{(k)} - \lambda_k d^{(k)}, d^{(k)} := Df(x^{(k)})^{-1} f(x^{(k)}) \tag{3.27}$$

Die λ_k werden dabei so gewählt, dass die Folge $\{h(x^{(k)})\}$ mit $h(x) = f(x)^T f(x)$ streng monoton fällt und die $x^{(k)}$ gegen ein Minimum von $h(x)$ konvergieren.
$\longrightarrow$ Übung.
Was hat das mit Nullstelle zu tun? Da

$$\begin{aligned} &h(x) \geqslant 0 && \forall x, \\ &h(\hat{x}) = 0 \iff && f(\hat{x}) = 0 \quad (f(x)^T f(x) = \|f(x)\|_2^2), \end{aligned}$$

so ist jedes lokale Minimum $\hat{x}$ von h mit $h(\hat{x}) = 0$ auch globales Minimum von h und Nullstelle von f.

Definiere Menge

$$D(\gamma, x) = \{s \in \mathbb{R}^n | \ \|s\| = 1 \text{ und } Dh(x)^T s \geqslant \gamma \|Dh(x)\|\}, \gamma > 0. \tag{3.28}$$

Lemma 3.29 *Sei $h : \mathbb{R}^n \to \mathbb{R}$ eine Funktion, deren Gradient $Dh(x)$ für alle $x \in V(\hat{x})$ ($V(\hat{x})$ Umgebung von $\hat{x}$) definiert und stetig ist. Sei ferner $Dh(\hat{x}) \neq 0$ und $\gamma > 0$. Dann gibt es eine Umgebung $U(\hat{x}) \subseteq V(\hat{x})$ von $\hat{x}$ und ein $\lambda > 0$, so dass*

$$h(x - \mu s) \leqslant h(x) - \frac{\mu\gamma}{4} \|Dh(\hat{x})\|,$$

für alle $x \in U(\hat{x}), s \in D(\gamma, x)$ und $0 \leqslant \mu \leqslant \lambda$.
! *(zeigt wann man $h(y) < h(x)$ finden kann.)*

Beweis: Sei

$$U^1(\hat{x}) = \{x \in V(\hat{x}) | \ \|Dh(x) - Dh(\hat{x})\| < \frac{\gamma}{4} \|Dh(\hat{x})\|\}.$$

Da $Dh(\hat{x}) \neq 0$ und $Dh(x)$ stetig auf $V(\hat{x})$, so folgt $U^1(\hat{x}) \neq \emptyset$ und U^1 ist Umgebung von $\hat{x}$. Aus demselben Grund folgt auch, dass

$$U^2(\hat{x}) = \{x \in V(\hat{x}) | \ D(\gamma, x) \subseteq D(\frac{\gamma}{2}, \hat{x})\}$$

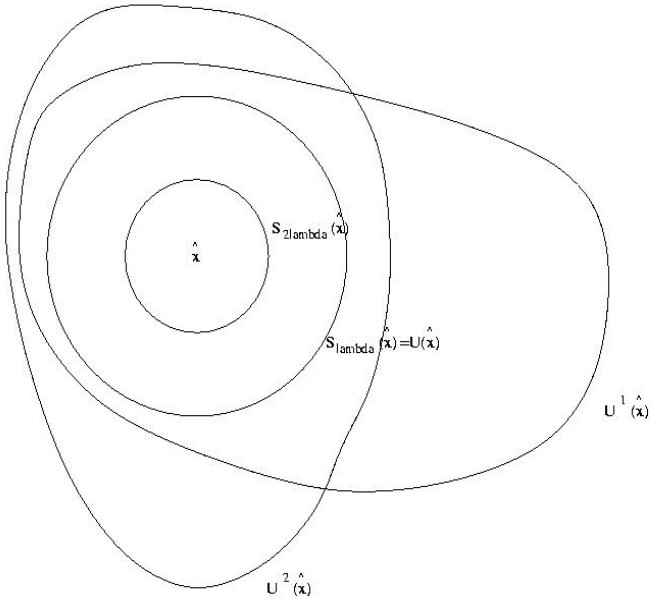
eine Umgebung von $\hat{x}$ ist. Wähle nun $\lambda > 0$, so dass

$$\overline{S_{2\lambda}(\hat{x})} := \{x | \ \|x - \hat{x}\| \leqslant 2\lambda\} \subseteq U^1(\hat{x}) \cap U^2(\hat{x})$$

und setze

$$U(\hat{x}) = \overline{S_\lambda(\hat{x})} = \{x | \ \|x - \hat{x}\| \leqslant \lambda\}.$$

Graphik 3.30



Dann gilt für alle $x \in U(\hat{x})$ mit $0 \leqslant \mu \leqslant \lambda, s \in D(\gamma, x)$, dass

$$\begin{aligned} h(x) - h(x - \mu s) &= \mu Dh(x - \theta\mu s)^T s \\ &= \mu \left[(Dh(x - \theta\mu s) - Dh(\hat{x}))^T s + Dh(\hat{x})^T s \right], \end{aligned}$$

für ein θ mit $0 < \theta < 1$. Mit $x \in U(\hat{x})$ gilt auch

$$x, x - \mu s, x - \theta\mu s \in U^1 \cap U^2,$$

also folgt

$$\begin{aligned} h(x) - h(x - \mu s) &\geqslant -\frac{\mu\gamma}{4}\|Dh(\hat{x})\| + \mu Dh(\hat{x})^T s \\ &\geqslant \frac{-\mu\gamma}{4}\|Dh(\hat{x})\| + \mu\frac{\gamma}{2}\|Dh(\hat{x})\| \\ &= \frac{\mu\gamma}{4}\|Dh(\hat{x})\|. \end{aligned}$$

□

Algorithmus 3.31

Input: *Differenzierbare Funktion* $h(x) : \mathbb{R}^n \to \mathbb{R}$, *und Zahlen* σ_k, γ_k, $k = 0, 1, 2, \ldots$ *mit*

$$\inf_k(\gamma_k) > 0, \inf_k \sigma_k > 0,$$

sowie einen Startwert $x^{(0)} \in \mathbb{R}^n$.

Output: *Folge* $x^{(k)}$, $k = 0, 1, 2, \ldots$, *die gegen Minimum von* $h(x)$ *konvergiert.*

FOR $k = 0, 1, 2, \ldots$ *UNTIL SATISFIED*
Sei $s^{(k)} \in D(\gamma_k, x^{(k)})$,

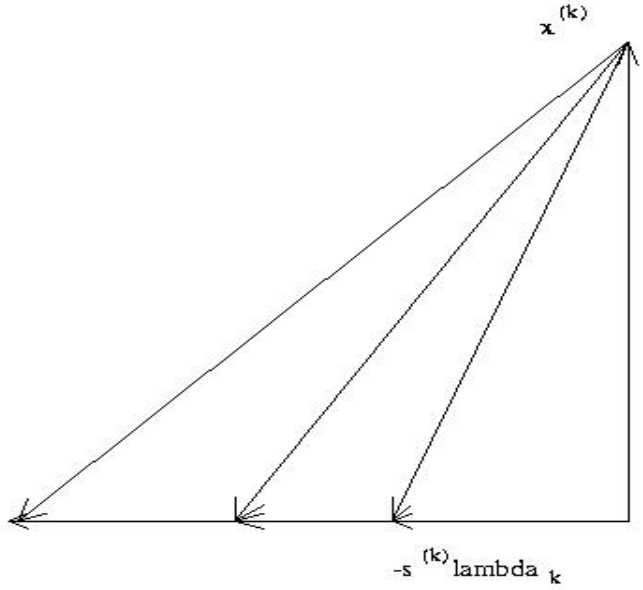
$$x^{(k+1)} = x^{(k)} - \lambda_k s^{(k)}, \tag{3.32}$$

wobei $\lambda_k \in [0, \sigma_k \|Dh(x^{(k)})\|]$, *so dass*

$$h(x^{(k+1)}) = \min_\mu \{h(x^{(k)} - \mu s^{(k)})|\ 0 \leqslant \mu \leqslant \sigma_k \|Dh(x^{(k)})\|\}.$$

END

Graphik 3.33 *I*



D.h. gesucht wird $\min \varphi(\mu) = \min h(x^{(k)} + \mu s^{(k)})$, $\mu \in [0, \sigma_k \|Dh(x^{(k)})\|]$. Suche Minimum über λ_k. Das ist jetzt eine eindimensionale Minimierung, und damit zwar einfacher als $\min h(x)$ aber immer noch sehr aufwendig. Dies wird später noch vereinfacht.
Konvergenz von (3.32).

Satz 3.34 *Seien* $h : \mathbb{R}^n \to \mathbb{R}$ *und* $x^{(0)} \in \mathbb{R}^n$, *so dass*

a) $K := \{x|\ h(x) \leqslant h(x^{(0)})\}$ *kompakt.*

b) h *stetig differenzierbar in einer Umgebung von* K.

Dann gilt für jede Folge $\{x^{(k)}\}$ *in (3.32)*

1) $x^{(k)} \in K$ *für alle* $k = 0, 1, 2, \ldots$.

2) $\left\{x^{(k)}\right\}$ *besitzt mindestens einen Häufungspunkt* $\hat{x}$ *in* K*.*

3) Jeder Häufungspunkt $\hat{x}$ *von* $\left\{x^{(k)}\right\}$ *erfüllt* $Dh(\hat{x}) = 0$*. (Stationärer Punkt)*

Beweis: Aus der Definition von $x^{(k)}$ folgt $h(x^{(0)}) \geqslant h(x^{(1)}) \geqslant \cdots \Longrightarrow x^{(k)} \in K$ für alle k und wegen K kompakt gibt es eine Häufungspunkt $\hat{x}$ in K. Also folgt 1), 2).
Nun zu 3). Mit Widerspruch. Angenommen $Dh(\hat{x}) \neq 0$, und sei o.B.d.A. $\lim\limits_{k\to\infty} x^{(k)} = \hat{x}$ (sonst nehme Teilfolge). Setze $\gamma = \inf\limits_k \gamma_k > 0, \sigma = \inf\limits_k \sigma_k > 0$. Nach Lemma 3.29 gibt es $U(\hat{x})$ und $\lambda > 0$, so dass

$$h(x - \mu s) \leqslant h(x) - \mu\frac{\gamma}{4}\|Dh(\hat{x})\|,\ \forall x \in U(\hat{x}),\ s \in D(\gamma, x),\ 0 \leqslant \mu \leqslant \lambda. \tag{3.35}$$

Da $\lim\limits_k x^{(k)} = \hat{x}$ und $Dh(x)$ stetig ist, gibt es k_0, so dass für alle $k \geqslant k_0$ gilt

$$x^{(k)} \in U(\hat{x}),\ \|Dh(x^{(k)})\| \geqslant \frac{1}{2}\|Dh(\hat{x})\|.$$

Sei nun

$$\begin{aligned} \Lambda &= \min\{\lambda, \frac{1}{2}\sigma\|Dh(\hat{x})\|\}, \\ \varepsilon &= \Lambda\frac{\gamma}{4}\|Dh(\hat{x})\| > 0 \end{aligned}$$

Wegen $\sigma_k \geqslant \sigma$ folgt

$$[0, \Lambda] \subseteq [0, \sigma_k\|Dh(\hat{x})\|],\ \forall\, k \geqslant k_0,$$

also $h(x^{(k+1)}) \leqslant \min\limits_{\mu}\{h(x^{(k)} - \mu s^{(k)})|\ 0 \leqslant \mu \leqslant \Lambda\}$.

$$\overset{(3.35)}{\Longrightarrow}\ h(x^{(k+1)}) \leqslant h(x^{(k)}) - \frac{\Lambda\gamma}{4}\|Dh(\hat{x})\| = h(x^{(k)}) - \varepsilon,\ \forall\, k \leqslant k_0,$$

da $\Lambda \leqslant \lambda,\ x^{(k)} \in U(\hat{x}),\ s^{(k)} \in D(\gamma_k, x^{(k)}) \subseteq D(\gamma, x^{(k)})$.

$$\Longrightarrow \lim_{k\to\infty} h(x^{(k)}) = -\infty,\ \text{Widerspruch!}$$

Denn $h(x^{(k)}) \geqslant h(x^{(k+1)}) \geqslant \cdots \geqslant h(\hat{x})$. □

Dies ist ein allgmeines Minimierungsverfahren, was sehr viele Anwendungen hat, benötigt wird noch die eindimensionale Minimierung. Diese wird in Algorithmus 3.31 durch einen endlichen Suchprozess ersetzt.

a) Wähle $s^{(k)} \in D(\gamma_k, x^{(k)})$ und definiere

$$\begin{aligned} \rho_k &:= \sigma_k\|Dh(x^{(k)})\| \\ h_k(\mu) &= h(x^{(k)} - \mu s^{(k)}) \end{aligned}$$

und bestimme die kleinste Zahl $j \geqslant 0$ mit

$$h_k(\rho_k 2^{-j}) \leqslant h_k(0) - \rho_k 2^{-j}\frac{\gamma_k}{4}\|Dh(x^{(k)})\|.$$

b) Bestimme λ_k und damit

$$\begin{aligned} x^{(k+1)} &:= x^{(k)} - \lambda_k s^{(k)}, \quad \text{so dass gilt} \\ h(x^{(k+1)}) &= \min_{0 \leqslant i \leqslant j} h_k(\rho_k 2^{-i}). \end{aligned}$$

Das gesuchte j existiert. Ist $x^{(k)}$ stationär, so ist $j = 0$, andernfalls wende Lemma 3.29 auf $\hat{x} = x^{(k)}$ an. In jedem Fall kann j und damit auch λ_k in einem endlichen Suchprozess bestimmt werden und es gilt Satz 3.34 auch für die so bestimmte Folge.

$\longrightarrow$ Übung.

3.2.1 Anwendung auf modifiziertes Newtonverfahren

Wir wenden nun zur Lösung von $f(x) = 0$ das Minimierungsverfahren 3.31 auf $h(x) = f(x)^T f(x)$ an.
Als Suchrichtung $s^{(k)}$ in Punkt $x^{(k)}$ wählen wir den nomierten Newtonvektor

$$s^{(k)} = \frac{d^{(k)}}{\|d^{(k)}\|} \text{ mit } d^{(k)} = Df(x^{(k)})^{-1} f(x^{(k)}).$$

(Immer definiert falls $Df(x^{(k)})^{-1}$ existiert.)

Lemma 3.36 *Sei x so, daß $d(x) = Df(x)^{-1} f(x)$ existiert und nicht verschwindet. Dann gilt*

$$s(x) = \frac{d(x)}{\|d(x)\|} \in D(\gamma, x), \text{ für alle } 0 < \gamma < \bar{\gamma}(x),$$

wobei

$$\bar{\gamma}(x) = \frac{1}{\mathrm{cond}_2(Df(x))} = \frac{1}{\|Df(x)\|_2 \|Df(x)^{-1}\|_2}.$$

Beweis: $Dh(x) = 2f^T(x)Df(x)$. Submultiplikativität von $\| \bullet \|_2$ liefert

$$\begin{aligned} \|f^T(x)Df(x)\|_2 &\leqslant \|Df(x)\|_2 \, \|f(x)\|_2 \\ \|Df(x)^{-1} f(x)\|_2 &\leqslant \|Df(x)^{-1}\|_2 \, \|f(x)\|_2 \end{aligned}$$

und daher

$$\begin{aligned} \frac{Dh(x)s}{\|Dh(x)\|} &= \frac{f(x)^T Df(x) Df(x)^{-1} f(x)}{\|Df(x)^{-1} f(x)\| \, \|f^T(x)Df(x)\|} \\ &\geqslant \frac{1}{\mathrm{cond}_2(Df(x))} \\ &> 0. \end{aligned}$$

Für alle γ mit $0 < \gamma < \frac{1}{\mathrm{cond}_2(Df(x))}$ gilt also $s \in D(\gamma, x)$. □

Falls $Df(x)^{-1}$ existiert folgt also, daß $Dh(x) = 0 \iff f(x) = 0$. Damit erhalten wir das modifizierte Newtonverfahren.

Algorithmus 3.37

Input: *Differenzierbare Funktion* $f(x) : \mathbb{R}^n \to \mathbb{R}^n$ *sowie Startwert* $x^{(0)} \in \mathbb{R}^n$.

Output: *Folge* $x^{(k)}$, $k = 0, 1, 2, \ldots$ *die gegen Nullstelle von* f *konvergiert.*

FOR $k = 0, 1, 2, \ldots$ *UNTIL SATISFIED*

Berechne $d^{(k)} = Df(x^{(k)})^{-1} f(x^{(k)})$, *(lineares Gleichungssystem mit* QR*)*

$\gamma_k = \frac{1}{\mathrm{cond}_2(Df(x^{(k)}))}$, *fast frei aus* QR

Setze $h_k(\tau) = h(x^{(k)} - \tau d^{(k)}) = f(x^{(k)} - \tau d^{(k)})^T f(x^{(k)} - \tau d^{(k)})$.

Bestimme $j \geqslant 0$, *so daß* $h_k(2^{-j}) \leqslant h_k(0) - 2^{-j} \frac{\gamma_k}{4} \|d^{(k)}\| \, \|Dh(x^{(k)})\|$.

Bestimme λ_k *und* $x^{(k+1)} = x^{(k)} - \lambda_k d^{(k)}$, *so daß gilt*

$$h(x^{(k+1)}) = \min_{0 \leqslant i \leqslant j} h_k(2^{-i}).$$

END

Kosten: Pro Schritt

a) Berechnung von $Df(x^{(k)})$

b) Lösung von $Df(x^{(k)})d^{(k)} = f(x^{(k)})$

c) eindimensionale Minimierung.

Fehleranalyse: b) mit QR klar. a),c) hängt von f ab.

Über dieses Verfahren hat man die folgende Konvergenzaussage:

Satz 3.38 *Gegeben sei* $f : \mathbb{R}^n \to \mathbb{R}^n, x^{(0)} \in \mathbb{R}^m$ *mit*

a) $K = \{x |\ h(x) \leqslant h(x_0)\}$, $h(x) = f^T(x) f(x)$ *ist kompakt.*

b) f *ist auf Umgebung von* K *stetig differenzierbar.*

c) $Df(x)^{-1}$ *existiert für alle* $x \in K$.

Dann ist die Folge $\{x^{(k)}\}$, *die in Algorithmus 3.37 erzeugt wird, wohldefiniert und es gilt*

a) $x^{(k)} \in K$, $\forall\, k = 0, 1, 2, \ldots, \{x^{(k)}\}$ *besitzt mindestens Häufungspunkt* $\hat{x} \in K$.

b) Jeder Häufungspunkt $\hat{x}$ *von* $\{x^{(k)}\}$ *ist Nullstelle von* f.

Beweis: hier nicht, siehe z.B. Stoer. □

$\longrightarrow$ Übung.
Die Voraussetzungen geben die Klassen von Funktionen an, auf die Alg. 3.37 anwendbar ist.

3.2.2 Praktische Realisierung des modifizierten Newton–Verfahrens

Problem bei der Berechnung ist, daß in jedem Schritt $Df(x^{(k)})$ ausgerechnet werden muß. Idee: approximiere $Df(x^{(k)})$ durch $\triangle f(x^{(k)}) = (\triangle_1 f, \cdots, \triangle_n f)(x^{(k)})$ (Differenzenquotientenmatrix), wobei

$$\triangle_i f(x) = \frac{f(x_1, \ldots, x_{i-1}, x_i + h_i, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_n)}{h_i} = \frac{f(x + h_i e_i) - f(x)}{h_i}.$$

Schwierigkeiten: h_i zu groß $\longrightarrow$ schlechte Approximation an $Df(x)$, h_i zu klein, dann $f(x + he_i) \approx f(x)$.
Wähle h_i so, daß $f(x)$ und $f(x+he_i)$ ungefähr die $\frac{t}{2}$ ersten Stellen gemeinsam haben (t–stellige Rechnung)

$$|h_i| \approx \sqrt{eps}\|f(x)\|/\| \triangle_i f(x)\|$$

! Auslöschung is dann noch erträglich.
Im allgemeinen ist jedoch auch die Berechnung von $\triangle f$ zu teuer daher verwende folgende Vereinfachung mit Satz von Broyden:

Satz 3.39 *Seien $A, B \in \mathbb{R}^{n,n}, b \in \mathbb{R}^n, F : \mathbb{R}^n \to \mathbb{R}^n$ gegeben durch $F(u) = Au + b$, $x, x' \in \mathbb{R}^n$ und $p = x - x'$, $q = F(x') - F(x) = Ap$. Dann gilt für die Rang–1–Modifikation $B' = B + \frac{1}{p^T p}(q - Bp)p^T$ die Abschätzung*

$$\|B' - A\|_2 \leqslant \|B - A\|_2$$

und $B'p = Ap = q$.

Beweis: $(B' - A)p = Bp + (q - Bp) - Ap = 0$.
Für jedes $u \in \mathbb{R}^n$ mit $\|u\|_2 = 1$ existiert ein v mit

$$u = \alpha p + v,\ v^T p = 0,\ \|v\|_2 \leqslant 1.$$

Also folgt mit $\|u\|_2 = 1$

$$\|(B' - A)u\|_2 = \|(B' - A)v\|_2 = \|(B - A)v\|_2 \leqslant \|B - A\|_2\ \|v\|_2 \leqslant \|B - A\|_2.$$

Also auch für $\|B' - A\|_2$, welches das Supremum über alle u ist. □

Also folgt, daß $DF(x) = A$ von B' genau so gut approximiert wird wie von B. B' und $DF(x)$ transformieren p in denselben Vektor. Die Idee ist nun, $f(x)$ durch eine affine Abbildung zu approximieren (in der Nähe einer Nullstelle) und diese Idee anzuwenden.
Ersetze in Algorithmus 3.37 den Teil in der Schleife.

Algorithmus 3.40 (Broyden–Rang 1–Verfahren)
Input: $f(x) : \mathbb{R}^n \to \mathbb{R}^n$ *differenzierbar und* $x^{(0)} \in \mathbb{R}^n$
Output: *Folge* $x^{(k)}, k = 0, 1, 2, \ldots$

FOR $k = 0, 1, 2, \ldots$ *UNTIL SATISFIED*
$d^{(k)} = B_k^{-1} f(x^{(k)})$
$x^{(k+1)} = x^{(k)} - \lambda_k d^{(k)}$
$p^{(k)} = x^{(k+1)} - x^{(k)}$
$q^{(k)} = f^{(x^{(k+1)})} - f(x^{(k)})$
$B^{(k+1)} = B_k + \frac{1}{p^{(k)^T} p^{(k)}}(q^{(k)} - B_k p^{(k)})p^{(k)^T}$
END

λ_k wird durch näherungsweise Minimierung von $\|f(x^{(k+1)})\|^2 \approx \min_{\lambda \geqslant 0} \|f(x^{(k)} - \lambda d^{(k)})\|^2$ bestimmt wie vorher.
$B_0 \approx \triangle f(x^{(0)})$. Praktische Erfahrung zeigt, daß man diese Wahl von B_{k+1} nur für $0.5 \leqslant \lambda_k \leqslant 1$ machen sollte, sonst besser $B_{k+1} = \triangle f(x_{k+1})$.
Die Lösung des linearen Gleichungssystems

$$B_k d^{(k)} = f(x^{(k)})$$

kann über Rang 1 Aufdatierungsformel sehr leicht gemacht werden.
$\longrightarrow$ Übung 3.58: Sherman–Morrison Formel
$\longrightarrow$ Zusammenhang modifiziertes Newton – Sekantenverfahren – andere Verfahren.

3.3 Nullstellenbestimmung für Polynome

Vieles vereinfacht sich, falls $f(x)$ skalares Polynom ist. Dies ist praktisch fast nicht mehr relevant außer in speziellen Fällen.
Gegeben: Polynom

$$p(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n.$$

Gesucht: reelle (alle) Nullstellen.

Newtonverfahren:

$$x^{(k+1)} = x^{(k)} - \frac{p(x^{(k)})}{p'(x^{(k)})}. \tag{3.41}$$

Probleme:

1. Preiswerte Auswertung von p, p'.
2. Wahl der Startwerte.
3. Deflation von Nullstellen.
4. Komplexe Nullstellen.

Zu Punkt 1: Horner Schema
Auswertung von $p(z)$ durch Algorithmus

$$\begin{aligned} b_0 &= a_0 \\ b_k &= b_{k-1} z + a_k, \ k = 1, 2, \ldots, n \end{aligned} \tag{3.42}$$

$$p(z) = ((\cdots((a_0 z + a_1) z + a_2) \cdots + a_{n-2}) z + a_{n-1}) z + a_n$$

kostet nur $2n$ flops und es gilt der folgende Satz:

Satz 3.43 *Sei $p(x) = a_0 x^n + \cdots + a_n$ und $p_1(x) = b_0 x^{n-1} + \cdots + b_{n-1}$, b_i wie in (3.42) für $p(z)$. Dann gilt:*

$$p(x) = (x - z) p_1(x) + p(z), \tag{3.44}$$

$$p'(z) = p_1(z). \tag{3.45}$$

Beweis:

$$\begin{aligned} b_n &= p(z) \\ \Longrightarrow (x-z)p_1(x)+p(z) &= xp_1(x)-zp_1(x)+p(z) \\ &= b_0x^n+b_1x^{n-1}+\cdots+b_{n-1}x+b_n-(zb_0x^{n-1}+\cdots+zb_{n-1}) \\ &= a_0x^n+\cdots+a_n \\ &= p(x). \end{aligned}$$

$$\Longrightarrow p'(x)=p_1(x)+(x-z)p'(z)$$

Einsetzen von z gibt (3.45). □

Durch 2–faches Anwenden des Hornerschemas kann man also $p(z), p'(z)$ mit $4n-2$ flops bestimmen.

Zu Punkt 2: Wahl der Startwerte

Satz 3.46 *Für alle Nullstellen ξ_i eines Polynoms $p(x)=a_0x^n\cdots+a_n$ gilt*

$$\begin{aligned} a)\quad |\xi_i| &\leqslant \max\left\{|\frac{a_n}{a_0}|, 1+|\frac{a_{n-1}}{a_0}|, \ldots, 1+|\frac{a_1}{a_0}|\right\} \\ b)\quad |\xi_i| &\leqslant \max\left\{1, \sum_{i=1}^{n}|\frac{a_i}{a_0}|\right\} \\ c)\quad |\xi_i| &\leqslant \max\left\{|\frac{a_n}{a_{n-1}}|, 2|\frac{a_{n-1}}{a_{n-2}}|, \ldots, 2|\frac{a_1}{a_0}|\right\} \\ d)\quad |\xi_i| &\leqslant \sum_{i=0}^{n-1}|\frac{a_{i+1}}{a_i}| \\ e)\quad |\xi_i| &\leqslant 2\max\left\{|\frac{a_1}{a_0}|, \sqrt{|\frac{a_2}{a_0}|}, \sqrt[3]{|\frac{a_3}{a_0}|}, \ldots, \sqrt[n]{|\frac{a_n}{a_0}|}\right\} \end{aligned}$$

Weitere Sätze in *Geometry of Polynomials* von M. Marden. Damit können Kreise bestimmt werden, in denen die Nullstellen liegen.
⟶ Übung 6.50,3.51: Gerschgorin-Satz angewendet auf Begleitmatrix.

Zu Punkt 3: Deflation
Falls man Nullstelle $\hat{x}$ bestimmt hat, könnte man Polynom p_1 bestimmen, $p_1(x-\hat{x})=p(x)$ bilden und weiterrechnen.
Im allgemeinen numerisch sehr schlecht, besser mit rationaler Funktion $p_1(x)=\frac{p(x)}{x-\hat{x}}$ weiterrechnen.
⟶ Übung 3.56

Zu Punkt 4: Komplexe Nullstellen, Bairstow–Verfahren
Für reelle Polynome will man komplexe Nullstellen mit reeller Rechnung bestimmen. Mit $z_j \in \mathbb{C}\setminus\mathbb{R}$ ist dann auch $\bar{z}_j$ Nullstelle und $(x-z_j)(x-\bar{z}_j)=x^2-rx+s$ reeller Faktor von $p(x)$.

$$Re(z_j)=-\frac{r}{2},\ Im(z_j)=\sqrt{s-\frac{r^2}{4}}$$

Wende Polynomdivision mit Newtonverfahren auf $p(x)/(x^2+ux+v)$ an.

$$\frac{p(x)}{x^2+ux+v}=Q(x)+\frac{Ax+B}{x^2+ux+v} \tag{3.47}$$

Die so definerten A, B sind eindeutig. A, B können mit dem Hornerschema berechnet werden.
$\longrightarrow$ Übung 3.57

Wende nun 2–dimensionales Newtonverfahren an, um r, s so zu bestimmen, daß

$$\left[\begin{array}{c} A(r,s) \\ B(r,s) \end{array} \right] = 0 \tag{3.48}$$

Dies ist das Bairstow–Verfahren.
$\longrightarrow$ Übung 3.57
$\longrightarrow$ Sekantenverfahren für Polynome.
Weitere Verfahren wie Bisektion und Sturmsche Ketten später bei Eigenwertproblemen.

3.4 Übungen zu Kapitel 3

Übung 3.49 *Sei $\Phi : K \to R^n$, $K \subseteq \mathbb{R}^n$ offen, konvex, Φ differenzierbar und nehme an, daß das totale Differential $D\Phi$ auf K beschränkt ist. Zeige:*

$$\|\Phi(x) - \Phi(y)\|_\infty \leqslant \|D\Phi\|_\infty|_K \|x - y\|_\infty, \ \forall x, y \in K.$$

Übung 3.50 *Für welche der folgenden Gleichungen und welche Startwerte konvergiert die zugehörige Fixpunktiteration?*

1. $x = e^x - \sin x + x$
2. $x = \sin x - e^x + x$
3. $x = \arcsin e^x, x < 0$
4. $x = \ln \sin x, x \in (0, \pi)$

Übung 3.51 *Sei $p(x) = a_n x^n + \cdots + a_1 x + a_0$, $a_n \neq 0$, Polynom n–ten Grades. Zeige:*

1. *Die Nullstellen von p sind die Eigenwerte der Begleitmatrix*

$$A = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ -\frac{a_0}{a_n} & -\frac{a_1}{a_n} & \cdots & -\frac{a_{n-1}}{a_n} \end{pmatrix}$$

2. *Die Nullstellen $\xi_1, \ldots, \xi_n$ von p erfüllen für alle $i = 1, \ldots, n$ die Ungleichungen*

$$|\xi_i| \leqslant \max\{1, \sum_{k=0}^{n-1} |\frac{a_i}{a_n}|\}, \ |\xi_i| \leqslant \max\{|\frac{a_0}{a_n}|, 1 + |\frac{a_1}{a_n}|, \ldots, 1 + |\frac{a_{n-1}}{a_n}|\},$$

(Hinweis: Für jeden Eigenwert λ von A und jede mit der Matrixmultiplikation verträgliche Norm gilt: $|\lambda| \leqslant \|A\|$)

Übung 3.52 *Sei $A \in \mathbb{R}^{n,n}$ und*

$$f(x_1, \ldots, x_n, x_{n+1}) = \begin{pmatrix} (A - x_{n+1} I) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\ \frac{1}{2}(\sum_{i=1}^n |x_i|^2 - 1) \end{pmatrix}.$$

Zeige:

1. *$(x_1, \ldots, x_{n+1})$ ist Nullstelle von f, genau dann wenn x_{n+1} Eigenwert und $(x_1, \ldots, x_n)^T$ normierter Eigenvektor von A sind.*
2. *Sei $(x_1, \ldots, x_{n+1})$ Nullstelle von f. Dann ist die Ableitungsmatrix $Df(x_1, \ldots, x_{n+1})$ nicht singulär, genau dann wenn x_{n+1} ein einfacher Eigenwert von A ist (Tip: Jordan–Form). Was bedeutet das für das Newton–Verfahren?*

Übung 3.53 *Schreibe ein* **MATLAB** *–Programm für das Newtonverfahren zur Bestimmung einer Nullstelle einer Funktion. Wende das Verfahren auf die Funktion $e^x - \sin x$ an und vergleiche das Programm bezüglich Rechenaufwand und Genauigkeit mit der Fixpunktiteration angewandt auf die Funktionen aus Aufgabe 3.50*

Übung 3.54 *Sei $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$. Bestimme den Rechenaufwand für die Berechnung von $p(x)$, falls man gerade durchrechnet, d.h. ohne Horner–Schema.*

Übung 3.55 *Sei p Polynom n–ten Grades. Zeige:*

1. *Zu gegebenem $x \in \mathbb{R}$ lässt sich $p(x)$ mit dem Hornerschema in $2n$ flops berechnen.*
2. *Zu gegebenem $x \in \mathbb{R}$ lassen sich $p(x), p'(x)$ mit einem angepassten Hornerschema in $4n-2$ flops berechnen, d.h. es werden gegenüber der Berechnung von $p(x)$ zusätzlich $2n-2$ flops für die Berechnung von $p'(x)$ benötigt (Tip: Satz 3.43 der Vorlesung).*

Übung 3.56 *Schreibe ein* **MATLAB** *–Programm für das Newtonverfahren zur Bestimmung der reellen Nullstellen einer rationalen Funktion $f(x) = p(x)/q(x)$, wobei p, q Polynome seien. Vergleiche das Programm bezüglich Rechenaufwand und Genauigkeit mit der Funktion 'roots' aus* **MATLAB** *anhand des Beispiels $p_n(x) = \prod_{i=1}^{n}(x-i),\ n = 2, 4, 6, \ldots, 20,\ q(x) = 1$.*

Übung 3.57 *Sei $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$ Polynom n–ten Grades mit $n \geqslant 3$. Zeige:*

1. *Bestimme in Abhängigkeit von gegebenen $r, q \in \mathbb{R}$ nacheinander die Koeffizienten $b_0, \ldots, b_n$, so dass*
$$p(x) = q(x)(x^2 - rx - q) + b_{n-1}(x - r) + b_n.$$
Dabei ist $q(x) = b_0x^{n-2} + b_1x^{n-3} + \cdots + b_{n-2}$. Was stellst Du fest?
2. *Setze $c_{j-1} = \frac{\partial b_j}{\partial r}$, $d_{j-2} = \frac{\partial b_j}{\partial q}$, $j = 0, \ldots, n$. Dann gilt für die partiellen Ableitungen der Koeffizienten $b_0, \ldots, b_n$:*
$$c_{-1} = 0,\ c_0 = b_0,\ c_{j-1} = b_{j-1} + rc_{j-2} + qc_{j-3},\ j = 2, \ldots, n$$
sowie
$$d_j = c_j,\ j = -1, \ldots, n.$$
Was kostet die Berechnung der partiellen Ableitungen von b_{n-1}, b_n?
3. *Das 2–dimensionale Newton–Verfahren zur Bestimmung der Nullstellen von $\Phi(r, q) = (b_{n-1}, b_n)$ lautet*
$$\begin{pmatrix} r_{k+1} \\ q_{k+1} \end{pmatrix} = \begin{pmatrix} r_k \\ q_k \end{pmatrix} - \begin{pmatrix} c_{n-2} & c_{n-3} \\ c_{n-1} & c_{n-2} \end{pmatrix}^{-1} \begin{pmatrix} b_{n-1} \\ b_n \end{pmatrix}.$$

Übung 3.58 *1. Sei $A \in \mathbb{R}^{n,n}$, $A = S - fg^T$ mit $f, g \in \mathbb{R}^n$ und $S \in \mathbb{R}^{n,n}$ nicht singulär. Setze $S_c = 1 - g^TS^{-1}f$. Dann ist A nicht singulär genau dann, wenn $S_c \neq 0$ ist und es gilt (Sherman–Morrison–Formel):*
$$A^{-1} = S^{-1} + S^{-1}f\frac{1}{S_c}g^TS^{-1}$$
Tip: multipliziere die rechte Seite an A heran.

2. *Was kostet das Lösen von m linearen Gleichungssystemen mit A verglichen mit dem Lösen m linearer Gleichungssysteme mit S?*

3. *Lässt sich die Formel auf Blöcke verallgemeinern? D.h. $F, G \in \mathbb{R}^{n,r}$.*

Übung 3.59 *Ist $A(x)$ stetige nichtsinguläre Matrixwertige Funktion, dann gilt das auch für ihre Inverse. Tip: Cramersche Regel.*

Übung 3.60 *Schreibe ein* **MATLAB** *–Algorithmus zur Lösung einer nichtlinearen Gleichung $\Phi(x) = 0$ mit Hilfe des Newton–Verfahrens bei gegebenen Startwert $x^{(0)}$ und Ableitungsmatrix $D\Phi(x)$. Teste das Verfahren für die Funktion $\Phi(x) = \phi(x) - x$ mit $\phi(x)$ aus Aufgabe 3.60 sowie an folgender Funktion Ψ für $n = 10$ mit Startwert $(1, \ldots, 1, 0)^\top$.*

$$\Psi(x_1, \ldots, x_n, x_{n+1}) = \begin{pmatrix} (A - x_{n+1} I) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\ \frac{1}{2}(1 - \sum_{i=1}^n |x_i|^2) \end{pmatrix}, \; A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}.$$

Was stellst du fest?

Übung 3.61 *Schreibe ein* **MATLAB** *–Algorithmus zur Lösung einer nichtlinearen Gleichung $\Phi(x) = 0$ mit Hilfe des Newton–Verfahrens bei gegebenen Startwert $x^{(0)}$ und Ableitungsmatrix $D\Phi(x)$. Teste das Verfahren für die Funktion $\Phi(x) = \phi(x) - x$ mit $\phi(x)$ aus Aufgabe 3.60 sowie an folgender Funktion Ψ für $n = 10$ mit Startwert $(1, \ldots, 1, 0)^\top$.*

$$\Psi(x_1, \ldots, x_n, x_{n+1}) = \begin{pmatrix} (A - x_{n+1} I) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\ \frac{1}{2}(1 - \sum_{i=1}^n |x_i|^2) \end{pmatrix}, \; A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}.$$

Was stellst du fest?

Übung 3.62 *Schreibe ein* **MATLAB** *–Algorithmus zur Lösung einer nichtlinearen Gleichung $\Phi(x) = 0$ mit Hilfe des MODIFIZIERTEN Newton–Verfahrens bei gegebenen Startwert $x^{(0)}$ und Ableitungsmatrix $D\Phi(x)$. Teste das Verfahren für die Funktion $\Phi(x) = \phi(x) - x$ mit $\phi(x)$ aus Aufgabe 3.60 sowie an folgender Funktion Ψ für $n = 10$ mit Startwert $(1, \ldots, 1, 0)^\top$.*

$$\Psi(x_1, \ldots, x_n, x_{n+1}) = \begin{pmatrix} (A - x_{n+1} I) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\ \frac{1}{2}(1 - \sum_{i=1}^n |x_i|^2) \end{pmatrix}, \; A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}.$$

Was stellst du fest?

Übung 3.63 *Es sei*

$$f(x, y) = \begin{pmatrix} \frac{3x}{1+x} + \frac{1}{y} \\ \frac{3y}{1+y} + \frac{2}{x} \end{pmatrix}.$$

1. *Man finde eine Teilmenge $I \subseteq \mathbb{R}^2$, in der f bzgl. der $\| \bullet \|_\infty$–Norm den Voraussetzungen des Banachschen Fixpunktsatzes genügt.*

2. *Man führe mit* $(x_0, y_0) = (2, 2)$ *zwei Iterationen durch und gebe eine a–posteriori Fehlerschranke für* (x_2, y_2) *an.*

3. *Wieviele Iterationsschritte sind hinreichend, um ausgehend von* $(x_0, y_0) = (2, 2)$ *den Fixpunkt mit einer Genauigkeit von* 10^{-4} *zu berechnen*

Kapitel 4

Interpolation

Ein Interpolationsproblem ist eine Aufgabe: Finde $a_0, \ldots, a_n$ so daß

$$P(x_i, a_0, \ldots, a_n) = f_i \quad i = 0, \ldots, n \tag{4.1}$$

für gegebene $(x_i, f_i) = (x_i, f(x_i)),\ i = 0, \ldots, n$ und eine vorgebene Funktion P. (x_i, f_i) heißen Stützpunkte/Knoten.
Wichtigste Beispiele:

Polynominterpolation

$$P = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Trigonometrische Interpolation

$$P = a_0 + a_1 e^{xi} + a_2 e^{2xi} + \cdots + a_n e^{nxi}$$

Rationale Interpolation

$$P = \frac{a_0 + a_1 x + \cdots + a_p x^p}{\beta_0 + \beta_1 x + \cdots + \beta_m x^m} \quad \text{hier nicht .}$$

Splines
stückweise Polynome.

Anwendungen:
Approximation von Funktionen, Quadratur, Lösung von gewöhnlichen oder Partiellen Differentialgleichungen, CAD, numerische Differentiation, Bildverarbeitung, Signalverarbeitung, ...

Beispiel aus alten Zeiten:
Logarithmentafel:

In Tafel:	$\log_{10} 1.0$
	$\log_{10} 1.1$
gesucht:	$\log 1.05$

Lösung lineare Interpolation: Berechne Gerade und werte diese bei 1.05 aus.

4.1 Polynominterpolation

Notation $\Pi_n = \{$ reelle oder komplexe Polynome vom Grad $\leqslant n\}$.

Satz 4.2 (Lagrange Interpolation) *Seien $n+1$ Stützpunkte $(x_i, f_i),\ i = 0, \ldots, n$ beliebig mit $x_i \neq x_k$ für $i \neq k$. Dann gibt es ein eindeutiges Polynom $P \in \Pi_n$, mit*

$$P(x_i) = f_i \quad i = 0, \ldots, n. \tag{4.3}$$

Beweis: Wir zeigen zuerst die Eindeutigkeit. Angenommen es gäbe $P_1, P_2 \in \Pi_n$

$$P_1(x_i) = P_2(x_i) = f_i \quad i = 0, \ldots, n.$$

Sei $P = P_1 - P_2 \in \Pi_n$. P hat grad $\leqslant n$ und mindestens $n+1$ verschiedene Nullstellen $x_i, i = 0, \ldots, n. \Longrightarrow P \equiv 0 \Longrightarrow P_1 = P_2$.

Die Existenz folgt per Konstruktion. Sei

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}. \tag{4.4}$$

Es gilt wegen a), daß L_i die eindeutigen Lösungen der Aufgabe

$$L_i(x_k) = \delta_{ik} = \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases}, k = 0, \ \ldots, n \tag{4.5}$$

sind. Dann folgt

$$P(x) = \sum_{i=0}^{n} f_i L_i(x) = \sum_{i=0}^{n} f_i \prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) \tag{4.6}$$

löst die Interpolationsaufgabe. □

$L_i(x)$ in 4.4 heißen Lagrange–Interpolations–Polynome. Sie bilden Basis für Π_n.
Andere Basis $x^0, x^1, \ldots, x^n$.
Für die praktische Berechnung sind die $L_i(x)$ nicht geeignet, genauso wie die Standardbasis. Für numerische Zwecke besser Neville-Aitken–Methode oder Newton'sche Interpolationsformel. Dazu führe zuerst dividierte Differenzen ein.

Definition 4.7 *Seien $(x_i, f_i),\ i = 0, \ldots, n$ gegeben. Der Ausdruck*

$$f[x_i, x_{i+1}, \ldots, x_{i+k}],$$

welcher durch die Rekursionen

$$f[x_i] = f_i \tag{4.8}$$

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] := \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}, \tag{4.9}$$

$i = 0, \ldots, n,\ k = 0, \ldots, n - i$ gegeben ist, heißt k–te dividierte Differenz von $(x_i, f_i), \ldots, (x_{i+k}, f_{i+k})$. Es gibt ein einfaches Schema zur Berechnung der dividierten Differenzen.

Tabelle 4.10 (Schema der dividierten Differenzen)

	$k=0$	$k=1$	$k=2$	$k=3$
x_0	$f_0 = f[x_0]$			
		$f[x_0,x_1] = \frac{f[x_1]-f[x_0]}{x_1-x_0}$		
x_1	$f_1 = f[x_1]$		$f[x_0,x_1,x_2]$	
		$f[x_1,x_2] = \frac{f[x_2]-f[x_1]}{x_2-x_1}$		$f[x_0,\ldots,x_3]$
x_2	$f_2 = f[x_2]$		$f[x_1,x_2,x_3]$	
		$f[x_2,x_3] = \frac{f[x_3]-f[x_2]}{x_3-x_2}$	$\vdots$	
x_3	$f_3 = f[x_3]$	$\vdots$		
x_4	$\vdots$			
$\vdots$				

Einfach rekursiv zu berechnen.
$\longrightarrow$ Übung 4.76
Bei der Newtoninterpolation verwendet man eine andere Basis: $N_0(x) = 1$

$$N_i(x) = \prod_{j=0}^{i-1}(x - x_j) = (x - x_0)(x - x_1)\cdots(x - x_{i-1}) \quad i = 1, \ldots, n \tag{4.11}$$

Das Interpolationspolynom ist dann

$$P(x) = \sum_{i=0}^{n} a_i N_i(x). \tag{4.12}$$

Die Koeffizienten a_i können aus den Formeln

$$\begin{array}{rcll} f_0 & = & P(x_0) & = & a_0 \\ f_1 & = & P(x_1) & = & a_0 + a_1(x_1 - x_0) \\ f_2 & = & P(x_2) & = & a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\ & \vdots & & & \end{array} \tag{4.13}$$

bestimmt werden.
Es gilt sofort, daß

$$P_{0\ldots k}(x) = \sum_{j=0}^{k} a_j N_j(x) \tag{4.14}$$

die Interpolationsaufgabe für $(x_i, f_i) \quad i = 0, \ldots, k$ erfüllt, für $k = 0, \ldots, n$.
Sei $P_{i,i+1,\ldots,i+k}$ das Polynom das die Interpolationsaufgabe für $(x_j, f_j),\ j = i, i+1, \ldots, i+k$ erfüllt.
Dann gilt

Satz 4.15

$$\begin{array}{rcl} P_{i,i+1,\ldots,i+k}(x) & = & f[x_i] + f[x_i, x_{i+1}](x - x_i) + \cdots \\ & & + f[x_i, x_{i+1}, \ldots, x_{i+k}](x - x_i)\cdots(x - x_{i+k-1}) \end{array}$$

Beweis: [Vollständige Induktion] Richtig für $k = 0$. Induktionsvor.: Richtig für $k - 1 \geqslant 0$. Nun gilt

$$P_{i,i+1,\ldots,i+k}(x) = P_{i,i+1,\ldots,i+k-1}(x) + a(x - x_i)\cdots(x - x_{i+k-1}) \tag{4.16}$$

per Konstruktion, wobei a gerade der Koeffizient von x^k in $P_{i,i+1,\ldots,i+k}(x)$ ist. Zu zeigen ist, daß

$$a = f[x_i, \ldots, x_{i+k}] \tag{4.17}$$

Nach IV sind die höchsten Koeffizienten von $P_{i,\ldots,i+k-1}$ und $P_{i+1,\ldots,i+k}$ gerade $f[x_i, \ldots, x_{i+k-1}]$ und $f[x_{i+1}, \ldots, x_{i+k}]$, also

$$\begin{aligned} P_{i,\ldots,i+k-1}(x) &= \cdots + f[x_i, \ldots, x_{i+k-1}]x^{k-1} \\ P_{i+1,\ldots,i+k}(x) &= \cdots + f[x_{i+1}, \ldots, x_{i+k}]x^{k-1}. \end{aligned}$$

Nun erfüllen die $P_{i,\ldots,i+k}$ gerade die Rekursionsformel (Neville–Formel)

$$P_{i,\ldots,i+k}(x) = \frac{(x - x_{i+k})P_{i,\ldots,i+k-1}(x) - (x - x_i)P_{i+1,\ldots,i+k}(x)}{x_{i+k} - x_i}, \tag{4.18}$$

wie man sofort durch Einsetzen und Satz 4.2 erhält. Also folgt, daß der Koeffizient von x^k

$$\frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} = f[x_i, \ldots, x_{i+k}] \tag{4.19}$$

nach (4.9) □

Also sind die Werte in der ersten Zeile des Dividierten Differenzen Schemas gerade die Koeffizienten des Newton–Polynoms.

Beispiel 4.20

$x_i =$	0	1	3
$f_i =$	1	3	2

Dividierte Differenzen Schema

$$\begin{array}{l|l|l|l} x_0 = 0 & f[x_0] = 1 & \searrow & \\ & & f[x_0, x_1] = 2 & \searrow \\ x_1 = 1 & f[x_1] = 3 & \nearrow \searrow & f[x_0, x_1, x_2] = -\frac{5}{6} \\ & & f[x_1, x_2] = -\frac{1}{2} & \nearrow \\ x_2 = 3 & f[x_2] = 2 & \nearrow & \end{array}$$

$$P_{0,1,2}(x) = 1 + 2(x - 0) - \frac{5}{6}(x - 0)(x - 1)$$

Auswertung von $P_{0,\ldots,n}(x)$ mit Hornerartigem Schema.
Vorteile der dividierten Differenzen und des Newton–Schemas.

- Weitere Stützstellen möglich.
- Man erhält Interpolationspolynom, welches mit Hornerartigem Schema leicht zu berechnen ist.
- $f[x_0, \ldots, x_k]$ symmetrisch in x_i, d.h. auf Anordnung kommt es nicht an. Siehe *Numerische Mathematik*, Stoer.

- Rundungsfehlereinfluß läßt sich durch Umordnung reduzieren.

- $f[x_0, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}$, für ein $\xi \in \mathcal{C}[x_0, \ldots, x_n]$.

$\longrightarrow$ Übung 4.76, 4.77: einfache Interpolationsaufgabe, mehr zu Dividierten Differenzen.

Dividierte Differenzen lassen sich auf zusammenfallende Stützstellen erweitern falls $f(x_j) = f(x_i)$.

$$\begin{aligned}
f[x_k, x_k] &:= \lim_{h\to 0} \frac{f[x_k + h] - f[x_k]}{x_k + h - x_k} = f'(x_k) \\
f[x_k, x_k, x_k] &:= \lim_{h\to 0} \frac{f[x_k + h, x_k + 2h] - f[x_k, x_k + h]}{x_k - 2h - x_k} \\
&= \lim_{h\to 0} \frac{f[x_k + 2h] - 2f[x_k + h] + f[x_k]}{2h^2} \\
&= \frac{1}{2} f''(x_k) \\
f[\underbrace{x_k, \ldots, x_k}_{m+1 \text{ mal}}] &:= \frac{1}{m!} f^{(m)}(x_k).
\end{aligned}$$

Damit kann man dann allgemeine Hermite Interpolation durchführen, bei der für mehrfache Stützstellen jeweils die entsprechenden Ableitungen mit interpoliert werden.

Beispiel 4.21

	c_0		c_1		c_2		c_3		c_4		c_5
x_0	$f[x_0]$	$\searrow$									
			$f'(x_0)$	$\searrow$							
x_0	$f[x_0]$	$\nearrow$			$\frac{1}{2}f''(x_0)$	$\searrow$					
		$\searrow$	$f'(x_0)$	$\nearrow$			$f[x_0, x_0, x_0, x_1]$	$\searrow$			
x_0	$f[x_0]$	$\nearrow$		$\searrow$	$f[x_0, x_0, x_1]$	$\nearrow$			$f[x_0, x_0, x_0, x_1, x_1]$	$\searrow$	
		$\searrow$	$f[x_0, x_1]$	$\nearrow$		$\searrow$	$f[x_0, x_0, x_1, x_1]$	$\nearrow$			$f[x_0, x_0, x_0, x_1, x_1, x_1]$
x_1	$f[x_1]$	$\nearrow$		$\searrow$	$f[x_0, x_1, x_1]$	$\nearrow$			$f[x_0, x_0, x_1, x_1, x_1]$	$\nearrow$	
		$\searrow$	$f'(x_1)$	$\nearrow$		$\searrow$	$f[x_0, x_1, x_1, x_1]$	$\nearrow$			
x_1	$f[x_1]$	$\nearrow$			$\frac{1}{2}f''(x_1)$	$\nearrow$					
		$\searrow$	$f'(x_1)$	$\nearrow$							
x_1	$f[x_1]$	$\nearrow$									

Das Interpolationspolynom ist damit

$$\begin{aligned}
P_{000111}(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)^2 + c_3(x - x_0)^3 \\
&\quad + c_4(x - x_0)^3(x - x_1) + c_5(x - x_0)^3(x - x_0)^2.
\end{aligned}$$

$\longrightarrow$ *Übung*

Beispiel 4.22 $x_0, x_0, x_1, x_1, \ldots$ *Schema ergibt*

$$\begin{aligned}
P_{001122\ldots kk}(x) &= [c_0 + c_1(x - x_0)] + [c_2 + c_3(x - x_1)](x - x_0)^2 \\
&\quad + [c_4 + c_5(x - x_2)](x - x_0)^2(x - x_1)^2 + \cdots \\
&\quad + [c_{2k} + c_{2k-1}(x - x_k)](x - x_0)^2 \cdots (x - x_{k-1})^2
\end{aligned}$$

Es gilt die Bedingung

$$\left.\begin{array}{rcl} P_{00\ldots kk}(x_i) & = & f(x_i) \\ P'_{00\ldots kk}(x_i) & = & f'(x_i) \end{array}\right\} i = 0, \ldots, k$$

$\longrightarrow$ *Übung*

Fehlerabschätzung: Angenommen $f_i = f(x_i) \quad i = 0, \ldots, n$.
Wie gut ist die Approximation?

Satz 4.23 *Es sei* $f \quad n+1$*–mal differenzierbar, so gibt es zu jedem* $\hat{x}$ *eine Zahl* ξ *aus dem kleinsten Intervall* I*, daß alle* x_i *und* $\hat{x}$ *enthält (konvexe Hülle* $\mathcal{C}[x_i, \hat{x}]$*), so daß*

$$f(\hat{x}) - P_{01\ldots n}(\hat{x}) = w(\hat{x}) \frac{f^{(n+1)}(\xi)}{(n+1)!} \tag{4.24}$$

wobei $w(x) = \prod_{i=0}^{n}(x - x_i)$.

Beweis: Sei für $\hat{x} \neq x_i$

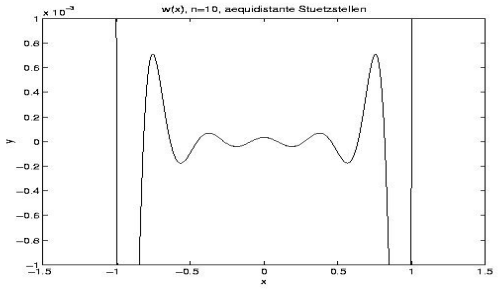
$$F(x) := f(x) - P(x) - kw(x).$$

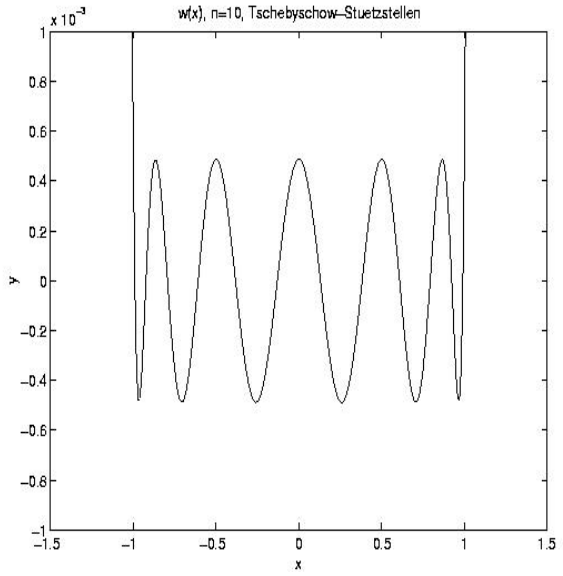
Bestimme k derart, daß $F(\hat{x}) = 0$. Dann hat $F(x)$ in $\mathcal{C}[x_0, \ldots, x_n, \hat{x}] \quad n+2$ Nullstellen $x_0, \ldots, x_n, \hat{x}$.
Nach dem Satz von Rolle hat dann $F'(x)$ mindestens $n+1$ Nullstellen, ..., $F^{(n+1)}(x)$ mindestens eine Nullstelle $\xi \in \mathcal{C}[x_0, \ldots, x_n \hat{x}]$.

$$\begin{array}{rcl} P^{(n+1)}(x) \equiv 0 & \Longrightarrow & F^{(n+1)}(\xi) = f^{(n+1)}(\xi) - k(n+1)! = 0 \\ & \Longrightarrow & k = \dfrac{f^{(n+1)}(\xi)}{(n+1)!}. \end{array}$$

□

Wie sieht $w(x)$ aus?





Wächst außerhalb von $\mathcal{C}[x_0, \ldots, x_n]$ stark an $\Longrightarrow$ Verwendung des Interpolationspolynoms außerhalb von $\mathcal{C}[x_0, \ldots, x_n]$ vermeiden.

Sei nun $f : [a,b] \to \mathbb{R}$ gegeben, f genügend glatt. Dann kann ich f beliebig genau durch Polynome interpolieren. Man könnte nun vermuten, daß die Interpolationspolynome gegen f konvergieren, falls ich die Intervallunterteilung feiner und feiner mache.

Betrachte Folge von Unterteilungen

$$\begin{aligned} \triangle_m &= \{a = x_0^{(m)} < x_1^{(m)} < \cdots < x_{n_m}^{(m)} = b\} \\ \| \triangle_m \| &= \max_i |x_{i+1}^{(m)} - x_i^{(m)}| \end{aligned}$$

Satz 4.25 *[Satz von Faber] Zu jeder Folge von Intervalleinteilungen $\triangle_m$ von $[a,b]$ kann man eine auf $[a,b]$ stetige Funktion f finden, so daß die Polynome $P_{\triangle_m}$ die auf $(x_i^{(m)}, f(x_i^{(m)})), i = 0,\ldots,n_m$ interpolieren für $m \to \infty$ NICHT gleichmäßig gegen $f(x)$ konvergieren.*

Nur für ganze Funktionen gilt für alle $\| \triangle_m \| \to 0$, $P_m \to f$ gleichmäßig.

Für beliebige stetige Funktionen gibt es spezielle $\triangle_m$, i.a. nicht äquidistant.

$\longrightarrow$ Übung 4.82: Tschebyschowknoten für $f(x) = \frac{1}{1+25x^2}$.
$\longrightarrow$ Übung: Orthogonale Polynome für Approximation.
Rationale Interpolation, hier nicht, Übung?

4.2 Trigonometrische Interpolation

Interpolation durch trigonometrische Polynome
Betrachte Intervall $[0, 2\pi]$, Stützpunkte $(x_k, f_k),\ k = 0,\ldots,n-1$.

$$\begin{aligned} x_k &= k\frac{2\pi}{n} \quad f_k \in \mathbb{C}, n \text{ fest} . \\ P(x) &= \beta_0 + \beta_1 e^{ix} + \beta_2 e^{2ix} + \cdots + \beta_{n-1} e^{(n-1)xi} \end{aligned}$$

soll die Interpolationsbedingungen

$$p(x_k) = f(x_k),\ k = 0,\ldots,n-1 \tag{4.26}$$

erfüllen. Periodizität $\Longrightarrow$

$$p(x_k) = f(x_k),\ k = \ldots,-2,-1,\ldots. \tag{4.27}$$

Setze

$$w = e^{ix},\ w_k = e^{ix_k} \tag{4.28}$$

Dann ist die Interpolationsaufgabe identisch mit der komplexen Interpolationsaufgabe

$$p(w_k) = f_k \tag{4.29}$$

mit

$$p = \beta_0 + \beta_1 w + \cdots + \beta_{n-1} w^{n-1}$$

Also folgt aus Satz 4.2 sofort

Satz 4.30 *Sei $x_k = \frac{2\pi k}{n}$, f_k beliebig für $k = 0, \ldots, n-1, n \in \mathbb{N}$. Dann gibt es ein eindeutiges trigonometrisches Polynom $p(w) = \beta_0 + \beta_1 w + \cdots + \beta_{n-1} w^{n-1}$ mit $p(w_k) = f_k$, für $w_k = e^{ix_k}$, $k = 0, \ldots, n-1$.*
Zusätzlich gilt

$$i) \quad w_k^j = w_j^k \quad \forall j, k \in \mathbb{Z}$$
$$ii) \quad \sum_{k=0}^{n-1} w_k^j w_k^{-l} = \begin{cases} n & \text{für } j = l \\ 0 & \text{für } j \neq l, 0 \leqslant j, l \leqslant n-1. \end{cases}$$

Beweis: i) trivial, ii) für alle $k = \mathbb{Z}$ ist w_k eine Wurzel von

$$w^n - 1 = (w-1)(w^{n-1} + w^{n-2} + \cdots + 1) = 0$$

für $k \neq 0, \pm n, \pm 2n$, ist $w_k \neq 1$, also folgt

$$\sum_{l=0}^{n-1} w_k^l = \begin{cases} n & k = 0, \pm n, \pm 2n, \ldots \\ 0 & \text{sonst.} \end{cases}$$

Die n–Vektoren $\Phi_j = (w_0^j, \ldots, w_{n-1}^j)$, $j = 0, \ldots, n-1$ bilden eine Orthonormalbasis von $\mathbb{C}^n$ unter dem Skalarprodukt

$$[f, g] := \frac{1}{n} \sum_{j=0}^{n-1} f_j \bar{g}_j \tag{4.31}$$

(diskrete Version von $(f, g) = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) g(\bar{x}) dx$). Denn es gilt wegen (4.27)

$$[\Phi_j, \Phi_l] = \begin{cases} 1 & j = l \\ 0 & j \neq l \quad 0 \leqslant j, l, \leqslant n-1. \end{cases}$$

Damit erhält man eine explizite Formel für die Koeffizienten β_i. □

Satz 4.32 *Sei $p(x) = \sum_{k=0}^{n-1} \beta_k e^{ikx}$ das interpolierende trigonometische Polynom für $p(x_k) = f_k$, $k = 0, \ldots, n-1$, so gilt*

$$\beta_j = \frac{1}{n} \sum_{k=0}^{n-1} f_k w_k^{-j} = \frac{1}{n} \sum_{k=0}^{n-1} f_k \exp\left\{ \frac{-2\pi i k j}{n} \right\}, \; j = 0, \ldots, n-1. \tag{4.33}$$

Beweis: Aus dem Skalarprodukt $[\bullet, \bullet]$ folgt

$$\frac{1}{n} \sum_{k=0}^{n-1} f_k w_k^{-j} = [f, \Phi_j] = [\beta_0 \Phi_0 + \cdots + \beta_{n-1} \Phi_{n-1}, \Phi_j] = \beta_j.$$

Es gilt die folgende Minimaleigenschaft der trigonometrischen Interpolation. □

Satz 4.34 *Unter allen trigonometrischen Polynomen q_s der Form $q_s(x) = \gamma_0 + \gamma_1 e^{ix} + \cdots + \gamma_s e^{isx}$, $s \leqslant n-1$, minimiert das s–te Abschnittspolynom $p_s(x) = \beta_0 + \beta_1 e^{xi} + \cdots + \beta_s e^{sxi}$ für $s = 0, \ldots, n-1$ die Fehlerquadratsumme*

$$S(q_s) = \sum_{k=0}^{n-1} |f_k - q_s(x_k)|^2 \tag{4.35}$$

Beweis: Ordne p_s, q_s die n–Tupel

$$\begin{aligned}\underline{p_s} &= (p_s(x_0), \ldots, p_s(x_{n-1})) \\ \underline{q_s} &= (q_s(x_0), \ldots, q_s(x_{n-1}))\end{aligned}$$

zu, die p_s, q_s eindeutig festlegen. Dann gilt
$\frac{1}{n}S(q_s) = [f - \underline{q_s}, f - \underline{q_s}]$. Da aber
$\beta_k = [f, \Phi_k]$ für $k = 0, \ldots, n-1$ folgt für $f \in \mathbb{C}^n$
$[f - \underline{p_s}, \Phi_j] = [f - \sum_{k=0}^{s} \beta_k \Phi_k, \Phi_j] = \beta_j - \beta_j = 0, j = 0, \ldots, s$ und

$$\begin{aligned}&[f - \underline{p_s}, \underline{p_s} - \underline{q_s}] = \sum_{j=0}^{s} [f - \underline{p_s}, (\beta_j - \gamma_j)\Phi_j] = 0, \\ &\tfrac{1}{n}S(q_s) = [f - \underline{q_s}, f - \underline{q_s}] = [(f - \underline{p_s} + \underline{p_s} - \underline{q_s}), (f - \underline{p_s} + \underline{p_s} - \underline{q_s}) \\ &= [f - \underline{p_s}, f - \underline{p_s}] + [\underline{p_s} - \underline{q_s}, \underline{p_s} - \underline{q_s}] \\ &\geqslant [f - \underline{p_s}, f - \underline{p_s}] = \tfrac{1}{n}S(p_s)\end{aligned}$$

und Gleichheit nur für $\underline{p_s} = \underline{q_s}$ da $[\bullet, \bullet]$ eine Norm definiert. □

Reelle Version: Beachte $e^{ix} = \cos x + i \sin x$. Setze

$$\begin{aligned}A_j &= \frac{2}{n} \sum_{k=0}^{n-1} f_k \cos \frac{2\pi jk}{n} \\ B_j &= \frac{2}{n} \sum_{k=0}^{n-1} f_k \sin \frac{2\pi jk}{n}\end{aligned} \tag{4.36}$$

für reelle f_k sind A_j, B_j reell und es gilt
$\longrightarrow$ Übung

$$\beta_{n-j} = \frac{1}{n} \sum_{k=0}^{n-1} f_k w^{j-n} w_k = \frac{1}{n} \sum_{k=0}^{n-1} f_k w_k^j \tag{4.37}$$

$$\beta_j = \frac{1}{2}[A_j - iB_j], \beta_{n-j} = \frac{1}{2}[A_j + iB_j] \tag{4.38}$$

$$\beta_j w_k^j + \beta_{n-j} w_k^{n-j} = A_j \cos jx_k + B_j \sin jx_k \tag{4.39}$$

Damit erhält man die reelle diskrete Fouriertransformation. A_j, B_j sind diskrete Approximationen an die Fourierkoeffizienten

$$\begin{aligned}a_k &= \tfrac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx)\, dx, \quad k = 0, \ldots, n-1 \\ b_k &= \tfrac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(kx)\, dx, \quad k = 1, \ldots, n-1.\end{aligned}$$

$\longrightarrow$ Übung bei Quadratur

Satz 4.40 *Gegeben* (x_k, f_k), $k = 0, \ldots, N-1$ *mit* $x_k = \frac{2\pi k}{N}$. *Man setze für* $j \in \mathbb{Z}$

$$\left.\begin{aligned}A_j &:= \tfrac{2}{N} \sum_{k=0}^{N-1} f_k \cos(kx_j) \\ B_j &:= \tfrac{2}{N} \sum_{k=0}^{N-1} f_k \sin(kx_j)\end{aligned}\right\} \tag{4.41}$$

Definiere für ungerades $N = 2M + 1$

$$\Psi(x) = \frac{A_0}{2} + \sum_{k=1}^{M}(A_k \cos kx + B_k \sin kx) \tag{4.42}$$

und für gerades $N = 2M$

$$\Psi(x) = \frac{A_0}{2} + \sum_{k=1}^{M-1}(A_k \cos kx + B_k \sin kx) + \frac{A_M}{2}\cos Mx \tag{4.43}$$

so gilt

$$\Psi(x_k) = f_k,\ k = 0, \ldots, N-1.$$

Beweis: $N = 2M \Longrightarrow$

$$f_k = \sum_{j=0}^{N-1} \beta_j w_k^j = \beta_0 + \sum_{j=1}^{M-1}(\beta_j w_k^j + \beta_{N-j} w_k^{(N-j)}) + \beta_M w_k^M$$

mit (4.38), (4.39) folgt

$$B_0 = \tfrac{2}{N} \sum_{k=0}^{N-1} f_k \sin \tfrac{2\pi k \cdot 0}{N} = 0,$$
$$B_M = \tfrac{2}{N} \sum_{k=0}^{N-1} \sin \tfrac{2\pi k \cdot 1}{N} = 0.$$

Analog für ungerades N. □

Berechnung von Fouriertransformation
Fast Fourier Transformation (FFT) für $N = 2^n \quad n > 0$. Was ist zu berechnen?

$$\begin{aligned} \beta_j &= \frac{1}{N}\sum_{k=0}^{N-1} f_k e^{\frac{-2\pi i k j}{N}} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} f_k \epsilon_N^{kj} \end{aligned}$$

mit $\epsilon_N = e^{\frac{-2\pi i}{N}}$.
Der Trick ist die Tasache auszunutzen, daß die $e^{\frac{-2\pi i}{N}}$ eine N–te Einheitswurzel ist, d.h. alle Potenzen liegen auch auf dem Einheitskreis und die Exponenten lassen sich nach N reduzieren.
Betrachte als Beispiel $N = 4$

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \tfrac{1}{N} \begin{bmatrix} \epsilon_N^0 & \epsilon_N^0 & \epsilon_N^0 & \epsilon_N^0 \\ \epsilon_N^0 & \epsilon_N^1 & \epsilon_N^2 & \epsilon_N^3 \\ \epsilon_N^0 & \epsilon_N^2 & \epsilon_N^4 & \epsilon_N^6 \\ \epsilon_N^0 & \epsilon_N^3 & \epsilon_N^6 & \epsilon_N^9 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = \tfrac{1}{N} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \epsilon_N^1 & \epsilon_N^2 & \epsilon_N^3 \\ 1 & \epsilon_N^2 & 1 & \epsilon_N^2 \\ 1 & \epsilon_N^3 & \epsilon_N^2 & \epsilon_N^1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

$$\begin{bmatrix} \beta_0 \\ \beta_2 \\ \beta_1 \\ \beta_3 \end{bmatrix} = \tfrac{1}{N} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \epsilon_N^2 & 1 & \epsilon_N^2 \\ 1 & \epsilon_N & \epsilon_N^2 & \epsilon_N^3 \\ 1 & \epsilon_N^3 & \epsilon_N^2 & \epsilon_N \end{bmatrix} = \tfrac{1}{N} \left[\begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 1 & \epsilon_N^2 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & \epsilon_N^2 \end{array}\right] \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 1 & 0 & \epsilon_N^2 & 0 \\ 0 & \epsilon_N & 0 & \epsilon_N^3 \end{array}\right] \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

Bilde Teilschritte: erst

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & \epsilon_N^2 & 0 \\ 0 & \epsilon_N & 0 & \epsilon_N^3 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

Verwende $\epsilon_N^2 = -1, \epsilon_N^3 = -\epsilon_N$

$$\begin{aligned} z_0 &= f_0 + f_2 \quad z_2 = f_0 - f_2 \\ z_1 &= f_1 + f_3 \quad z_3 = \epsilon_N (f_1 - f_3) \end{aligned}$$

$$\begin{bmatrix} \beta_0 \\ \beta_2 \\ \beta_1 \\ \beta_3 \end{bmatrix} = \frac{1}{N} \underbrace{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & \epsilon_N^2 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & \epsilon_N^2 \end{bmatrix}}_{\text{2 komplexe FT der Ordnung 2 mit } \epsilon_N^2} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \frac{1}{N} \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

$$\Longrightarrow \begin{aligned} N\beta_0 &= z_0 + 1 z_1, \quad N\beta_1 = z_2 + z_3 \\ N\beta_2 &= z_0 - z_1, \quad N\beta_3 = z_2 - z_3 \end{aligned}$$

Es findet jeweils eine Rückführung einer FFT der Ordnung $N = 2^2$ auf 2 FFT der Ordnung $\frac{N}{2} = 2^{n-1}$ statt $+\mathcal{O}(N)$ Arbeit. Anstatt N^2 für Matrixmultiplikation erhält man dann $\frac{1}{2} N \log_2 N = 2^{n-1} \times (n-1)$ Operationen.

Da die FFT halber Dimension unabhängig laufen können lassen sie sich gut auf Parallelrechner implementieren.
$\longrightarrow$ Übung

Allgemeine Formeln:

$$\text{Sei } M = \frac{N}{2}, \text{ Setze } j = 2l$$

$$\begin{aligned} \beta_{2l} &= \sum_{k=0}^{2M-1} f_k \epsilon_N^{2lk} = \sum_{k=0}^{M-1} (f_k + f_{M+k}) \epsilon_N^{2lk} \\ &= \sum_{k=0}^{M-1} (f_k + f_{M+k}) (\epsilon_N^2)^{lk}, \end{aligned} \tag{4.44}$$

wobei wir benutzen

$$\epsilon_N^{2l(M+k)} = \epsilon_N^{2lk} \epsilon_N^{2lM} = \epsilon_N^{2lk}.$$

Mit Hilfswerten

$$z_k = f_k + f_{M+k}, \; k = 0, \ldots, M-1 \tag{4.45}$$

und mit $\epsilon_N^2 = \epsilon_M$ folgt

$$\beta_{2l} = \sum_{k=0}^{M-1} z_k \epsilon_M^{kl} \quad (l = 0, \ldots, M-1). \tag{4.46}$$

Für die ungeraden Indizes gilt analog

$$\beta_{2l+1} = \sum_{k=0}^{M-1} z_{M+k} \epsilon_M^{kl} \quad l = 0, \ldots, m-1 \tag{4.47}$$

mit Hilfswerten

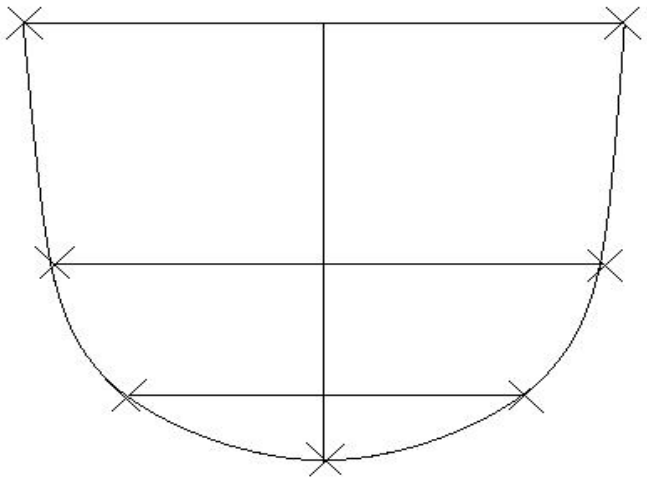
$$z_{M+k} = (f_k - f_{M+k} \epsilon_N^k), \; k = 0, \ldots, M-1 \tag{4.48}$$

Algorithmus von Cooley/Tukey (1965)
$\longrightarrow$ Übung 4.85, 4.86
Weitere Tricks möglich.

4.3 Spline Interpolation

Idee: Anstatt Polynome zu verwenden zur Interpolation, verwende stückweise Polynome. Spline kommt aus Schiffbau.

Latte um Schiffsrumpf



Anwendungen: CAD, numerische Lösung partieller + gewöhnlicher Differentialgleichungen. Mehr in Numerik III, Numerik IV; hier nur Grundlagen.

4.3.1 Allgemeine Eigenschaften

Durch $\triangle := \{a = x_0 < x_1 \cdots < x_n = b\}$ sei Unterteilung von $[a, b]$ gegeben.

Definition 4.49 *Unter einer zu $\triangle$ gehörenden Spline–Funktion $S_\triangle$ versteht man eine reelle Funktion $S_\triangle : [a, b] \to \mathbb{R}$ mit*

a) $S_\triangle \in \mathcal{C}^{2k}[a, b]$ (2k mal stetig differenzierbar).

b) Auf jedem Teilintervall $[x_i, x_{i+1}]$ stimmt $S_\triangle$ mit Polynom $2k + 1$–ten Grades überein.

Hier im folgenden $k = 1$, kubische Splines. Fast alles läßt sich übertragen.

Definition 4.50

$$\kappa^n(a, b) := \left\{ f : [a, b] \to \mathbb{R};\ \begin{array}{l} f^{(i)},\ i = 1, \ldots, n-1,\ \textit{existiert und ist absolut stetig,} \\ f^{(n)}\ \ \textit{existiert fast überall auf } [a, b] \textit{ und } f^{(n)} \in L^2[a, b] \end{array} \right\}$$

(f absolut stetig, falls zu jedem $\epsilon > 0$ ein $\delta > 0$ so daß für jedes endliche System von Intervallen $[a_i, b_i]$ mit $a \leqslant a_1 < b_1 \leqslant a_2 < b_2 \cdots \leqslant a_m < b_m \leqslant b$ mit $\sum_i |b_i - a_i| < \delta$ gilt: $\sum_i |f(b_i) - f(a_i)| < \epsilon$.)

$$\begin{aligned} \kappa_p^n(a, b) \quad = \quad & \{f \in \kappa^n(a, b), f, f', \ldots, f^{(n-1)}, b - a \textit{ periodisch, d.h.} \\ & f^{(i)}(a) = f^{(i)}(b) \quad i = 0, \ldots, n-1\}. \end{aligned}$$

Es folgt, daß $S_\triangle \in \kappa^{2k+1}(a,b)$.
Eine Spline–Funktion heißt periodisch falls

$$S_\triangle \in \kappa_p^{2k+1}(a,b).$$

Wir wollen mit Splines interpolieren, d.h. wir wollen, daß $S_\triangle(x_i) = f_i = f(x_i),\ i = 0,\dots,n$.
Dann $S_\triangle^f$. *Führe Maß* $w(f) := \int_a^b |f''(x)|^2\,dx$ *ein.*
$w(f) \geqslant 0$ *jedoch* $w(f) = 0 \quad \forall f = cx + 1$.

Satz 4.51 *Sei* $f \in \kappa^2(a,b), \triangle = \{a = x_0 < \cdots < x_n = b\}$ *und* $S_\triangle$ *kubischer Spline zu* $\triangle$. *Dann gilt*

$$w(f - S_\triangle) = w(f) - w(S_\triangle) - 2\left[(f' - S'_\triangle)S''_\Delta\big|_a^b - \sum_{i=1}^n (f - S_\triangle)S'''_\triangle\big|_{x_{i-1}^+}^{x_i^-}\right].$$

Beweis:

$$\begin{aligned} w(f - S_\triangle) &= \int_a^b |f'' - S''_\triangle|^2\,dx \\ &= \int_a^b |f''|^2\,dx - 2\int_a^b f''S''_\triangle dx + \int_a^b |S''_\triangle|^2\,dx \\ &= w(f) - 2\int_a^b (f'' - S''_\triangle)S''_\triangle\,dx - w(S_\triangle). \end{aligned}$$

Partielle Integration in Teilintervallen.

$$\begin{aligned} \int_{x_{i-1}}^{x_i} (f'' - S''_\triangle)S''_\triangle dx &= (f' - S'_\triangle)S''_\triangle\big|_{x_{i-1}}^{x_i} - \int_{x_{i-1}}^{x_i} (f' - S'_\triangle)S'''_\triangle dx \\ &= (f' - S'_\triangle)S''_\triangle\big|_{x_{i-1}}^{x_i} - (f - S_\triangle)S'''_\triangle\big|_{x_{i-1}^+}^{x_i^-} + \int_{x_{i-1}}^{x_i} (f - S_\triangle)S_\triangle^{(4)}\,dx. \end{aligned}$$

$S^{(4)} \equiv 0$ auf allen $[x_{i-1}, x_i]$ und

$$\sum_{i=1}^n (f' - S'_\triangle)S''_\triangle\big|_{x_{i-1}}^{x_i} = (f' - S'_\Delta)S''_\triangle\big|_a^b$$

impliziert Beh. □

Dies ist Grundlage für Minimaleigenschaft der Splines.

Satz 4.52 *Sei* $\triangle = \{a = x_0 < x_1 \cdots < x_n = b\}$.
$F = \{f_0,\dots,f_n\}, f \in \kappa^2(a,b)$ *mit* $f(x_i) = f_i,\ i = 0,\dots,n$. *Dann gilt*

$$w(f - S_\triangle^f) = w(f) - w(S_\triangle^f) \geqslant 0 \tag{4.53}$$

für alle interpolierenden $S^f_\triangle$ die einer der 3 folgenden Bedingungen erfüllen.

$$a) \quad S''_\triangle(a) = S''_\triangle(b) = 0. \tag{4.54}$$

$$\textit{oder } b) \quad f \in \kappa^2_p(a,b) \quad S_\triangle \textit{ periodisch .} \tag{4.55}$$

$$\textit{oder } c) \quad f'(a) = S'_\triangle(a), f'(b) = S'_\triangle(b). \tag{4.56}$$

Für jeden dieser Fälle ist $S_\triangle$ eindeutig bestimmt.

Beweis: In jedem dieser Fälle verschwindet

$$\big((f' - S'_\triangle)S''_\triangle\big)\big|_b^a - \sum_{i=1}^{n} (f - S_\triangle)S'''_\triangle\big|_{x_{i-1}^+}^{x_i^-} = 0.$$

Eindeutigkeit folgt so. Angenommen es gäbe $\tilde{S}^f_\triangle$ mit den gleichen Eigenschaften. Dann kann ich $f(x) = \tilde{S}^f_\triangle(x)$ setzen. $\Longrightarrow$
$w(\tilde{S}^f_\triangle(x) - S^f_\triangle(x)) = w(\tilde{S}^f_\triangle(x)) - w(S^f_\triangle(x)) \geqslant 0$ und genauso $w(S^f_\triangle(x)) - w(\tilde{S}^f_\triangle(x)) \geqslant 0$
also $\int_a^b (\tilde{S}''_\triangle - S''_\triangle)^2 dx = 0$ beide $\tilde{S}'', S''$ stetig $\Longrightarrow \tilde{S}''_\triangle = S''_\triangle \Longrightarrow \tilde{S}^f_\triangle = S^f_\triangle + cx + d$, wegen $\tilde{S}^f_\triangle(x) = S^f_\Delta(x)$ für a, b folgt $c = d = 0$. □

Dieser Satz besagt, daß die Splines unter allen Funktionen aus $\kappa^2(a,b)$ die interpolierende Funktion mit der geringsten Krümmung sind.
$\kappa(x) = \frac{f''(x)}{\sqrt{1+f'(x)^2}^3} \approx f''(x)$ für $f'(x)$ klein.

Analoge Eigenschaften für gelten andere Splines.

4.3.2 Berechnung der Splines

Setze $h_{j+1} = x_{j+1} - x_j,\ j = 0, \ldots, n-1$ und

$$M_j = S''_\triangle(x_j),\ j = 0, \ldots, n. \tag{4.57}$$

M_j heißen Momente (aus Physik 2. Ableitungen). Berechne Splines die (4.54), (4.55) oder (4.32) erfüllen mit Hilfe der Momente.
Da $S_\triangle$ kubisch in $[x_i, x_{i+1}]$ ist, folgt daß $S''_\triangle$ linear in $[x_i, x_{i+1}]$ und es gilt

$$S''_\triangle(x) = M_j \frac{x_{j+1} - x}{h_{j+1}} + M_{j+1} \frac{x - x_j}{h_{j+1}},\ x \in [x_j, x_{j+1}] \ \text{ (2 Punkte Formel)} \tag{4.58}$$

Integration

$$S'_\triangle(x) = -M_j \frac{(x_{j+1} - x)^2}{2h_{j+1}} + M_{j+1} \frac{(x - x_j)^2}{2h_{j+1}} + \alpha_j \tag{4.59}$$

$$S_\triangle(x) = M_j \frac{(x_{j+1} - x)^3}{6h_{j+1}} + M_{j+1} \frac{(x - x_j)^3}{6h_{j+1}} + \alpha_j(x - x_j) + \beta_j. \tag{4.60}$$

Interpolationsbedingungen

$$\begin{aligned} S_\Delta(x_j) &= f_j \\ \Longrightarrow M_j\frac{h_{j+1}^2}{6}+\beta_j &= f_j \Longrightarrow \beta_j = f_j - M_j\frac{h_{j+1}^2}{6}, \\ S_\Delta(x_{j+1}) &= f_{j+1} \\ \Longrightarrow M_{j+1}\frac{h_{j+1}^2}{6}+\alpha_j h_{j+1}+\beta_j &= f_{j+1} \Longrightarrow \alpha_j = \frac{f_{j+1}-f_j}{h_{j+1}} - \frac{h_{j+1}}{6}(M_{j+1}-M_j), \end{aligned}$$

und damit

$$S_\Delta(x) = a_j + b_j(x-x_j) + c_j(x-x_j)^2 + d_j(x-x_j)^3 \tag{4.61}$$

mit

$$\begin{aligned} a_j &= f_j, c_j = \frac{M_j}{2}, b_j = -\frac{M_j h_{j+1}}{2} + \alpha_j = S'_\Delta(x_j) \\ &= \frac{f_{j+1}-f_j}{h_{j+1}} - \frac{2M_j + M_{j+1}h_{j+1}}{6}, d_j = \frac{M_{j+1}-M_j}{6h_{j+1}} = \frac{S'''_\Delta(x_j^+)}{6} \end{aligned} \tag{4.62}$$

Wir brauchen also nur die M_j zu berechnen. Wir haben gefordert, daß $S'_\Delta(x)$ an den Knoten $x = x_j,\ j = 1,\ldots,n-1$ stetig sein soll. $\Longrightarrow n-1$ Gleichungen. Es gilt

$$\begin{aligned} S'_\Delta(x) &= -M_j\frac{(x_{j+1}-x)^2}{2h_{j+1}} + M_{j+1}\frac{(x-x_j)^2}{2h_{j+1}} + \frac{f_{j+1}-f_j}{h_{j+1}} \\ &\quad -\frac{h_{j+1}}{6}(M_{j+1}-M_j) \end{aligned} \tag{4.63}$$

wenn man die Werte einsetzt. Damit ergibt sich für $j = 1,\ldots,n-1$

$$\begin{aligned} S'_\Delta(x_j^-) &= \frac{f_j - f_{j-1}}{h_j} + \frac{h_j}{3}M_j + \frac{h_j}{6}M_{j-1} \\ S'_\Delta(x_j^+) &= \frac{f_{j+1}-f_j}{h_{j+1}} - \frac{h_{j+1}}{3}M_j - \frac{h_{j+1}}{6}M_{j+1} \end{aligned} \tag{4.64}$$

Gleichsetzen wegen Stetigkeit $\Longrightarrow$

$$\begin{aligned} \frac{h_j}{6}M_{j-1} + \frac{h_j+h_{j+1}}{3}M_j + \frac{h_{j+1}}{6}M_{j+1} &= \frac{f_{j+1}-f_j}{h_{j+1}} - \frac{f_j-f_{j-1}}{h_j} \\ &= f[x_j, x_{j+1}] - f[x_{j-1}, x_j],\ j = 1,\ldots,n-1 \end{aligned} \tag{4.65}$$

Das sind $n-1$ Gleichungen für $M_0,\ldots,M_n$.
Die zwei weiteren notwendigen Bedingungen erhält man aus (4.54), (4.55), (4.56).
Aus (4.54)

$$S''_\Delta(a) = M_0 = M_n = S''_\Delta(b) = 0.$$

Aus (4.55)

$$\begin{aligned} S''_\Delta(a) = S''_\Delta(b) &\Longrightarrow M_0 = M_n. \\ S'_\Delta(a) = S'_\Delta(b) &\Longrightarrow \frac{h_n}{6}M_{n-1} + \frac{h_n+h_1}{3}M_n + \frac{h_1}{6}M_1 \\ &\quad = \frac{f_1-f_n}{h_1} - \frac{f_n-f_{n-1}}{h_n} \end{aligned}$$

Aus $f_n = f_0$ und mit der Setzung
$h_{n+1} := h_1$ $M_{n+1} := M_1$, $f_{n+1} := f_1$ erhält man wieder Gleichung (4.65).
Aus (4.56)

$$\begin{aligned} S'_\Delta(a) = f'_0 \quad &\Longrightarrow \quad \frac{h_1}{3}M_0 + \frac{h_1}{6}M_1 = \frac{f_1 - f_0}{h_1} - f'_0 \\ S'_\Delta(b) = f'_n \quad &\Longrightarrow \quad \frac{h_n}{6}M_{n-1} + \frac{h_n}{3}M_n = f'_n - \frac{f_n - f_{n-1}}{h_n}. \end{aligned}$$

Führe nun folgende Abkürzungen ein:

$$\lambda_j := \frac{h_{j+1}}{h_j + h_{j+1}},\ \mu_j := 1 - \lambda_j = \frac{h_j}{h_j + h_{j+1}}$$

Dann erhält man das System von linearen Gleichungen

$$\mu_j M_{j-1} + 2M_j + \lambda_j M_{j+1} = 6f[x_{j-1}, x_j, x_{j+1}] =: d_j \tag{4.66}$$

d.h. die folgenden Gleichungssysteme:

$$\begin{bmatrix} 2 & \lambda_0 & & & \\ \mu_1 & 2 & \lambda_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \lambda_{n-1} \\ & & & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ \vdots \\ \vdots \\ \vdots \\ M_n \end{bmatrix} = \begin{bmatrix} d_0 \\ \vdots \\ \vdots \\ \vdots \\ d_n \end{bmatrix} \tag{4.67}$$

im Fall von Bedingung (4.54), wobei
$\lambda_0 := 0, d_0 = 0, \mu_n := 0, d_n := 0$ gesetzt wurde.

$$\begin{bmatrix} 2 & \lambda_1 & & & & \mu_1 \\ \mu_2 & 2 & \lambda_2 & & & \\ & \mu_3 & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & & & \lambda_{n-1} \\ \lambda_n & & & & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ \vdots \\ \vdots \\ \vdots \\ M_n \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ \vdots \\ \vdots \\ d_n \end{bmatrix} \tag{4.68}$$

im Fall von Bedingung (4.55), wobei

$$\lambda_n := \frac{h_1}{h_n + h_1}, \mu_n = 1 - \lambda_n, d_n = 6f[x_{n-1}, x_n, x_1]$$

Im Fall von Bedingung (4.56) erhält man wiederum System (4.67) nur mit

$$\lambda_0 = 1, \mu_n = 1, d_0 = \tfrac{6}{h_1}\left(f[x_0, x_1] - f'_0\right), d_n = \tfrac{6}{h_n}\left(f'_n - f[x_{n-1}, x_n]\right).$$

Für alle 3 Gleichungssysteme gilt $\lambda_i \geqslant 0, \mu_i \geqslant 0, \lambda_i + \mu_i \leqslant 1$ und der Satz

Satz 4.69 *Die Matrizen der Systeme (4.67), (4.68) sind für jede Zerlegung Δ von $[a, b]$ nichtsingulär.*

Beweis: Sei

$$A = \begin{bmatrix} 2 & \lambda_0 & & \\ \mu_1 & \ddots & \ddots & \\ & \ddots & \ddots & \lambda_{n-1} \\ & & \mu_n & 2 \end{bmatrix}$$

Sei $x, y \in \mathbb{R}^{n+1}$ mit $Ax = y$ und setze $\lambda_n = \mu_0 = 0$. Dann gilt

$$\max_i |x_i| \leqslant \max_i |y_i| \quad \text{(A ist monoton)}.$$

Denn sei r der Index für den $\max_i |x_i|$ angenommen wird.
Dann gilt

$$\begin{aligned} \mu_r x_{r-1} + 2x_r + \lambda_r x_{r+1} &= y_r \\ \Longrightarrow \max_i |y_i| \geqslant |y_r| &\geqslant 2|x_r| - \mu_r|x_{r-1}| - \lambda_r|x_{r+1}| \\ &\geqslant 2|x_r| - (\mu_r + \lambda_r)|x_r| \\ &\geqslant |x_r| = \max_i |x_i|. \end{aligned}$$

Angenommen A singulär $\Longrightarrow x \neq 0$ mit $Ax = 0$.
Monotonie führt sofort zum Widerspruch. Anderes System analog. □

Lösung der Gleichungssysteme:
$\longrightarrow$ Übung: Gauß ohne Pivot (Diagonaldominanz).
Zyklische Reduktion, Sherman–Morrison, iterative Verfahren; alles geht sehr gut.
$\longrightarrow$ Übung!

Typischerweise werden jedoch Bandlöser benutzt, da die Matrizen diagonalähnlich zu einer positiv definiten Matrix sind.

4.3.3 Fehlerabschätzung und Konvergenz bei Interpolation mit Splines

Sei $\Delta_m = \{a = x_0^{(m)} < x_1^{(m)} < \cdots < x_{n_m}^{(m)} = b\}$ eine Folge von Unterteilungen von $[a, b]$ und sei $\sigma_m = \max_i (x_{i+1}^{(m)} - x_i^{(m)})$.

Satz 4.70 *Sei $f \in \mathcal{C}^4[a, b]$ und $|f^{(4)}(x)| \leqslant L \ \forall x \in [a, b], \Delta_m$ wie oben mit $\sup_{m,j} \frac{\sigma_m}{x_{j+1}^{(m)} - x_j^{(m)}} \leqslant \kappa < \infty$. Seien S_{Δ_m} die zu f gehörigen Splines, die die Interpolationsbedingungen*

$$\begin{aligned} S_{\Delta_m}(\zeta) &= f(\zeta) \quad \text{für } \zeta \in \Delta_m \\ S'_{\Delta_m}(x) &= f'(x) \quad \text{für } x = a, b \end{aligned} \tag{4.71}$$

erfüllen. Dann gibt es von Δ_m unabhängige Konstanten $C_i \leqslant 2$, so daß für alle $x \in [a, b]$

$$\left| f^{(i)}(x) - S^{(i)}_{\Delta_m}(x) \right| \leqslant C_i L \kappa \sigma_m^{4-i}, \ i = 0, 1, 2, 3. \tag{4.72}$$

Beweis: Betrachte feste Zerlegung Δ von $[a, b]$. Die Momente M_j erfüllen für den Fall den wir hier haben, das Gleichungssystem

$$\begin{bmatrix} 2 & \lambda_0 & & \\ \mu_1 & 2 & \ddots & \\ & \ddots & \ddots & \lambda_{n-1} \\ & & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ \vdots \\ \vdots \\ M_n \end{bmatrix} \begin{bmatrix} d_0 \\ \vdots \\ \vdots \\ d_n \end{bmatrix}$$

$$\lambda_0 = \mu_n = 1, \lambda_j = \frac{h_{j+1}}{h_j + h_{j+1}}, \mu_j = 1 - \lambda_j,\ j = 1, \ldots, n-1$$

$$\begin{aligned} d_j &= 6f[x_{j-1}, x_j, x_{j+1}],\ j = 1, \ldots, n-1 \\ d_0 &= \frac{6}{h_1}\left(f[x_0, x_1] - f_0'\right), \\ d_n &= \frac{6}{h_n}\left(f_n' - f[x_{n-1}, x_n]\right). \end{aligned}$$

Definiere

$$F = \begin{bmatrix} f''(x_o) \\ \vdots \\ f''(x_n) \end{bmatrix}, r = d - AF, M = \begin{bmatrix} M_0 \\ \vdots \\ M_n \end{bmatrix}.$$

Dann gilt $r_0 = d_0 - 2f''(x_0) - f''(x_1)$. Schätze zuerst $\|r_0\|_\infty$ ab. Taylorentwicklung um x_0 liefert

$$\begin{aligned} r_0 &= \frac{6}{h_1}\left(f[x_0, x_1] - f_0'\right) - 2f''(x_0) - f''(x_1) \\ &= \frac{6}{h_1}\left(f'(x_0) + \frac{h_1}{2}f''(x_0) + \frac{h_1^2}{6}f'''(x_0) + \frac{h_1^3}{24}f^{(4)}(\tau_1) - f'(x_0)\right) \\ &\quad -2f''(x_0) - \left(f''(x_0) + h_1 f'''(x_0) + \frac{h_1^2}{2}f^{(4)}(\tau_2)\right) \\ &= \frac{h_1^2}{4}f^{(4)}(\tau_1) - \frac{h_1^2}{2}f^{(4)}(\tau_2),\ \text{für } \tau_1, \tau_2 \in [x_0, x_1] \end{aligned}$$

$$\Longrightarrow |r_0| \leqslant \frac{3}{4}L\sigma_m^2. \tag{4.73}$$

Analog gilt für $r_n = d_n - f''(x_{n-1}) - 2f''(x_n)$ die Abschätzung

$$|r_n| \leqslant \frac{3}{4}L\sigma_m^2. \tag{4.74}$$

und für $j = 1, \ldots, n-1$ mit Taylorentwicklung um x_j.

$$\begin{aligned} r_j &= d_j - \mu_j f''(x_{j-1}) - 2f''(x_j) - \lambda_j f''(x_{j+1}) \\ &= 6f[x_{j-1}, x_j, x_{j+1}] - \frac{h_j}{h_j + h_{j+1}}f''(x_{j-1}) - 2f''(x_j) - \frac{h_{j+1}}{h_j + h_{j+1}}f''(x_{j+1}) \\ &= \frac{1}{h_j + h_{j+1}}\left\{6\left(f'(x_j) + \frac{h_{j+1}}{2}f''(x_j) + \frac{h_{j+1}^2}{6}f'''(x_j) + \frac{h_{j+1}^3}{24}f^{(4)}(\tau_{j_1})\right.\right. \\ &\qquad \left. -f'(x_j) + \frac{h_j}{2}f''(x_j) - \frac{h_j^2}{6}f'''(x_j) + \frac{h_j^3}{24}f^{(4)}(\tau_{j_2})\right) \\ &\qquad - h_j\left(f''(x_j) - h_j f'''(x_j) + \frac{h_j^2}{2}f^{(4)}(\tau_{j_3})\right) \\ &\qquad - 2f''(x_j)(h_j + h_{j+1}) \\ &\qquad \left. - h_{j+1}\left(f''(x_j) + h_{j+1}f'''(x_j) + \frac{h_{j+1}^2}{2}f^{(4)}(\tau_{j_4})\right)\right\} \\ &= \frac{1}{h_j + h_{j+1}}\left(\frac{h_{j+1}^3}{4}f^{(4)}(\tau_{j_1}) + \frac{h_j^3}{4}f^{(4)}(\tau_{j_2}) - \frac{h_j^3}{2}f^{(4)}(\tau_{j_3}) - \frac{h_{j+1}^3}{2}f^{(4)}(\tau_{j_4})\right), \end{aligned}$$

mit $\tau_{j_i} \in [x_{j-1}, x_{j+1}],\ i = 1,2,3,4.$

$$\Longrightarrow |r_j| \leqslant \frac{3}{4} L \frac{h_{j+1}^3 + h_j^3}{h_j + h_{j+1}} \leqslant \frac{3}{4} L \sigma_m^2.$$

Insgesamt gilt

$$\|r\|_\infty \leqslant \frac{3}{4} L \sigma_m^2 \Longrightarrow$$

$$\|A(M-F)\|_\infty \leqslant \frac{3}{4} L \sigma_m^2 \Longrightarrow \|M-F\|_\infty \leqslant \frac{3}{4} L \sigma_m^2, \tag{4.75}$$

da A monoton.

Sei $x \in [x_{j-1}, x_j]$. Dann folgt

$$\begin{aligned} S_\Delta''' - f'''(x) &= \frac{M_j - M_{j-1}}{h_j} - f'''(x) \\ &= \frac{M_j - f''(x_j)}{h_j} - \frac{M_{j-1} - f''(x_{j-1})}{h_j} \\ &\quad + \frac{[f''(x_j) - f''(x)] - [f''(x_{j-1}) - f''(x)]}{h_j} - f'''(x). \end{aligned}$$

Taylorentwicklung um x und (4.75) $\Longrightarrow$

$$\begin{aligned} |S_\Delta'''(x) - f'''(x)| &\leqslant \frac{3}{2} L \frac{\sigma_m^2}{h_j} + \frac{1}{h_j} \Big| (x_j - x) f'''(x) + \frac{(x_j - x)^2}{2} f^{(4)}(\eta_1) - (x_{j-1} - x) f'''(x) \\ &\qquad - \frac{(x_{j-1} - x)^2}{2} f^{(4)}(\eta_2) - h_j f'''(x) \Big| \\ &\leqslant \frac{3}{2} L \frac{\sigma_m^2}{h_j} + \frac{1}{2} \frac{\sigma_m^2}{h_j} L,\ \tau_1, \tau_2 \in [x_{j-1}, x_j] \\ &\leqslant 2 L \kappa \sigma_m, \text{ wegen Vor. } \frac{\sigma_m}{h_j} \leqslant \kappa. \end{aligned}$$

Hierbei wurde noch $\max_{x \in [x_{j-1}, x_j]} \{(x_j - x)^2 + (x - x_{j-1})^2\} = h_j^2$ benutzt. Nun gibt es für jedes $x \in (a,b)$ ein x_j mit

$$\begin{aligned} |x_j - x| &\leqslant \frac{1}{2} \sigma_m \\ \Longrightarrow f''(x) - S_\Delta''(x) &= f''(x_j) - S_\Delta''(x_j) + \int_{x_j}^{x} \left(f'''(t) - S_\Delta'''(t) \right) dt \end{aligned}$$

$\Longrightarrow$ mit der Abschätzung für $f''' - S_\Delta''' \Longrightarrow$

$$\begin{aligned} \left| f''(x) - S_\Delta''(x) \right| &\leqslant \frac{3}{4} L \sigma_m^2 + \frac{1}{2} \sigma_m \cdot 2 L \kappa \sigma_m \\ &\leqslant \frac{7}{4} L \kappa \sigma_m^2, \text{ da } \kappa \geqslant 1. \end{aligned}$$

Damit haben wir die Abschätzung für $f'' - S_\Delta''$.
Es gilt $f(x_{j-1}) = S_\Delta(x_{j-1}), f(x_j) = S_\Delta(x_j)$. Außer den Randpunkten $\alpha_0 = a, \alpha_{n+1} = b$ gibt es also noch den Satz von Rolle n Stellen $\alpha_j \in (x_{j-1}, x_j)$ mit

$$f'(\alpha_j) = S_\Delta'(\alpha_j) \quad j = 0, \ldots, n+1.$$

Es gilt also für jedes $x \in [a,b]$ ein α_j mit $|\alpha_j - x| < \sigma_m$. Daher gilt für alle $x \in [a,b]$:

$$\begin{aligned} f'(x) - S'_\Delta(x) &= \int_{\alpha_j(x)}^{x} \left(f''(t) - S''_\Delta(t)\right) dt \\ \Longrightarrow \left|f'(x) - S'_\Delta(x)\right| &\leqslant \frac{7}{4} L\kappa\sigma_m^2 \cdot \sigma_m = \frac{7}{4} L\kappa\sigma_m^3. \end{aligned}$$

Analog folgt für $x \in [a,b]$ (Hier wieder $|x - x_j| \leqslant \frac{1}{2}\sigma_m$)

$$f(x) - S_\Delta(x) = \int_{x_j}^{x} f'(t) - S'_\Delta(t) dt$$

und damit

$$|f(x) - S_\Delta(x)| \leqslant \frac{7}{4} L\kappa\sigma_m^3 \frac{1}{2}\sigma_m = \frac{7}{8} L\kappa\sigma_m^4.$$

□

Es gäbe noch viel mehr zu Splines zu sagen.

4.4 Übungen zu Kapitel 4

Übung 4.76 *Seien $x_0, \ldots, x_n \in \mathbb{R}$ paarweise verschieden und $f_0, \ldots, f_n \in \mathbb{R}$ zugehörige Funktionswerte. Definiere die dividierten Differenzen $f[x_i, x_{i+1}, \ldots, x_{i+k}]$ rekursiv durch*

$$f[x_i] = f_i,\ i = 0, \ldots, n,$$

sowie

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] := \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i},$$

$i = 0, \ldots, n,\ k = 0, \ldots, n-i$. Nehme an, daß

$$f[x_k], f[x_{k-1}, x_k], f[x_{k-2}, x_{k-1}, x_k], \ldots, f[x_0, \ldots, x_k]$$

bereits vorhanden sind. Wie kann man daraus $f[x_0, \ldots, x_{k+1}]$ berechnen? Was folgt daraus insgesamt für die Berechnung von $f[x_n], f[x_{n-1}, x_n], f[x_{n-2}, x_{n-1}, x_n], \ldots, f[x_0, \ldots, x_n]$?

Übung 4.77 *Seien $x_0, \ldots, x_n \in \mathbb{R}$ paarweise verschieden und $f_0, \ldots, f_n \in \mathbb{R}$ zugehörige Funktionswerte und $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$ das eindeutige Polynom n–ten Grades mit $p(x_i) = f_i$, $i = 0, \ldots, n$.*

1. *Bestimme den Rechenaufwand (*flops*) zur Bestimmung von p für die Newtondarstellung von p (d.h. die dividierten Differenzen) sowie für die Berechnung der Koeffizienten von p durch Lösen des linearen Gleichungssystems $Va = f$. Dabei ist V die Matrix aus Aufgabe 3 und $a = (a_0, \ldots, a_n)^T$, $f = (f_0, \ldots, f_n)^T$.*
2. *Wie teuer ist die Auswertung von p in der Newtondarstellung bei gegebenen dividierten Differenzen, bzw. bei gegebenen Koeffizienten $a_0, \ldots, a_n$ verglichen mit der Lagrange–Darstellung von p?*
3. *Welche der drei Varianten ist die billigste, wenn man $p(x)$ ein einziges Mal auswerten will. Wie sieht es für eine große Anzahl von Werten x (Größenordnung n) aus?*

Übung 4.78 *Zeige, daß*

$$\frac{(b-x)^2}{(b-a)^2}\left[f_1\frac{2x+b-3a}{b-a}+s_1(x-a)\right]+\frac{(a-x)^2}{(a-b)^2}\left[f_2\frac{2x+a-3b}{a-b}+s_2(x-b)\right]$$

das eindeutige Polynom dritten Grades ist, welches für $a<b$ und vorgegebene Werte $f_1,s_1,f_2,s_2\in\mathbb{R}$ die Interpolationsaufgabe

$$p(a)=f_1,p'(a)=s_1,p(b)=f_2,p'(b)=s_2$$

erfüllt ('Hermite–Interpolierende').

Übung 4.79 *Seien $x_0,\ldots,x_n\in\mathbb{R}$ und*

$$V=\begin{pmatrix}1 & x_0 & x_0^2 & \cdots & x_0^n\\ 1 & x_1 & x_1^2 & \cdots & x_1^n\\ \vdots & \vdots & \vdots & & \vdots\\ 1 & x_n & x_n^2 & \cdots & x_n^n\end{pmatrix}.$$

Zeige, dass V (Vandermonde-Matrix) invertierbar ist genau dann, wenn $x_0,\ldots,x_n$ paarweise verschieden sind.

Übung 4.80 *Zu vorgegebenen Stützstellen $t_0<t_1<\cdots<t_n$ mit Funktionswerten $f_0,\ldots,f_n$ ist das Neville–Schema zur Berechnung des zugehörigen Interpolationspolynoms p an der Stelle x wie folgt definiert: $P_{i,0}:=f_i,\ n\geqslant i\geqslant 0$.*
Für $k=1,\ldots,n$ setze $P_{i,k}:=\dfrac{(x-t_{i-k})P_{i,k-1}-(x-t_i)P_{i-1,k-1}}{t_i-t_{i-k}},\ n\geqslant i\geqslant k$.
Dann ist $p(x)=P_{n,n}$.

a) *Lässt sich das Neville–Schema in einem ähnlichen Schema wie die dividierten Differenzen aufschreiben?*

b) *Wie teuer ist die Berechnung von $P_{n,n}$ im günstigsten Fall und ist dies günstiger oder teurer als die Newton–Darstellung mittels dividierter Differenzen?*

c) *Nehme an, dass $x=0$, $t_i=h^2 4^{-i}$, $i=0,\ldots,n$. Schreibe das Neville–Schema für diesen Fall um.*

Übung 4.81 *Schreibe ein* **MATLAB** *–Programm, welches für beliebige Vorgabe $t_0<t_1<\cdots<t_n$ von Stützstellen mit Funktionswerten $f_0,\ldots,f_n$ das zugehörige Interpolationspolynom p an der Stelle x mit Hilfe des Neville–Schemas auswertet, welches wie folgt definiert ist:*
$P_{i,0}:=f_i,\ n\geqslant i\geqslant 0$.
Für $k=1,\ldots,n$ setze $P_{i,k}:=\dfrac{(x-t_{i-k})P_{i,k-1}-(x-t_i)P_{i-1,k-1}}{t_i-t_{i-k}},\ n\geqslant i\geqslant k$.
Dann ist $p(x)=P_{n,n}$. Wende das Programm auf $f(t)=\log_{10}(t)-\frac{t-1}{t}, x=5.25$ und folgende Stützstellenverteilung an und interpretiere die Ergebnisse.

a) *1.0,2.0,4.0,8.0,10.0*

b) *2.0,4.0,8.0,10.0*

c) *4.0,8.0,10.0*

d) *2.0,4.0,8.0*

Übung 4.82 *Bestimme mit dem Newtonschema die Interpolationspolynome p zu den Funktionen $f(t)=\frac{1}{1+25t^2}$ und $g(t)=\sqrt{|t|}$. Als Stützstellen verwende jeweils*

i) $t_i = -1 + ih,\ i = 0, \ldots, n; \quad h = \dfrac{2}{n},$

ii) $t_i = \cos \dfrac{(2i+1)\pi}{2(n+1)},\ i = 0, \ldots, n,$

für $n = 2, 4, 6, \ldots, 20$. *Werte das Polynom an den Stellen* $y_j^i = t_i + j\frac{t_{i+1}-t_i}{21},\ j = 1, \ldots, 20$ *aus. Vergleiche als Schätzung für den maximalen Fehler* $\|p - f\|_{\infty\,[-1,1]}$

$$\max_{i\in\{0,\ldots,n-1\},j\in\{1,\ldots,20\}} |p(y_j^i) - f(y_j^i)|.$$

Plotte die Interpolationspolynome zu den Stützstellen in i), ii) für $n = 20$. *Erläutere die Ergebnisse.*

Übung 4.83 *Gegeben sei die Funktion* $f : \mathbb{R} \mapsto \mathbb{R}$ *durch*

$$f|_{[0,2\pi)}(t) := \begin{cases} 1 & \text{für } 0 \leqslant t < \pi \\ -1 & \text{für } \pi \leqslant t < 2\pi \end{cases}$$

und $f(t + 2k\pi) = f(t)$ *für alle* $k \in \mathbb{Z}$. *(Rechteckimpuls erster Art)*

i) Bestimme die zu f *gehörige Fourierreihe*

$$\frac{1}{2}a_0 + \sum_{k=1}^{\infty}(a_k \cos kx + b_k \sin kx), \quad a_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos kx\, dx, b_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin kx\, dx.$$

ii) Zeichne (oder Plotte) die ersten fünf Partialsummen und vergleiche mit f.

Übung 4.84 *Seien* $x_0, x_1, x_2 \in \mathbb{R}$ *mit* $x_0 \neq x_2$. *Zeige:*

1. *es gibt eindeutige Polynome* p_1, p_2, p_3, p_4 *dritten Grades , welche den Bedingungen*

$$\begin{array}{llll} p_1(x_0) = 1 & p_2(x_0) = 0 & p_3(x_0) = 0 & p_4(x_0) = 0 \\ p_1(x_2) = 0 & p_2(x_2) = 1 & p_3(x_2) = 0 & p_4(x_2) = 0 \\ p_1'(x_1) = 0 & p_2'(x_1) = 0 & p_3'(x_1) = 1 & p_4'(x_1) = 0 \\ p_1''(x_1) = 0 & p_2''(x_1) = 0 & p_3''(x_1) = 0 & p_4''(x_1) = 1 \end{array}$$

   genügen.

2. *Es existiert ein eindeutiges Polynom* p *dritten Grades gibt, daß für die Werte* f_0, f_2, f_1', f_1'' *die Interpolationsaufgabe*

$$p(x_0) = f_0, \quad p(x_2) = f_2, \quad p'(x_1) = f_1', \quad p''(x_1) = f_1''$$

   erfüllt.

Übung 4.85 *Sei* $c \in \mathbb{C}^n, J \in \mathbb{C}^{n\times n}$ *definert durch*

$$c := \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}, \quad J := \begin{pmatrix} 0 & & & 1 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{pmatrix}$$

Definiere die Matrix $C \in \mathbb{C}^{n\times n}$ *durch*

$$C := \left(c,\ Jc,\ J^2c, \ldots, J^{n-1}c\right)$$

(C *bezeichnet man auch als Zirkulante).*
Sei $\omega_n := e^{-i\frac{2\pi}{n}}$ *und* $W \in \mathbb{C}^{n\times n}$ *definert als* $W := \dfrac{1}{\sqrt{n}}\left(\omega_n^{kl}\right)_{k,l=0,\ldots,n-1}$. *Zeige:*

i) $W^*W = I$, *d.h.* W *ist unitär.*

ii) W^*CW *ist eine Diagonalmatrix. (Tip: Zeige zuerst, daß* W^*JW *diagonal ist und nutze dann die Def. von* C*).*

iii) Was sind die Eigenwerte von C *und wie lassen sie sich berechnen?*

iv) Sei $n = 2^k$*. Zeige, daß sich das Gleichungssystem* $Cx = b$, $b \in \mathbb{C}^n$ *gegeben, durch drei FFT berechnen läßt. Bestimme die wesentliche Rechenzeit.*

Übung 4.86 *Gegeben die Werte*

$$\begin{array}{cccc} y_0 = 0.580 & y_1 = 0.951 & y_2 = 0.786 & y_3 = 0.298 \\ y_4 = 0.454 & y_5 = 0.006 & y_6 = 0.276 & y_7 = 0.306 \end{array}$$

Führe eine FFT durch, d.h. berechne

$$c_k = \sum_{m=0}^{n-1} \omega_n^{km} y_m, \quad \text{für } k = 0, \ldots, n-1$$

mittels FFT, wobei $\omega_n := e^{-i\frac{2\pi}{n}}$.

Übung 4.87 *Seien* $t_{-3} < t_{-2} < \cdots < t_{n+2} < t_{n+3}$ *und* $t_{i+1} - t_i = h$ *für alle* $i = -3, \ldots, n+2$*. Definiere für* $i = -1, \ldots, n+1$ *die Funktion*

$$B_i(t) = \frac{1}{h^3} \begin{cases} (t - t_{i-2})^3 & t \in [t_{i-2}, t_{i-1}) \\ h^3 + 3h^2(t - t_{i-1}) + 3h(t - t_{i-1})^2 - 3(t - t_{i-1})^3 & t \in [t_{i-1}, t_i) \\ h^3 + 3h^2(t_{i+1} - t) + 3h(t_{i+1} - t)^2 - 3(t_{i+1} - t)^3 & t \in [t_i, t_{i+1}) \\ (t_{i+2} - t)^3 & t \in [t_{i+1}, t_{i+2}) \\ 0 & \text{sonst} \end{cases}$$

Zeige:

1. $B_i(t)$ *ist zweimal stetig differenzierbar.*
2. *Skizziere* $B_i(t)$*.*
3. *Betrachte für* $p(t) = \sum_{i=-1}^{n+1} a_i B_i(t)$ *und Funktionswerte* $f_0, \ldots, f_n$ *die Interpolationsaufgabe* $p(t_i) = f_i$, $i = 0, \ldots, n$, $p''(t_0) = 0 = p''(t_n)$*. Lässt sich die Aufgabe eindeutig lösen und wenn ja, wie teuer ist die Berechnung von* $a_0, \ldots, a_n$*? Wie teuer ist die Auswertung von* $p(x)$ *bei gegebenen* $a_0, \ldots, a_n$*?*
4. *Was ändert sich bei der abgewandelten Aufgabe* $p(t_i) = f_i$, $i = 0, \ldots, n$, $p'(t_0) = f_0'$, $p'(t_n) = f_n'$ *bei zusätzlichen Funktionswerten* f_0', f_n'*?*

Übung 4.88 *Sei* $A \in \mathbb{R}^{n,n}$*.*

1. *Sei* $B \in \mathbb{R}^{n,n}$, $B = \begin{pmatrix} 0 & 0 & b_{1n} \\ 0 & 0 & 0 \\ b_{n1} & 0 & 0 \end{pmatrix}$ *Finde* $U, V \in \mathbb{R}^{n,2}$*, so dass* $B = UV^\top$*.*
2. *Zeige die Sherman–Morrison–Woodbury Formel, d.h. sind* A *und* $A + UV^\top$ *nicht singulär, dann ist auch* $C = I + V^\top A^{-1} U$ *nichtsingulär und es gilt:*

$$(A + UV^\top)^{-1} = A^{-1} - A^{-1} U C^{-1} V^\top A^{-1}.$$

3. *Betrachte das lineare Gleichungssystem* $(A + B)x = b$*, wobei* A *und* $A + B$ *nichtsingulär seien mit* B *aus Teil 1. Nehme an, dass* A *tridiagonal ist und* A *und* $A + B$ *eine LU–Zerlegung besitzen (ohne Pivotisierung). Wie teuer ist die Lösung von* $(A + B)x = b$ *wenn man einmal eine LU–Zerlegung von* $A + B$ *macht verglichen mit der Variante, bei der man zunächst eine LU–Zerlegung von* A *macht und dann die Sherman–Morrison–Woodbury Formel verwendet.*

Übung 4.89 *Bestimme die kubischen Splines S_Δ zu den Funktionen $f(t) = \frac{1}{1+25t^2}$ und $g(t) = \sqrt{|t|}$, wobei $S''_\Delta(-1) = S''_\Delta(1) = 0$ (natürliche Splineinterpolation). Als Stützstellen verwende $t_i = -1 + ih,\ i = 0, \ldots, n;\quad h = \dfrac{2}{n}$, für $n = 2, 4, 6, \ldots, 20$. Werte den Spline an den Stellen $y_j^i = t_i + j\frac{t_{i+1}-t_i}{21},\ j = 1, \ldots, 20$ aus. Vergleiche als Schätzung für den maximalen Fehler $\|S_\Delta - f\|_{\infty\,[-1,1]}$*

$$\max_{i\in\{0,\ldots,n-1\},j\in\{1,\ldots,20\}} |S_\Delta(y_j^i) - f(y_j^i)|.$$

Plotte die Splines zu den Stützstellen für $n = 20$. Erläutere die Ergebnisse.

Übung 4.90 *Seien $\alpha, \beta \in \mathbb{R}$ und die Funktion $s_{\alpha,\beta} : [-1, 2] \to \mathbb{R}$ definiert durch*

$$s_{\alpha,\beta}(x) = \begin{cases} (x+1)^4 + \alpha(x-1)^4 + 1 & x \in [-1, 0] \\ -x^3 - 8\alpha x + 1 & x \in (0, 1] \\ \beta x^3 + 8x^2 + \frac{11}{3} & x \in (1, 2] \end{cases}$$

Bestimme α, β so, dass $s_{\alpha,\beta}$ ein kubischer Spline bezüglich der Knotenverteilung $\Delta = \{-1, 0, 1, 2\}$ ist.

Kapitel 5

Numerische Integration

Aufgabe der numerischen Integration (Quadratur) ist die Berechnung von

$$\int_a^b f(x)dx, \quad a, b < \infty$$

Typischerweise ist f nicht direkt integrierbar.

Idee: Approximiere $f(x)$ durch Funktion, die einfach zu integrieren ist; Polynome, trigonometrische Polynome, Splines, rationale Funktionen, Exponentialsummen, ...

5.1 Newton–Cotes Formeln

Die Idee der Newton–Cotes Formeln ist $f(x)$ durch ein Polynom zu interpolieren und dieses zu integrieren. Bilde äquidistante Unterteilung von $[a, b] \quad x_i = a + ih \quad i = 0, \cdots, n$, mit $h = \frac{b-a}{n}$ $n > 0$ $n \in \mathbb{N}$ und bilde Interpolationspolynom in Π_n, das

$$P_n(x_i) = f(x_i) \quad i = 0, \cdots, n \tag{5.1}$$

erfüllt. Satz (4.2) $\Longrightarrow$

$$P_n(x) = \sum_{i=0}^{n} f_i L_i(x). \tag{5.2}$$

$\Longrightarrow$

$$\begin{aligned} \int_a^b P_n(x)dx &= \sum_{i=0}^{n} f_i \int_a^b L_i(x)dx \quad \text{mit } x = a + hs \\ &= h \sum_{i=0}^{n} f_i \int_0^n \Pi_{\substack{k=0 \\ k\neq i}}^{n} \frac{s-k}{i-k} ds \qquad (5.3) \\ &= h \sum_{i=0}^{n} f_i \alpha_i = \frac{b-a}{n} \sum_{i=0}^{n} \sigma_i f_i \quad \sigma_i \in \mathbb{Z}. \end{aligned}$$

Gewichte α_i sind unabhängig von f_i, a, b jedoch abhängig von n.
Die Gewichte liegen tabelliert vor. Diese Formeln heißen Newton–Cotes–Formeln.

Fehler:

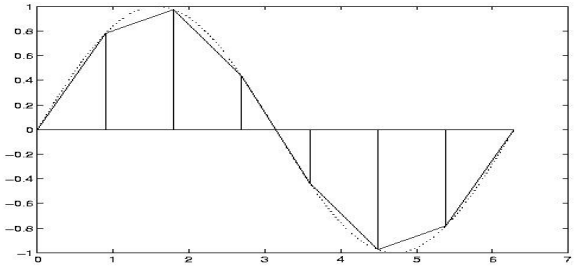
$$\int_a^b (P_n(x) - f(x))\,dx = h^{p+1}\kappa \cdot f^{(p)}(\xi), \xi \in (a,b). \tag{5.4}$$

Tabelle 5.5

n	σ_i	ns	*Fehler*	*Name*
1	1 1	2	$\frac{h^3}{12} f^{(2)}(\xi)$	*Trapezregel*
2	1 4 1	6	$\frac{h^5}{90} f^{(4)}(\xi)$	*Simpsonregel*
3	1 3 3 1	8	$\frac{h^5 3}{80} f^{(4)}(\xi)$	3/8 *Regel*
4	7 32 12 32 7	90	$\frac{h^7 178}{945} f^{(6)}(\xi)$	*Milne–Regel*
5	19 75 50 50 75 19	288	$\frac{h^7 275}{12096} f^{(6)}(\xi)$	–
6	41 216 27 272 27 216 41	840	$\frac{h^9 9}{1400} f^{(8)}(\xi)$	*Weddle–Regel*

größere $n \Longrightarrow$ negative Gewichte.
Wegen der schlechten Approximationseigenschaften (global) von Polynomen werden die Regeln stückweise auf Teilintervalle angewendet und dann aufsummiert.

Beispiel 5.6 (Summierte Trapezregel: stückweise lineare Interpolation)



Teilintervalle $[x_i, x_{i+1}], x_i = a + ih \quad i = 0, \cdots, N \quad h = \frac{b-a}{N}$. Dann ist die summierte Trapezregel

$$\begin{aligned} T(h) &:= \sum_{i=0}^{N-1} \frac{h}{2} \left[f(x_i) + f(x_{i+1})\right] \\ &= h\left[\frac{f(a)}{2} + f(a+h) + \cdots + f(b-h) + \frac{f(b)}{2}\right] \end{aligned} \tag{5.7}$$

Für jedes Teilintervall ist der Fehler

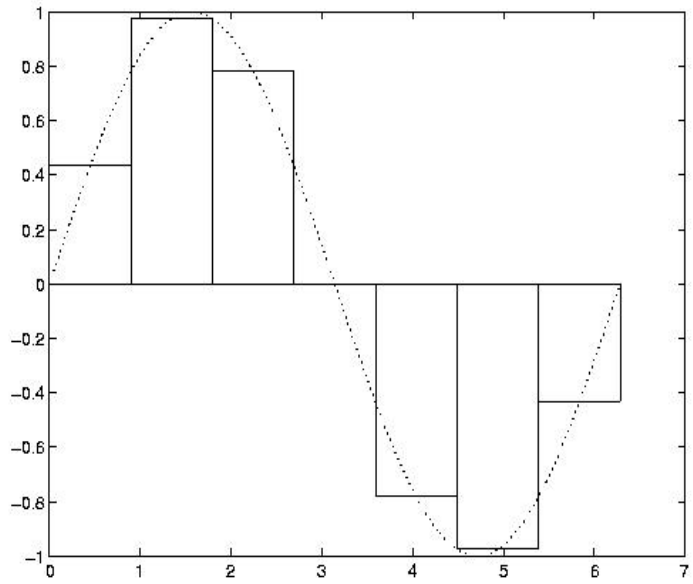
$$\frac{h}{2}\left[f(x_i) - f(x_{i+1})\right] - \int_{x_i}^{x_{i+1}} f(x)dx = \frac{h^3}{12} f^{(2)}(\xi_i) \text{ für } \xi_i \in (x_i, x_{i+1}). \tag{5.8}$$

Insgesamt ergibt sich also

$$\begin{aligned} \left| T(h) - \int_a^b f(x)dx \right| &= \left| \sum_{i=0}^{n-1} \frac{h^3}{12} f^{(2)}(\xi_i) \right| \\ &\leqslant \frac{h^3}{12} N \cdot \underbrace{\sup_{\xi\in[a,b]} f''(\xi)}_{=M} = \frac{b-a}{12} h^2 M. \end{aligned} \tag{5.9}$$

Fehler geht mit h^2 gegen 0, Verfahren 2. Ordnung.

Beispiel 5.10 (Mittelpunktregel)



Beispiel 5.11 *Sei N gerade und verwende Simpsonregel auf $[x_{2i}, x_{2i+1}, x_{2i+2}]$, $i = 0, \cdots, \frac{N}{2} - 1$. Der Näherungswert auf jedem der Teilintervalle ist*

$$\frac{h}{3}\left[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right].$$

Summation ergibt

$$\begin{aligned} S(h) &= \frac{h}{3}\left[(f(a) + 4f(a+b) + 2f(a+2h) + 4f(a+3h) + \right. \\ &\quad \left. + 2f(b-2h) + 4f(b-h) + f(b)\right] \end{aligned} \tag{5.12}$$

und als Fehler

$$\left| S(h) - \int_a^b f(x)dx \right| \leqslant \frac{h^5}{90} \cdot \frac{N}{2} \cdot \underbrace{\sup_{\xi\in[a,b]} f^{(4)}(\xi)}_{=M} \tag{5.13}$$

$$= \frac{b-a}{180} h^4 \cdot M.$$

Verfahren 4. Ordnung.

Weitere Quadraturformeln erhält man durch andere Approximationsaufgaben für $f(x)$.
Zum Beispiel Gauss–Quadratur für Integrale der Form

$$\int_a^b w(x) f(x) dx \approx \sum_{i=1}^{n} w_i f(x_i).$$

wobei w_i, f_i so gewählt werden, daß der Fehler

$$\int_a^b w(x) f(x) dx - \sum_{i=1}^{n} w_i f(x).$$

für Polynome möglichst hohen Grades verschwindet $\longrightarrow$ Übung.
Andere Ideen später bei der Lösung von Dgl.

5.2 Extrapolation

Untersuche das asymptotische Verhalten der Trapezsumme $T(h)$ für $h \to 0$. Dazu brauchen wir zuerst einige Eigenschaften der Bernoulli–Polynome, die wie folgt rekursiv definiert sind.

Definition 5.14 *Die Polynome $B_k(x), k = 0, 1, 2, \cdots$, die durch die rekursive Folge*

$$\begin{aligned} &B_0(x) = 1, B_k'(x) = k \cdot B_{k-1}(x) \quad k \geqslant 1, \\ &\int_0^1 B_k(x)\, dx = 0, \ k \geqslant 1 \end{aligned} \tag{5.15}$$

definiert sind, heißen Bernoulli–Polynome.
Eigenschaften der Bernoulli–Polynome:

$$B_k(x) = a_k + k \int_0^x B_{k-1}(t)\, dt \quad k \geqslant 1 \tag{5.16}$$

wobei a_k so zu bestimmen ist, daß $\int_0^1 B_k(x)\, dx = 0$. Man erhält sofort

$$\begin{aligned} &B_1(x) = x - \tfrac{1}{2}, B_2 = x^2 - x + \tfrac{1}{6}, B_3(x) = x^3 - \tfrac{3}{2}x^2 + \tfrac{1}{2}x \\ &B_4(x) = x^4 - 2x^3 + x^2 - \tfrac{1}{30}, \cdots \end{aligned} \tag{5.17}$$

und es gilt das folgende Lemma.

Lemma 5.18

i) $B_k(0) = B_k(1)$ *für* $k \geqslant 2$.

ii) Die Polynome $P_k(x) = B_k(\frac{1}{2} + x)$ sind für gerades k gerade, d.h. $P(x) = P(-x)$ und für ungerades k ungerade Polynome, d.h. $P(x) = -P(-x)$ in x.

iii) $B_{2k+1}(0) = 0$ für alle $k \geqslant 1$.

Beweis:

i) Aus (5.16)

$$B_k(1) - B_k(0) = k \int_0^1 B_{k-1}(t)\, dt = 0 \text{ für } k \geqslant 2.$$

ii) $P_0(x) \equiv 1$ ist gerade. Sei $P_{2k}(x)$ für $k \geqslant 0$ gerades Polynom. Zeige, dass $P_{2k+2}(x)$ wieder gerade ist. Es gilt

$$P'_{2k+1}(x) = B'_{2k+1}(\frac{1}{2} + x) = (2k+1)B_{2k}(\frac{1}{2} + x) = (2k+1)P_{2k}(x)$$

Also

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} P'_{2k+1}(x)dx = 0.$$

Aus $P_{2k}(x)$ gerade folgt, daß

$$q(x) := (2k+1) \int_0^x P_{2k}(t)\, dt$$

ein ungerades Polynom ist und es gilt

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} q(x)dx = -\int_{+\frac{1}{2}}^{-\frac{1}{2}} q(-x)\, dx = \int_{\frac{1}{2}}^{-\frac{1}{2}} q(x)dx = -\int_{-\frac{1}{2}}^{\frac{1}{2}} q(x)dx = 0.$$

Weiterhin ist $q'(x) = (2k+1)P_{2k}(x)$ und $\int_{-\frac{1}{2}}^{\frac{1}{2}} q(x)dx = 0 \Longrightarrow q(x) = P_{2k+1}(x)$ ungerade.

$P_{2k+2}(x) = (2k+2) \int_0^x P_{2k+1}(t)\, dt + a \Longrightarrow P_{2k+2}$ ist wieder gerade. $\Longrightarrow$ ii).

iii) Für $k \geqslant 1$ gilt

$$B_{2k+1}(1) = B_{2k+1}(0) = P_{2k+1}(-\tfrac{1}{2}) = -P_{2k+1}(\tfrac{1}{2}) = -B_{2k+1}(1)$$
$$\Longrightarrow B_{2k+1}(0) = B_{2k+1}(1) = 0.$$

□

$\beta_k = (-1)^{k+1}B_{2k}(0)$ heißen Bernoullische Zahlen.
$\beta_1 = \frac{1}{6}, \beta_2 = \frac{1}{30}, \beta_3 = \frac{1}{42}, \beta_4 = \frac{1}{30}$.
Wir definieren nun neue Funktionen

$$S_k(x) := B_k(x - i) \text{ für } x \in E_i := \{x | i < x < i + 1\},\ i \leqslant k. \tag{5.19}$$

Diese Funktionen sind periodisch mit Periode 1 und durch die folgende Rekursion bestimmt.

$$S_k(x) = S_k(0) + k \int_0^x S_{k-1}(t)\,dt,\ k \geqslant 2 \tag{5.20}$$

S_1 ist stückweise linear mit Sprungstellen bei $i \in \mathbb{N}$ und Nullstellen bei $i + \frac{1}{2}$ und es gilt wegen Lemma 5.18

$$\begin{aligned} &B_k(0) = S_k(0) = S_k(1) = S_k(2) = \cdots \\ &S_{2k+1}(0) = 0, S_{2k}(0) = (-1)^{k+1}\beta_k. \end{aligned} \tag{5.21}$$

Damit können wir nun die Euler–MacLaurin'sche Summenformel beweisen.

Satz 5.22 *Für $f \in \mathcal{C}^{2m+2}[a,b]$ hat $T(h)$ die Entwicklung*

$$T(h) = \tau_0 + \tau_1 h^2 + \tau_2 h^4 + \cdots + \tau_m h^{2m} + \alpha_{m+1}(h)h^{2m+2} \tag{5.23}$$

mit $\tau_0 := \int_a^b f(x)\,dx$. Dabei sind die τ_i von h unabhängige Konstanten und $|\alpha_{m+1}(h)| \leqslant M$ für alle $h = \frac{b-a}{n}, n \in \mathbb{N}$.

Beweis: Betrachte $g(t) := f(a + th)$. Damit können wir nun das Integral

$$\int_0^n S_1(t)g'(t)\,dt$$

durch partielle Integration umformen. $S_1(x)$ stückweise linear aus $B_1(x) = x - \frac{1}{2}$.

$$\int_0^1 S_1(t)g'(t)\,dt = \int_0^1 B_1(t)g'(t)\,dt = B_1(t)g(t)|_0^1 - \int_0^1 B_0(t)g(t)\,dt = \frac{1}{2}\left[g(1) + g(0)\right] - \int_0^1 g(t)\,dt.$$

Analog

$$\begin{aligned} \int_i^{i+1} S_1(t)g'(t)\,dt &= \int_0^1 B_1(t)g'(t+i)\,dt \\ &= \frac{1}{2}\left[g(i+1) + g(i)\right] - \int_i^{i+1} g(t)\,dt,\ i = 0, \ldots, n-1 \end{aligned}$$

$$\Longrightarrow \frac{g(0)}{2} + g(1) + \cdots + g(n-1) + \frac{g(n)}{2} - \int_0^n g(t)\,dt = \int_0^n S_1(t)g'(t)\,dt \tag{5.24}$$

Für die rechte Seite gilt

$$\begin{aligned} \int_0^n S_1(t)g'(t)\,dt &= \frac{1}{2!}\left(S_2(t)g'(t)|_0^n - \int_0^n S_2(t)g''(t)\,dt\right) \\ &= \frac{\beta_1}{2!}\left[g'(n) - g'(0)\right] - \frac{1}{2!}\int_0^n S_2(t)g''(t)\,dt \end{aligned}$$

$$
\begin{aligned}
&= \frac{\beta_1}{2!}\left[g'(n)-g'(0)\right] - \frac{1}{3!}S_3(t)g''(t)|_0^n + \frac{1}{3!}\int_0^n S_3(t)g^{(3)}(t)\,dt \\
&= \frac{\beta_1}{2!}\left[g'(n)-g'(0)\right] + \frac{1}{3!}\int_0^n S_3(t)g^{(3)}\,dt
\end{aligned}
$$

$2m$ mal partielle Integration ergibt

$$
\begin{aligned}
&\tfrac{g(0)}{2} + g(1) + \cdots + g(n-1) + \tfrac{g(n)}{2} - \int_0^n g(t)\,dt \\
&= \sum_{k=1}^{m}(-1)^{k+1}\tfrac{\beta_k}{(2k!)}\left[g^{(2k-1)}(n) - g^{(2k-1)}(0)\right] + R_{m+1}
\end{aligned}
\tag{5.25}
$$

mit

$$
\begin{aligned}
R_{m+1} &= \frac{1}{(2m+1)!}\int_0^n S_{2m+1}(t)g^{(2m+1)}(t)\,dt \\
&= \frac{1}{(2m+2)!}\left(S_{2m+2}(t) - S_{2m+2}(0)\right)g^{(2m+1)}(t)\Big|_0^n \\
&\quad -\frac{1}{(2m+2)!}\int_0^n \left[S_{2m+2}(t) - S_{2m+2}(0)\right]g^{(2m+2)}(t)\,dt \\
&= -\frac{1}{(2m+2)!}\int_0^n \left[S_{2m+2}(t) - S_{2m+2}(0)\right]g^{(2m+2)}(t)\,dt
\end{aligned}
\tag{5.26}
$$

Es ist $\int_0^n g(t)\,dt = \frac{1}{h}\int_a^b f(x)dx$ und $g^{(k)}(t) = h^k f^{(k)}(a+th) \quad k = 0,1,2,\cdots$

$$
\begin{aligned}
\Longrightarrow \frac{g(0)}{2} + g(1) + \cdots + g(n-1) + \frac{g(n)}{2} &= \frac{f(a)}{2} + f(a+h) + \cdots + f(b-h) + \frac{f(b)}{2} \\
&= \frac{1}{h}T(h).
\end{aligned}
$$

Also folgt

$$
T(h) = \tau_0 + \tau_1 h^2 + \cdots + \tau_m h^{2m} + \alpha_{m+1}(h)h^{2m+2} \tag{5.27}
$$

mit $\tau_0 = \int_a^b f(x)dx$.

$$
\tau_k = \frac{(-1)^{k+1}}{(2k)!}\beta_k\left[f^{(2k-1)}(b) - f^{(2k-1)}(a)\right] \tag{5.28}
$$

$$
\alpha_{m+1} = \frac{-1}{(2m+2)!}\int_a^b f^{(2m+2)}(x)\left[S_{2m+2}\left(\frac{x-a}{n}\right) - S_{2m+2}(0)\right]dx \tag{5.29}
$$

Da S_{2m+2} stetig und periodisch, folgt, daß es eine von h unabhängige Schranke für $|\alpha_{m+1}^{(n)}|$ gibt. □

Die Euler–MacLaurinformel läßt sich nun zur Extrapolation verwenden (Romberg/Bulisch).

Wenn man das Restglied vernachlässigt, ist $T(h)$ ein Polynom in h^2, das für $h = 0$ den Wert $\tau_0 = \int_a^b f(x)\,dx$ liefert.

Idee: Bestimme für verschiedene Schrittweiten
$h_0 = b - a, h_1 = \frac{h_0}{n_1}, \cdots, h_m = \frac{h_0}{n_m},\ n_i \in \mathbb{N}$, bestimme die zugehörige Trapezsumme

$$T_{i,0} = T(h_i),\ i = 0, \cdots, m \tag{5.30}$$

und dann durch Interpolation das Polynom

$$\tilde{T}_{m,m}(h) = a_0 + a_1 h^2 + \cdots + a_m h^{2m}, \tag{5.31}$$

daß die Interpolationsbedingungen

$$\tilde{T}_{m,m}(h_i) = T(h_i),\ i = 0, \ldots, m \tag{5.32}$$

erfüllt. Werte dann dies Polynom bei $h = 0$ mit dem Neville–Schema aus, dies liefert i.a. sehr gute Näherung für Integral.
Sei $\tilde{T}_{i,k}(h)$ das Polynom vom Grad k in h^2, daß

$$\tilde{T}_{i,k}(h_j) = T_{j,0},\ j = i - k, \cdots, i \tag{5.33}$$

erfüllt. Damit gilt für die extrapolieren Werte $T_{i,k} := \tilde{T}_{i,k}(0)$.

$$T_{i,k} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{h_{i-k}}{h_i}\right)^2 - 1},\ 1 \leqslant k \leqslant i \leqslant m \tag{5.34}$$

$\longrightarrow$ Übung 5.42 $\begin{cases} h_0 = b - a, h_1 = \frac{h_0}{2}, \cdots \Longrightarrow T_{11} \text{ Simpsonregel.} \\ h_0 = b - a, h_1 = \frac{h_0}{3}, \cdots \Longrightarrow T_{11}\ \frac{3}{8} - \text{ Regel.} \end{cases}$

Extrapolationsfolgen

a) $h_0 = b - a, h_1 = \frac{h_0}{2}, h_2 = \frac{h_1}{2}, h_3 = \frac{h_2}{2}, \cdots$ (Romberg Folge)

b) $h_0 = b - a, h_1 = \frac{h_0}{2}, h_2 = \frac{h_0}{3}, h_3 = \frac{h_1}{2}, h_4 = \frac{h_2}{2}, h_5 = \frac{h_3}{2}, \cdots$ (Bulirschfolge).

b) besser, da Rechenaufwand weniger schnell ansteigt. Vorherige Werte können verwendet werden. $\longrightarrow$ Übung 5.42.

Fehlerbetrachtung: Für ein $\xi \in [a, b]$

$$T_{mm} - \int_a^b f(x)\,dx = -\frac{1}{(2m+2)!} f^{(2m+2)}(\xi) \int_a^b K(x)\,dx \tag{5.35}$$

mit

$$K(x) = \sum_{i=0}^{m} c_{m_i} h_i^{2m+2} \left[S_{2m+2}\left(\frac{x-a}{h_i}\right) - S_{2m+2}(0) \right] \tag{5.36}$$

$$\Longrightarrow T_{mm} - \int_a^b f(x)dx = (b-a)h_0^2 \cdots h_m^2 \frac{\beta_{m+1}}{(2m+1)!} f^{(2m+2)}(\xi) \tag{5.37}$$

Details siehe z.B. Stoer, *Numerische Mathematik*.

5.3 Übungen zu Kapitel 5

Übung 5.38 *Bestimme für eine auf dem Intervall $[a,b]$ stetig differenzierbare Funktion f eine Quadraturformel für $\int_a^b f(t)\,dt$, indem Du Stützstellen $a = x_0 < x_1 < \cdots < x_n = b$ einführst und auf dem Intervall $[x_i, x_{i+1}]$ die Funktion f durch die (stückweise) Hermite–Interpolierende aus Übung 12, Nr.2 ersetzt (neben $f(x_0), \ldots, f(x_n)$ sollen auch $f'(x_0), \ldots, f'(x_n)$ gegeben sein).*

Übung 5.39 *Sei $f : [a,b] \longrightarrow \mathbb{R}$ eine zweimal stetig differenzierbare Funktion.*

a) *Sei F Stammfunktion zu f, d.h. $F' = f$. Seien $x, h \in \mathbb{R}$ mit $[x-h, x+h] \subseteq [a,b]$. Zeige:*

$$F(x+\frac{h}{2}) - F(x-\frac{h}{2}) = hf(x) + \frac{h^3}{24}\,\frac{f''(\theta_+) + f''(\theta_-)}{2}$$

für geeignet zu wählende $\theta_+, \theta_- \in [a,b]$.

b) *Zeige: Sei $l \in \mathbb{N} \setminus \{0\}$, $\theta_1, \ldots, \theta_l \in [a,b]$. Dann existiert ein $\theta \in [a,b]$, so daß*

$$f''(\theta) = \frac{1}{l}\sum_{k=1}^{l} f''(\theta_k)$$

ist.

c) *Sei $z_i = a + ih,\ i = 0, \ldots, N, \quad h = \frac{b-a}{N}$. Zeige:*

$$\int_{z_i}^{z_{i+1}} f(t)\ dt - hf(\frac{z_i + z_{i+1}}{2}) = \frac{h^3}{24} f''(\theta_i)$$

für ein geeignetes $\theta_i \in [a,b], i = 0, \ldots, N-1$.

d) *Sei $M(h) = h\sum_{i=0}^{N-1} f(\frac{z_i+z_{i+1}}{2})$. (Summierte Mittelpunktregel) Zeige:*

$$\int_a^b f(t)\ dt - M(h) = (b-a)\frac{h^2}{24} f''(\theta),$$

wobei $\theta \in [a,b]$ geeignet zu wählen ist.

Übung 5.40 *Bestimme für eine auf dem Intervall $[a,b]$ stetig differenzierbare Funktion f eine Quadraturformel für $\int_a^b f(t)\,dt$, indem Du Stützstellen $a = x_0 < x_1 < \cdots < x_n = b$ einführst und f durch den kubischen Spline S ersetzt (neben $f(x_0), \ldots, f(x_n)$ sollen auch $f'(x_0)$ und $f'(x_n)$ gegeben sein). Tipp: Interpretiere S als Hermite–Interpolierende, bei der $s_i = S'(x_i), 0 < i < n$ als Unbekannte auftritt.*

Übung 5.41 *Verwende zur Berechnung der Integrale*

$$\int_0^{\frac{\pi}{2}} \sin x\ dx,\ \int_0^1 \sqrt{x}\ dx,\ \int_0^1 x\sqrt{x}\ dx,\ \int_0^1 x^2\sqrt{x}\ dx$$

a) *die summierte Trapezregel für $N = 1, 2, 4, 8, 16, \ldots, 2^9$ und folgende Unterteilung:*

$$x_i = i \cdot h,\ i = 0, 1, \ldots, N, \quad h = \frac{\pi}{2N}.$$

b) *die summierte Mittelpunktregel für $N = 1, 2, 4, 8, 16, \ldots, 2^9$ und der Unterteilung aus a).*

c) *die summierte Simpsonregel für $N = 2, 4, 8, 16, \ldots, 2^9$ und der Unterteilung aus a).*

Vergleiche die Verfahren bezüglich Genauigkeit und Rechenaufwand.

Übung 5.42 *Verwende zur Berechnung des Integrals aus Aufgabe 5.41 das Extrapolationsverfahren zur summierten Trapezregel für $k = 0, 1, 2, \ldots, 9$ mit der Rombergfolge*

$$h_l = \frac{\pi}{2^{l+1}},\ l = 0, \ldots, k.$$

Untersuche das Verfahren bezüglich Genauigkeit und Rechenzeit.

Übung 5.43 *Zu einer gegebenen stetigen Funktion $p : [-1, 1] \to \mathbb{R}$ betrachte folgende Quadraturformel*

$$\int_{-1}^{1} p(x)\ dx \approx p(-\frac{1}{\sqrt{3}}) + p(\frac{1}{\sqrt{3}}).$$

Zeige: Die Formel ist für Polynome p vom Grade höchstens 3 exakt.
Tipp: Zerlege p als $p(x) = L(x)q(x) + r(x)$, wobei $L(x)$ ein quadratisches Polynom ist, welches $\pm\frac{1}{\sqrt{3}}$ als Nullstelle hat und r ein höchstens lineares Polynom ist.

Übung 5.44 *Gegeben sei eine Quadraturformel $\int_{-1}^{1} f(x)\,dx = Q(f) + R_Q(f)$, wobei $Q(f) = \sum_{i=1}^{s} \omega_i f(\xi_i)$, $-1 \leqslant \xi_1 < \cdots < \xi_s \leqslant 1$ und $\omega_i \geqslant 0$, $\sum_{i=1}^{s} \omega_i = 2$. Sei ausserdem eine stetige $F : [-1, 1]^2 \to \mathbb{R}$ gegeben.*

1. *Konstruiere mit Hilfe der Quadraturformel Q eine Kubaturformel K, d.h. eine Vorschrift zur Berechnung von*
$$\int_{-1}^{1}\int_{-1}^{1} F(x, y)\,dx\,dy = K(F) + R_K(F),$$
wobei $R_K(F)$ zugehöriger Approximationsfehler.
2. *Zeige, dass für den Fehler $R_K(F)$ der Kubaturformel die folgende Abschätzung gilt.*
$$R_K(F) \leqslant 2\left(\max_{x\in[-1,1]} |R_Q(F(x, \bullet))| + \max_{y\in[-1,1]} |R_Q(F(\bullet, y))|\right).$$

Kapitel 6

Eigenwertprobleme

6.1 Grundlagen (Wiederholung)

Eigenwerte einer Matrix $A = [a_{ij}]_{i,j=1,\dots,n} \in \mathbb{C}^{n,n}$ sind die n Nullstellen $\lambda_1, \cdots, \lambda_n$ des charakteristischen Polynoms. $p(z) = \det(zI - A)$.
$\sigma(A) = \{\lambda_1, \cdots, \lambda_n\} =$ Spektrum.
$\det(A) = \lambda_1 \cdots \lambda_n$, Spur $(A) = \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} a_{i,i}$.
Sei $\lambda \in \sigma(A)$, dann heißen Vektoren $x \in \mathbb{C}^n \setminus \{0\}$ die $Ax = \lambda x$ erfüllen, Eigenvektoren.
Ein Unterraum $S \in \mathbb{C}^n$ mit der Eigenschaft $x \in S \Longrightarrow Ax \in S$ heißt invariant. Falls $AX = XB \quad B \in C^{k,k}, \ X \in \mathbb{C}^{n,k}$, rank $X = k$, so spannen die Spalten von X einen k dimensionalen invarianten Unterraum auf und $By = \lambda y \Longrightarrow AXy = \lambda Xy$.

Falls X vollen Spaltenrang hat gilt daher $\sigma(B) \subset \sigma(A)$. Falls X quadratisch, so folgt $\sigma(B) = \sigma(A)$, und $A, B = X^{-1}AX$ heißen ähnlich. X ist Ähnlichkeitstransformation

Lemma 6.1 *Sei $A \in \mathbb{C}^{n,n} B \in \mathbb{C}^{k,k}, X \in \mathbb{C}^{n,k} AX = XB$ und rang $(X) = k$, dann gibt es unitäre Matrix $Q \in \mathbb{C}^{n,n}$, so daß*

$$Q^*AQ = T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{matrix} k \\ n-k \end{matrix} \tag{6.2}$$

und $\sigma(T_{11}) = \sigma(A) \cap \sigma(B)$.

Beweis: Sei $X = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$ die QR–Zerlegung von X. Es folgt:

$$AQ \begin{bmatrix} R \\ 0 \end{bmatrix} = Q \begin{bmatrix} R \\ 0 \end{bmatrix} B \Longleftrightarrow (Q^*AQ) \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix} B =: \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} \begin{matrix} p \\ n-p \end{matrix}.$$

R ist nichtsingulär da Rang $(X) = k$. Aus $T_{21}R = 0 \cdot B = 0$ folgt $T_{21} = 0$ und aus $T_{11}R = RB$ folgt $\sigma(T_{11}) = \sigma(B)$. $\sigma(A) = \sigma(T) = \sigma(T_{11}) \cup \sigma(T_{22}) \Longrightarrow$ Beh. □

Damit erhält man den folgenden Satz

Satz 6.3 (Satz von Schur (1909)) *Sei $A \in \mathbb{C}^{n,n}$. Dann gibt es $Q \in \mathbb{C}^{n,n}$ unitär so daß*

$$Q^*AQ = T = D + N, \tag{6.4}$$

mit $D = diag\,(\lambda_1, \cdots, \lambda_n), N$ strikte obere Δ–matrix. Q kann so gewählt werden, daß Eigenwerte in beliebiger Folge auf der Diagonale stehen.

Beweis: Mit Induktion $n = 1$ trivial. Induktionsvor.: Sei Beh. richtig für $n-1$. Sei $Ax = \lambda x$, für $x \neq 0$. Mit Lemma 6.1 $B = \lambda$ gibt es unitäres U so daß

$$U^*AU = \begin{bmatrix} \lambda & w^* \\ 0 & c \end{bmatrix} \begin{matrix} 1 \\ n-1 \end{matrix}$$

Mit Induktion gibt es $\tilde{U}$ so daß $\tilde{U}^*C\tilde{U}$ obere $\triangle$–Matrix.
Sei $Q = U \begin{bmatrix} T & \tilde{w}^* \\ 0 & \tilde{U} \end{bmatrix}$ dann ist Q^*AQ obere $\triangle$–Matrix. □

Sei $Q = [q_1, \cdots, q_n], q_i$ heißen Schur–Vektoren, und spannen $S_k = \text{span}\,\{q_1, \cdots, q_k\}$ invariant auf. Eine Matrix heißt normal falls $A^*A = AA^*$.

Korollar 6.5 *$A \in \mathbb{C}^{n,n}$ ist normal $\iff \exists$ unitäres $Q \in \mathbb{C}^{n,n}$, so daß $Q^*AQ = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$.*

Beweis: A normal $\iff Q^*AQ$ normal $\forall Q$ unitär, denn

$$(Q^*AQ)^*(Q^*AQ) = Q^*A^*AQ = Q^*AA^*Q = (Q^*AQ)(Q^*AQ)^*.$$

$Q^*AQ = T$ obere $\triangle$–matrix, normal. $\iff T$ diagonal. □

Es gibt eine reelle Variante des Satzes von Schur:

Satz 6.6 *Sei $A \in \mathbb{R}^{n,n}$. Es gibt orthogonale Matrix $Q \in \mathbb{R}^{n,n}$ so daß*

$$Q^TAQ = \begin{bmatrix} R_{11} & \cdots & R_{1m} \\ & \ddots & \vdots \\ & & R_{mm} \end{bmatrix}$$

wobei jedes R_{ii} entweder 1×1 oder 2×2 mit komplex–konjugierten Eigenwerten ist.

Beweis $\Longrightarrow$ Übung.

Satz 6.7 (Jordan'sche Normalform) *Sei $A \in \mathbb{C}^{n,n}$. Dann gibt es $X \in C^{n,n}$ so daß $X^{-1}AX = diag\,(J_1, \cdots, J_t)$, wobei*

$$J_i = \begin{bmatrix} \lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix} \in \mathbb{C}^{m_i, m_i}$$

und $m_1 + \cdots + m_t = n$.

Beweis: bekannt ekelhaft. □

Die Summe der Dimensionen m_i zu einem Eigenwert λ heißt algebraische Vielfachheit, falls diese 1 ist, so heißt λ einfacher Eigenwert. Die Anzahl der Jordanblöcke zu λ heißt geometrische Vielfachheit. Es gilt immer algebraische Vielfachheit $\geqslant$ geometrische Vielfachheit. Falls algebraische Vielfachheit $>$ geometrische Vielfachheit, so heißt λ defektiver Eigenwert und A defektiv.

Korollar 6.8 *$A \in \mathbb{C}^{m,n}$ ist nicht defektiv genau, wenn es $X \in \mathbb{C}^{n,n}$ nicht singulär gibt, so daß*

$$X^{-1}AX = \ diag\,(\lambda_1, \ldots, \lambda_n). \tag{6.9}$$

Beweis: Klar am Satz 6.7. Jordanform numerisch schwer zu berechnen. Kleine Störungen in der Matrix ändern Jordanform stark, numerisch ist jede Matrix diagonalisierbar, diagonalisierbare Matrizen liegen dicht in der Menge der Matrizen. Matrizen die fast defektiv sind, können schlecht konditionierte Eigenvektormatrizen haben. Und dies kann drastische Folgen haben, falls man nichtunitäre Transformationen zur Eigenwertberechnung verwendet. Denn es gilt

$$fl(X^{-1}AX) = X^{-1}AX + E \tag{6.10}$$

wobei

$$\|E\|_2 \cong \ \text{eps}\ \kappa_2(X)\|A\|_2 \tag{6.11}$$

Falls X nicht unitär ist , kann E sehr groß werden.

Beispiel 6.12

$$\begin{aligned} A &= \begin{bmatrix} 5 & -1 \\ 2 & 2 \end{bmatrix} && \sigma(A) = \{3, 4\}, \\ Q &= \begin{bmatrix} 0.6 & 0.8 \\ -0.8 & 0.6 \end{bmatrix} && \kappa_2(Q) = 1, \\ X &= \begin{bmatrix} 1.0 & 1.01 \\ 2. & 2.00 \end{bmatrix} && \kappa_2(X) = 167. \end{aligned}$$

Sei $\beta = 10, t = 3$ dann ergibt sich mit Rundung

$$\begin{aligned} \|fl(X^{-1}AX) - X^{-1}AX\|_2 &\cong 1.9 \\ \|fl(Q^*AQ - Q^*AQ\|_2 &\cong 10^{-3}. \end{aligned}$$

Satz 6.13 (Satz von Gerschgorin) *Sei*

$$A = D + F \ \textit{mit}\ D = diag(d_1, \cdots, d_n)$$

und F habe Nulldiagonale, dann gilt

$$\sigma(A) \subseteq \bigcup_{i=1}^{n} D_i$$

wobei

$$D_i = \left\{ z \in \mathbb{C} : \ |z - d_i| \leqslant \sum_{j=1}^{n} |f_{ij}| \right\}.$$

Beweis: $\longrightarrow$ Übung 6.50. □

Satz 6.14 (Satz von Bauer–Fike.) *Sei μ ein Eigenwert von $A+E \in \mathbb{C}^{n,n}$ und $X^{-1}AX = D = diag\,(\lambda_1, \cdots, \lambda_n)$ diagonal, dann gilt*

$$\min_{\lambda \in \sigma(A)} |\lambda - \mu| \leqslant \kappa_p(X)\|E\|_p$$

Beweis: $\longrightarrow$ Übung 6.51 □

Der Satz von Bauer–Fike besagt, daß $\kappa(X)\cdot\|E\|$ große Störungen in den Eigenwerten auftreten können. X orthogonal $\Longrightarrow \kappa_2(X) = 1$. Resultate für invariante Unterräume im allgemeinen sehr schwer.

Satz 6.15 (Satz von Stewart) *Sei Q unitär*

$$Q^*AQ = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{matrix} p \\ n-p \end{matrix}$$

eine Schur–Zerlegung mit $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$. Sei $\operatorname{sep}(T_{11}, T_{22}) = \min\limits_{X \neq 0} \frac{\|T_{11}X - XT_{22}\|_F}{\|X\|_F}$ und sei $E \in \mathbb{C}^{n,n}$ eine Störung. Sei

$$Q^*EQ = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix} \begin{matrix} p \\ n-p \end{matrix} .$$

Falls $\delta = \operatorname{sep}(T_{11}, T_{22}) - \|E_{11}\|_2 - \|E_{22}\|_2 > 0$ und $\|E_{21}\|_2(\|T_{12}\|_2 + \|E_{21}\|_2) \leqslant \frac{\delta^2}{4}$, so gibt es $P \in \mathbb{C}^{n-k,k}$ mit $\|P\|_2 \leqslant 2\|E_{21}\|_2/\delta$ und die Spalten von $\hat{Q} = (Q_1 + Q_2P)(I + P^*P)^{-\frac{1}{2}}$ bilden eine Orthonormalbasis für invarianten Unterraum von $A + E$.

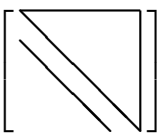
6.2 Der QR–Algorithmus für unsymmetrische Matrizen

Wir betrachten hier nur den reellen Fall, der Berechnung der reellen Schur–Form. Der beste Algorithmus zur Berechnung der Schur–Form ist der QR–Algorithmus von Francis, der im Prinzip die folgende Iterationsschrift ausführt.

$H_0 = Q_0^T A Q_0$
FOR $k = 1, 2, \cdots$
$\quad H_{k-1} = Q_k R_k \quad QR$–Zerlegung
$\quad H_k = R_k Q_k$
END

Diese Iteration würde pro Schritt $\mathcal{O}(n^3)$ flops brauchen (ungefähr $2n$ Iterationen) also $\mathcal{O}(n^4)$ insgesamt. Um das zu verbessern benutze Q_0, um A auf sogenannte Hessenbergform zu transformieren.

$$Q_0^T A Q_0 = H \equiv$$



Dies geschieht mit Hilfe von Householder–Transformation (HOUSE, COLHOUSE, ROWHOUSE)

$$A = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \end{bmatrix} \to P_1^T A P_1 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & x & x & x & x & x \end{bmatrix} \to$$

$$P_2^T P_1^T A P_1 P_2 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & x & x & x & x \end{bmatrix} \to$$

$$P_3^T P_2^R P_1^T A P_1 P_2 P_3 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x & x \end{bmatrix} \to P_4^T P_3^T P_2^T P_1^T A P_1 P_2 P_3 P_4 = ...$$

Algorithmus 6.16 (Householder–Reduktion auf Hessenberg–Form)

Input: $A \in \mathbb{R}^{n,n}$

Output: $H \in \mathbb{R}^{n,n} H = Q^T A Q, Q$ *Produkt von Householder–Transformation gespeichert auf unterem Teil von* H.

FOR $k = 1 : n-2$

$v(k+1:n) =$ *HOUSE* $(A(k+1:n,k))$;

$A(k+1:n,k:n) =$ *ROWHOUSE* $(A(k+1:n,k:n), v(k+1:n))$;

$A(1:n,k+1:n) =$ *COLHOUSE* $(A(1:n,k+1:n), v(k+1:n))$;

$A(k+2:n,k) = v(k+2:n)$;

END

Kosten: $20n^3/3$ flops $+\frac{4n^3}{3}$ flops falls Q gebraucht wird.

Fehleranalyse: $\tilde{H} = Q^T(A+E)Q$ mit $Q^TQ = I$ und $\|E\|_F \leqslant cn^2 \operatorname{eps} \|A\|_F$.

→ Übung 6.55.

Die Hessenberg–Reduktion ist nicht eindeutig. Falls Hessenberg–Matrix unreduziert ist, d.h. alle Elemente in der ersten unteren Nebendiagonalen $\neq 0$ sind, so ist die Hessenreduktion fast eindeutig durch die erste Spalte von Q bestimmt, d.h. wird diese festgelegt, so ist der Rest eindeutig bis auf Vorzeichenmuster.

Satz 6.17 (Implizites Q–Theorem) *Sei* $A \in \mathbb{R}^{n,n}$, *seien* $Q = [q_1, \ldots, q_n], V = [v_1, \cdots, v_n]$ *orthogonale Matrizen , so daß* $Q^TAQ = H, V^TAV = G$ *beides Hessenbergmatrizen. Sei* k *der kleinste Index so daß* $h_{k+1,k} = 0$ ($k = n$, *falls* H *unreduziert). Falls* $v_1 = q_1$ *so gilt* $v_i = \pm q_i$ *und* $|h_{i,i-1}| = |q_{i,i-1}|$ *für* $i = 2, k$. *Falls* $k < n$ *so gilt* $g_{k+1,k} = 0$.

Beweis: Sei $W = V^TQ$. Dann gilt $GW = WH$. Also folgt für $i = 2 : k$

$$h_{i,i-1} w_i = G w_{i-1} - \sum_{j=1}^{i-1} h_{j,i-1} w_j$$

Da $w_1 = e_1$ folgt, daß $[w_1, \cdots, w_l]$ obere $\triangle$–Matrix ist. $w_i = v_i^T q_i$ und $h_{i,i-1} = w_i^T G w_{i-1}$ implizieren
$v_i = \pm q_i, |h_{i,i-1}| = |g_{i,i-1}|$ für $i = 2 : k$. Falls $h_{k-1,k} = 0$ so folgt (bis auf Vorzeichen)

$$\begin{aligned} g_{k-1,k} &= e_{k+1}^T G e_k = e_{k+1}^T G W e_k = (e_{k+1}^T W)(H e_k) \\ &= e_{k+1}^T \sum_{i=1}^{k} h_{ik} W e_i = \sum_{i=1}^{k} h_{ik} e_{k+1}^T e_i = 0. \end{aligned}$$

□

$\Longrightarrow$ Erste Spalte festlegen ergibt im wesentlichen gleiche Hessenbergreduktion (bis auf Vorzeichen).
Damit kann man nun eine praktische Variante des QR–Algorithmus machen.

$H = Q_0^T A Q_0$ Hessenberg–Reduktion
FOR $k = 1, 2$
  $H = QR$ QR–Zerlegung
  $H = RQ$
END

H bleibt Hessenberg, denn da $H = QR$ und R obere $\triangle$–Matrix folgt, daß $Q = HR^{-1}$ obere Hessenberg–Matrix $\Longrightarrow$ das neue $H = RQ$ ebendfalls obere Hessenbergmatrix.

Im folgenden kann man annnehmen, daß $h_{i+1,i} \neq 0$ für die Hessenberg–Matrix. Sonst kann man das Problem zerlegen.
Falls

$$H = \left[\begin{array}{c|c} H_{11} & H_{12} \\ \hline 0 & H_{22} \end{array}\right] \begin{array}{c} p \\ n-p \end{array}$$

mit z. B.

$$|h_{p+1,p}| \leqslant c \operatorname{eps} (|h_{pp}| + |h_{p+1,p+1}|) \tag{6.18}$$

so spaltet man H in zwei Teilprobleme auf: H_{11}, H_{22}.

Beschleunigung durch Shifts

Satz 6.19 *Sei μ ein Eigenwert einer unreduzierten oberen Hessenbergmatrix H. Sei $H - \mu I = QR$ QR–Zerlegung und $\tilde{H} = RQ + \mu I$ so gilt $\tilde{h}_{n,n-1} = 0, \tilde{h}_{nn} = \mu$.*

Beweis: H unreduziert $\Longrightarrow$ $H - \mu I$ unreduziert $U^T(H - \mu I) = R$ ist singulär und da man zeigen kann, daß $|r_{ii}| \geqslant |h_{i+1,i}| \Longrightarrow r_m = 0$. $\Longrightarrow$ letzte Zeile von $\tilde{H}$ ist 0. □

Idee ist also mit Näherungen an Eigenwerte zu shiften z.B. mit einem Eigenwert von

$$\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{bmatrix}$$

falls der reell ist oder mit zwei aufeinanderfolgenden komplex konjugierten Shifts $\lambda, \bar{\lambda}$.
$(A - \lambda I)(A - \bar{\lambda} I)$, um komplexe Arithmetik zu vermeiden.

$$\left.\begin{aligned} A_i - \lambda I &= Q_1 R_1 \\ A_{i+1} &= R_1 Q_1 + \lambda I \\ A_{i+1} - \bar{\lambda} I &= Q_2 R_2 \\ A_{i+2} &= R_2 Q_2 + \bar{\lambda} I \end{aligned}\right| \quad (A_i - \lambda I)(A_i - \bar{\lambda} I) = (Q_1 Q_2)(R_2 R_1)$$

$\longrightarrow$ Übung $(A_i - \lambda I)(A_i - \bar{\lambda} I) = A_i^2 + 2Re(\lambda)A_i + |\lambda|^2$ ist reell auch wenn λ nicht reell. Man könnte $A_i^2 - 2Re(\lambda)A_i + |\lambda_1|^2$ berechnen und davon QR–Zerlegungen $Q_i R_i$ bilden und dann $A_{i+2} = Q_i^T A_i Q_i$. Dies ist zu teuer. Besser nutze implizites Q–Theorem.

i) Berechne erste Spalte $z = (A_i - \lambda_i I)(A_i - \bar{\lambda}_i I)e_1$.

ii) Bestimme Householder–Matrix P_0, so daß $P_0 z = \alpha e_1$.

iii) Bestimme Householder–Matrizen $P_1, P_2, \cdots, P_{n-2}$ so daß $P_{n-2}^T \cdots P_0^T A_i P_0 \cdots P_{n-2}$ wieder obere Hessenbergmatrix.

Algorithmus 6.20 (Francis QR–Schritt)

Input: *Unreduzierte obere Hessenbergmatrix $H \in \mathbb{R}^{n,n}$, dessen untere rechte 2×2 Matrix die Eigenwerte a_1, a_2 hat.*

Output: *$Z^T H Z$ wobei $Z = P_1 \cdots P_{n-2}$ Produkt von Householder–Matrizen ist und $Z^T(H - a_1 I)(H - a_2 I)$ obere $\triangle$–Matrix.*

$m = n - 1$
$s = H(m,m) + H(n,n)$
$t = H(m,m) * H(n,n) - H(m,n) * H(n,m)$
$x = H(1,1) * H(1,1) + H(1,2) * H(2,1) - s * H(1,1) + t$
$y = H(2,1) * H(1,1) + H(2,2) + s$
$z = H(2,1) * H(3,2)$
FOR $k = 0 : n-3$
  $v =$ *HOUSE* $((x,y,z)^T)$
  $H(k+1 : k+3, k+1 : n) =$ *ROWHOUSE* $(H(k+1 : k+3, k+1 : n), v)$
  $r = \min\{k+4, n\}$
  $H(1 : r, k+1 : k+3) =$ *COLHOUSE* $(H(1 : r, k+1 : k+3), v)$
  $x = H(k+2, k+1)$
  $y = H(k+3, k+1)$
  IF $k < n-3$
    $z = H(k+4, k+1)$
  END
END
$v =$ *HOUSE* $([x,y]^T)$
$H(n-1 : n, n-2 : n) =$ *ROWHOUSE*$(H(n-1 : n, n-2 : n), v)$
$H(1 : n, n-1 : n) =$ *COLHOUSE*$(H(1 : n, n-1 : n), v)$

Kosten $10n^2$ flops + $20n^2$ flops für Akkumulation von Q.

$\rightarrow$ Übung 6.58

Algorithmus 6.21 (QR–Algorithmus)

Input: $A \in \mathbb{R}^{n,n}$, *Toleranz* $tol >$ eps

Output: *Reelle Schurform* $Q^T AT = T$. *A wird mit Hessenbergform von A überschrieben. Falls* Q, T *gebraucht werden, so wird* T *in A gespeichert. Falls nur Eigenwerte gebraucht werden, so werden die Diagonal Blöcke in den entsprechenden Positionen gespeichert.*

(1) Verwende Algorithmus 6.18 zur Transformation auf Hessenbergform.

UNTIL $q == n$

Setze alle Subdiagonalelemente zu Null, die $|h_{i,i-1}| \leqslant tol\,(|h_{ii}| + |h_{i-1,i-1}|)$ *erfüllen.*

Finde größtes $q \geqslant 0$ *und kleinstes* $p \geqslant 0$ *so daß*

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ 0 & H_{22} & H_{23} \\ 0 & 0 & H_{33} \end{bmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix},$$

wobei H_{33} *quasi obere* $\triangle$*–Matrix und* H_{22} *unreduziert.*

IF $q < n$

Mache einen Francis QR–Schnitt mit $H_{22}, H_{22} = Z^T H_{22} Z$

IF Q *benötigt wird*

$Q = Q \cdot\ diag(I_p, Z, I_q)$

$H_{12} = H_{12} Z$

$H_{23} = Z^T H_{23}$

END

END

END

Bringe alle 2×2 *Blöcke mit reellen Eigenwerten auf obere* $\triangle$*–Form und akkumuliere ggf. Transformationen*

Kosten: $10n^3$ für Eigenwerte sowie $25n^3$ für Q und T

Beispiel 6.22

$$A = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 \\ 4 & 4 & 5 & 6 & 7 \\ 0 & 3 & 6 & 7 & 8 \\ 0 & 0 & 2 & 8 & 9 \\ 0 & 0 & 0 & 1 & 10 \end{bmatrix}$$

Iteration	$\prime(\lvert h_{21}\rvert)$	$\prime(\lvert h_{32}\rvert)$	$\prime(\lvert h_{43}\rvert)$	$\prime(\lvert h_{34}\rvert)$
1	10^0	10^0	10^0	10^0
2	10^0	10^0	10^0	10^0
3	10^0	10^0	10^{-1}	10^0
4	10^0	10^0	10^{-3}	10^{-3}
5	10^0	10^0	10^{-6}	10^{-5}
6	10^{-1}	10^0	10^{-13}	10^{-13}
7	10^{-1}	10^0	10^{-28}	10^{-13}
8	10^{-4}	10^0	*conv.*	*conv.*
9	10^{-8}	10^0		
10	10^{-8}	10^0		
11	10^{-16}	10^0		
12	10^{-32}	10^0		
13	*conv.*	*conv.*		

Fehleranalyse: Algorithmus berechnet $\tilde{T} = Q^T(A+E)Q$
$Q^TQ = I \quad \|E\|_2 \approx eps\ \|A\|_2$
$\tilde{Q}$ *fast orthogonal* $\quad \tilde{Q}^T\tilde{Q} = I + F \quad \|F\|_2 \approx eps$

Was macht man nun wenn man nur bestimmte Eigenvektoren zu bestimmten Eigenwerten will, nicht den invarianten Unterraum.
Es gibt verschiedene Möglichkeiten; entweder man sortiert um oder man verwendet die sogenannte Inverse Iteration. Sei μ eine Eigenwertnäherung und sei $(A-\mu I)$ nicht exakt singulär, so ist der Eigenvektor zum größten Eigenwert von $(A-\mu I)^{-1}$ gerade der gesuchte.

Algorithmus 6.23 (Inverse Iteration) *Gegeben* $A \in \mathbb{C}^{n,n} \mu \in \mathbb{C}$ *Eigenwertnäherung und Startvektor* $q^{(0)}$

FOR $k = 1, 2, \cdots$
Löse $(A-\mu I)t^{(k)} = q^{(k-1)}$ $\qquad q^{(k)} = z^{(k)}/\|z^{(k)}\|_2$
$\lambda^{(k)} = (q^{(k)})^T A q^{(k)}$
END

Kann das überhaupt klappen? $A - \mu I$ ist doch fast singulär.
Angenommen A hat Basis aus Eigenvektoren $\{x_1, \cdots, x_n\}$ und $Ax_i = \lambda x_i,\ i = 1, \cdots, n$. Falls

$$q^{(0)} = \sum_{i=1}^{n} \beta_i x_i$$

so ist $q^{(k)}$ ein Vektor in der Richtung

$$(a-\mu I)^{-k} q^{(0)} = \sum_{i=1}^{n} \frac{\beta_i}{(\lambda_i - \mu)^k} x_i. \tag{6.24}$$

Das Hauptgewicht liegt also in der Richtung von x_i, der zu $\lambda_i \approx \mu$ gehört. In diese Richtung ist das Gleichungssystem auch lösbar, da x_i im Bild von $A - \mu I$ liegt.

Abbruchkriterium Sei $r^{(k)} = (A-\mu I)q^{(k)}$ das Residuum. Man bricht die Iteration ab wenn

$$\|r^{(k)}\|_\infty \leqslant c \cdot \text{ eps } \|A\|_\infty. \tag{6.25}$$

Falls man Transformation auf Hessenbergform schon hat, so braucht jeder Schritt nur $0(n^2)$ flops.

Für symmetrische Matrizen kann man man QR genau so anwenden. Man braucht weniger Speicher und flops, symmetrische Speicherung und man erhält schnellere Konvergenz. Doppelshifts sind nicht notwendig. Viele Tricks und Varianten. Für ausführliche Inhalte siehe B. Parlett, *The Symmetric Eigenvalue Problem.*

6.3 Der QR–Algorithmus für symmetrische Matrizen

→ Übung 6.56. Reduktion symmetrischer Matrizen auf Tridiagonalgestalt.

→ Übung 6.52, 6.57. Symmetrischer QR–Algorithmus mit Wilkinson–Shift.

6.4 Die Singulärwertzerlegung

Eines der wichtigsten Hilfsmittel in allen Bereichen der numerischen Mathematik ist heutzutage die Singulärwertzerlegung einer Matrix.

Satz 6.26 *Sei A eine reelle $m \times n$ Matrix. Dann gibt es orthogonale Matrizen $U = [u_1, \cdots, u_m] \in \mathbb{R}^{m,m}$ und $V = [v_1, \cdots, v_n] \in \mathbb{R}^{n,n}$, so daß*

$$U^T AV = \ diag\ (\sigma_1, \cdots, \sigma_p) \in \mathbb{R}^{m,n},\ p = \min(m,n) \tag{6.27}$$

wobei $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_p \geqslant 0$.

Beweis: Seien $x \in \mathbb{R}^n, y \in \mathbb{R}^n$ Vektoren mit $\|x\|_2 = \|y\|_2 = 1$, so daß $Ax = \sigma y, \sigma = \|A\|_2$. Dies ist möglich, da $\|A\|_2 = \sup_{\|x\|_2=1} \|Ax\|_2$ angenommen wird.
Da jede Menge von orthogonalen Vektoren zu einer Basis des ganzen Raumes ergänzt werden kann, gibt es $V_1 \in \mathbb{R}^{n,(n-1)}$ und $U_1 \in \mathbb{R}^{m,(m-1)}$, so daß
$V = [x, V_1] \in \mathbb{R}^{n,n}, U = [y, U_1] \in \mathbb{R}^{m,m}$ orthogonal sind. Dann gilt

$$\begin{aligned} U^T AV &= \begin{bmatrix} y^T \\ U_1^T \end{bmatrix} A \begin{bmatrix} x & V_1 \end{bmatrix} = \begin{bmatrix} y^T \\ U_1^T \end{bmatrix} \begin{bmatrix} \sigma y, & AV_1 \end{bmatrix} \\ &= \begin{bmatrix} \sigma & w^T \\ 0 & B \end{bmatrix} =: A_1. \end{aligned} \tag{6.28}$$

Da

$$\left\| A_1 \begin{bmatrix} \sigma \\ w \end{bmatrix} \right\|_2^2 \geqslant (\sigma^2 + w^T w)^2, \tag{6.29}$$

so folgt, daß

$$\|A_1\|_2^2 \geqslant \sigma^2 + w^T w. \tag{6.30}$$

Aber $\sigma^2 = \|A\|_2^2 = \|A_1\|_2^2$. Also folgt $w = 0$. Per Induktion folgt das Resultat. □

Die Werte σ_i heißen Singularwerte, die u_i, v_i linke und rechte Singulärvektoren. Es gilt

$$Av_i = \sigma_i u_i, A^T u_i = \sigma_i v_i,\ i = 1, \cdots, \min(m,n) \tag{6.31}$$

$\longrightarrow$ Übung
Die Singulärwerte sind die Längen der Halbachsen der Hyperellipsoide $E = \{Ax : \|x\| = 1\}$.
$\longrightarrow$ Übung
Sei $\sigma_1 \geqslant \cdots \geqslant \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0$, so gilt

$$\begin{aligned} \text{rang } (A) &= r \\ \text{Kern } (A) &= \text{ span } \{v_{r+1}, \cdots, v_n\} \\ \text{Bild } (A) &= \text{ span } \{u_1, \cdots, u_r\} \\ \|A\|_p^2 &= \sigma_1^2 + \cdots + \sigma_p^2 \quad p = \min\{m, n\} \\ \|A\|_2 &= \sigma_1 \\ \text{cond }_2(A) &= \frac{\sigma_1}{\sigma_p} \end{aligned}$$

Die Singulärwertzerlegung wird daher zur Rangbestimmung verwendet. Weitere Anwendungen, Bildverarbeitung, Ausgleichsprobleme, Abspaltung von Kern und Co-Kern.

Satz 6.32 *Sei $A = U\Sigma V^T$ die SWZ von $A \in \mathbb{R}^{m,n}$. Sei $k < r = rang\ (A)$ und*

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T. \tag{6.33}$$

Dann gilt

$$\min_{rang(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}. \tag{6.34}$$

Beweis: $U^T A_k V = \text{diag } (\sigma_1, \cdots, \sigma_k, , 0 \cdots, 0) \Longrightarrow \text{rang } (A_k) = k$ und
$U^T(A - A_k)V = \text{diag } (0, \cdots, 0, \sigma_{k+1}, \cdots, \sigma_p) \Longrightarrow \|A - A_k\|_2 = \sigma_{k+1}$.
Angenommen rang $(B) = k$ für $B \in \mathbb{R}^{m,n}$. Dann gibt es orthonormale Vektoren $x_1, \cdots, x_{n-k}$ die Kern (B) aufspannen. Kern $(B) = \text{span } \{x_1, \cdots, x_{n-k}\}$. Aus Dimensionsgründen gilt

$$w = \text{span}\{x_1, \cdots, x_{n-k}\} \cap \text{span}\{v_1, \cdots, v_{k+1}\} \neq \{0\} \tag{6.35}$$

$(z = \sum \alpha_i v_i \quad \alpha_i = v_i^T z)$.
Sei $z \in W, \|z\|_2 = 1$. Da $Bz = 0$ und
$Az = \sum_{i=1}^{k+1} \sigma_i (v_i^T z) u_i$, so folgt

$$\|A - B\|_2^2 \geqslant \|(A - B)z\|_2^2 = \|Az\|_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 (v_i^T z) \geqslant \sigma_{k+1}^2. \tag{6.36}$$

6.4.1 Die Berechnung der Singulärwertzerlegung

Man könnte die Berechnung der Singulärwertzerlegung direkt auf ein symmetrisches Eigenwertproblem zurückführen, denn falls $A = U\Sigma V^T \in \mathbb{R}^{m,n},\ m \geqslant n$, so gilt
$AA^T = U\Sigma V^T V \Sigma^T U^T = U\Sigma^2 U^T = U \text{ diag } (\sigma_1^2, \cdots, \sigma_n^2) U^T$ und da AA^T symmetrisch ist dies eine Schurform mit spezieller Anordnung. Analog
$A^T A = V\Sigma^2 V^T = V \text{ diag } (\sigma_1^2, \cdots, \sigma_n^2, 0, \cdots, 0) V^T$. Man könnte also $A^T A$ und AA^T bilden, aber wie wir schon beim Ausgleichsproblem gesehen haben, würde dieses zur Quadrierung der Kondition führen.

Besser ist daher der Algorithmus von Golub/Kahan der auf den symmetrischen QR–Algorithmus aufbaut.
(1) Schritt transformiere A auf Bidiagonalform mit Householder–Transformationen.

Algorithmus 6.37 (Householder–Bidiagonalisierung)
Input: $A \in \mathbb{R}^{m,n}, m \geqslant n$
Output: *Der obere Bidiagonalteil von A wird dem oberen Bidiagonalteil von $B = U^T AV$ überschrieben, B obere Bidiagonalmatrix.*

$$U = U_1 \cdots U_{n-1}(\cdot U_n),\ V = V_1 \cdots V_{n-2}$$

Produkt von Householder–Matrizen. Der wesentliche Teil von U_j ist in $A(j+1:m, j)$ und der wesentliche Teil von V_j in $A(j, j+2:m)$ gespeichert.

FOR $j = 1 : n-1$
$v(j:m) = HOUSE(A(j:m, j))$;
$A(j:m, j:n) = ROWHOUSE(A(j:m, j:n), v(j:m))$;
$A(j+1:m, j) = v(j+1:m)$;
IF $j \leqslant n-2$
$v(j+1:n) = HOUSE(A(j, j+1:n)^T)$;
$A(j:m, j+1:n) = COLHOUSE(A(j:m, j+1:n), v(j+1:n))$;
$A(j, j+2:n) = v(j+2:n)^T$;
END
END
IF $m > n$
$v(n:m) = HOUSE(A(n:m, n))$;
$A(n:m, n) = ROWHOUSE(A(n:m, n), v(n:m))$;
$A(n+1:m, n) = v(n+1:m)$;
END

Kosten: $4mn^2 - 4n^3/3$ flops
Falls U, V explizit benötigt werden, diese erhält man mit $4m^2n - 4n^3/3$ für U und $4n^3/3$ für V.

Fehleranalyse: $B = U^T(A+E)V$ mit $U^TU = I, V^TV = I, \|E\|_F \leqslant cn^2 \text{ eps } \|A\|_F$.

Vorgehen:

$$\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix} \xrightarrow{U_1} \begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} \xrightarrow{V_1}$$

$$\begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} \xrightarrow{U_2} \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{bmatrix} \xrightarrow{V_2} \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{bmatrix}$$

$$\xrightarrow{U_3} \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & x \end{bmatrix} \xrightarrow{U_4} \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

(2) Wende symmetrischen QR–Algorithmus implizit auf $T = B^T B$ (tridiagonal) an,

$$B = \left[\begin{array}{cccc} d_1 & f_1 & & \\ & \ddots & \ddots & \\ & & \ddots & f_{n-1} \\ & & & d_n \\ \hline & & 0 & \end{array}\right], \; B^T B = \left[\begin{array}{cccc} d_1^2 & f_1 d_1 & & \\ f_1 d_1 & d_2^2 + f_1^2 & f_2 d_2 & \\ \ddots & \ddots & \ddots & \\ & & & f_{n-1} d_{n-1} \\ & & f_{n-1} d_{n-1} & d_n^2 + f_{n-1}^2 \end{array}\right]$$

Berechne Eigenwert λ von

$$T(n-1:n, n-1:n) = \left[\begin{array}{cc} d_{n-1}^2 + f_{n-2}^2 & d_{n-1} f_{n-1} \\ d_{n-1} f_{n-1} & d_n^2 + f_{n-1}^2 \end{array}\right]$$

der näher an $d_n^2 + f_{n-1}^2$ liegt.
Berechne die beiden Elemente von

$$(T - \lambda I) e_1 = \left[\begin{array}{c} d_1^2 - \lambda \\ d_1 f_1 \end{array}\right]$$

und eine Givensrotation $G = \left[\begin{array}{cc} c_1 & s_1 \\ -s_1 & c_1 \end{array}\right]$

$$\left[\begin{array}{cc} c_1 & s_1 \\ -s_1 & c_1 \end{array}\right]^T \left[\begin{array}{c} d_1^2 - \lambda \\ d_1 f_1 \end{array}\right] = \left[\begin{array}{c} * \\ 0 \end{array}\right]$$

und transformiere

$$BG_1 = \left[\begin{array}{ccccc} * & * & & & \\ \circledast & * & * & & \\ & & \ddots & \ddots & \\ & & & \ddots & * \\ & & & & * \end{array}\right]$$

wieder auf Bidiagonalform.
$\longrightarrow$ Übung

Berechnung und Anwendung von Givensrotationen

Bestimme $c, s \in \mathbb{R}$ mit $c^2 + s^2 = 1$, so daß

$$\left[\begin{array}{cc} c & -s \\ s & c \end{array}\right] \left[\begin{array}{c} a \\ b \end{array}\right] = \left[\begin{array}{c} * \\ 0 \end{array}\right].$$

Wichtige Details:
Stabilität der Berechnung von c, s. Im Prinzip ist $c = \pm b/\sqrt{a^2 + b^2}, s = \mp a/\sqrt{a^2 + b^2}$. Man berechnet aber den Quotienten $t = a/b$ bzw. $t = b/a$ und macht Fallunterscheidung. Aus Stabilitätgründen sollte $|t| \leqslant 1$ gelten.

Algorithmus 6.38 (Erzeugung der Givens–Parameter)

Input: $a, b \in \mathbb{R}$.

Output: $c, s \in \mathbb{R}$ *mit* $c^2 + s^2 = 1$, *so daß* $\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$.

```
FUNCTION    [c,s] = GIVENS(a,b)
            IF      |b| > |a|
                    t = -a/b;
                    s = 1/sqrt(1+t^2);
                    c = s*t;
            ELSE
                    t = -b/a;
                    c = 1/sqrt(1+t^2);
                    s = c*t;
            END
END GIVENS
```

Kosten: 5 flops, 1 $\sqrt{}$.

Algorithmus 6.39 (Vormultiplikation mit Givens–Rotation)

Input: $A \in \mathbb{K}^{2,n}, c, s \in \mathbb{R}, c^2 + s^2 = 1$.

Output: A *überschrieben mit* $\begin{bmatrix} c & -s \\ s & c \end{bmatrix} A$.

```
FUNCTION    A = ROWROT(A,c,s)
            A = [c,-s;s,c]*A;
END ROWROT
```

Kosten: $6n$ flops.
Fehler: $gl(\tilde{G}A) = G(A+E),\ \|E\|_2 = \mathcal{O}(\text{eps}\,\|A\|_2)$

Algorithmus 6.40 (Nachmultiplikation mit Givens–Rotation)

Input: $A \in \mathbb{K}^{m,2}, c, s \in \mathbb{R}, c^2 + s^2 = 1$.

Output: A *überschrieben mit* $A \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$.

```
FUNCTION    A = COLROT(A,c,s)
            A = A*[c,s;-s,c];
END COLROT
```

Kosten: $6m$ flops.
Fehler: $gl(A\tilde{G}) = (A+E)G,\ \|E\|_2 = \mathcal{O}(\text{eps}\,\|A\|_2)$

Diese 3 Algorithmen werden verwendet um bestimmte einzelne Einträge einer Matrix zu eliminieren.
Dieses ist ideal für den SWZ-Schritt.

Algorithmus 6.41 (Golub/Kahan SWZ–Schritt)

Input: *Bidiagonalmatrix $B \in \mathbb{R}^{m,n}$, die keine Nullen auf Diagonale und Superdiagonale hat.*

Output: *B überschrieben mit der Bidiagonalmatrix $\bar{B} = \bar{U}^T B \bar{V}, \bar{U}, \bar{V}$ orthogonal und $\bar{V}$ ist im wesentlichen die orthogonale Matrix die man erhalten würde, wennman die symmetrische Version des QR–Algorithmus auf $T = B^T B$ anwenden würde.*

Sei μ der Eigenwert der rechten unteren 2×2 Matrix von $B^T B$ der näher am letzten Diagonalelement von $B^T B$ ist.

$y = B(1,1)^2 - \mu$
$z = B(1,1) * B(1,2)$
FOR $k = 1 : n-1$
  $[c, s] = GIVENS(y, z);$
  $l = \max\{1, k-1\}$
  $B(l : k+1, k : k+1) = COLROT(B(l : k+1, k : k+1), c, s);$
  $y = B(k, k);$
  $z = B(k+1, k);$
  $[c, s] = GIVENS(y, z);$
  $l = \min\{k+2, n\}$
  $B(k : k+1, k : l) = ROWROT(B(k : k+1, k : l), c, s);$
  IF $k < n-1$
    $y = B(k, k+1);$
    $z = B(k, k+2);$
  END
END

Kosten: $30n$ flops, $2n$ $\sqrt{}$
Akkumulation von U $6mn$ flops
Akkumulation von V $6n^2$ flops
Fehler: Wie bei QR.
Deflation wie bei QR–Algorithmus

Die Anfangsreduktion auf Bidiagonalform und die Iterationsfolge zusammen mit Deflation wird zusammengefaat im SWZ–Algorithmus.

Algorithmus 6.42 (SWZ Berechnung–Golub/Reinsch)

Input: $A \in \mathbb{R}^{m,n}, m \geqslant n, \varepsilon = c \cdot \text{ eps}\,, c$ *klein.*
Output: A *überschrieben mit* $U^T AV = D + E \quad U, V$ *orthogonal,* D *diagnonal,* $\|E\|_2 \approx \text{eps}\,\|A\|_2$.

Bidiagonalisiere A *mit Algorithmus 6.37.* $q = 0$*;*
WHILE $q < n$
Setze $A(i, i+1)$ *zu Null, falls*
$|A(i, i+1)| \leqslant \varepsilon(|A(i,i)| + |A(i+1, i+1)|),\ i = 1 : n-1.$
Bestimme größtes q *und kleinstes* p*, so daß in*

$$B = \begin{array}{c} \\ p \\ n-p-q \\ q+m-n \end{array} \begin{array}{c} \begin{array}{ccc} p & n-p-q & q \end{array} \\ \begin{bmatrix} B_{11} & 0 & 0 \\ 0 & B_{22} & 0 \\ 0 & 0 & B_{33} \end{bmatrix} \end{array}$$

B_{33} *diagonal ist, und* B_{22} *nur Nebendiagonalelemente* $\neq 0$ *hat.*
IF $q < n$
Setze in B_{22} *alle* $A(i,i)$ *zu Null, falls* $|A(i,i)| \leqslant \varepsilon \|B_{22}\|$.
IF Diagonalelement von B_{22} *Null*
mache Spezialschritt (unten)
END
Wende Algorithmus 6.41 auf B_{22} *an*
END
END
Spezialschritt:
Sei k *der größte Index, so daß für die Einträge in* B_{22} $A(k,k) = 0$ *ist und* $A(l,l) \neq 0, l > k$.
Schiebe Eintrag an Position $(k, k+1)$ *nach rechts heraus*
FOR $j = k+1 : n-p-q$
Eliminiere Eintrag in Position (k, j)
$[c, s] = GIVENS(A(j,j), A(k,j));$
$A([j,k], j) = ROWROT(A([j,k], j), c, s);$
Neuer Eintrag in Position $(k, j+1)$ *kommt herein*
IF $j < n-p-q$
$A([j,k], j+1) = ROWROT(A([j,k], j+1), c, s);$
END
END
Schiebe Eintrag an Position $(k-1, k)$ *nach oben heraus*
FOR $j = k-1 : -1 : p+1$
Eliminiere Eintrag in Position (j, k)
$[c, s] = GIVENS(A(j,j), A(j,k));$
$A(j, [j,k]) = COLROT(A(j, [j,k]), c, s);$
Neuer Eintrag in Position $(j-1, k)$ *kommt herein*
IF $j > p+1$
$A(j-1, [j,k]) = COLROT(A(j-1, [j,k]), c, s);$
END
END

		flops
	Σ	$4mn^2 - 4n^3/3$
Koten:	Σ, V	$4mn^2 + 8n^3$
	Σ, U	$4m^2n - 8mn^2$
	Σ, U, V	$4m^2n + 8mn^2 + 9n^3$

Den Spezialschritt verdeutlichen wir an folgendem Beispiel
Zuerst schieben wir den Eintrag $(k, k+1)$ nach rechts heraus

$$\left[\begin{array}{cc|c|cc} * & * & & & \\ & * & \circledast & & \\ \hline & & 0 & \circledast & \\ \hline & & & * & * \\ & & & & * \end{array}\right] \mapsto \left[\begin{array}{cc|c|ccc} * & * & & & \\ & * & \circledast & & \\ \hline & & 0 & 0 & \circledast \\ \hline & & & \circledast & \circledast \\ & & & & * \end{array}\right] \mapsto \left[\begin{array}{cc|c|ccc} * & * & & & \\ & * & * & & \\ \hline & & 0 & 0 & 0 \\ \hline & & & \circledast & \circledast \\ & & & & \circledast \end{array}\right]$$

Dann schieben wir den Eintrag $(k-1, k)$ nach oben heraus

$$\left[\begin{array}{cc|c|cc} * & * & & & \\ & * & \circledast & & \\ \hline & & 0 & 0 & 0 \\ \hline & & & \circledast & \circledast \\ & & & & \circledast \end{array}\right] \mapsto \left[\begin{array}{cc|c|ccc} * & \circledast & \circledast & & \\ & \circledast & 0 & & \\ \hline & & 0 & 0 & 0 \\ \hline & & & \circledast & \circledast \\ & & & & \circledast \end{array}\right] \mapsto \left[\begin{array}{cc|c|ccc} \circledast & \circledast & 0 & & \\ & \circledast & 0 & & \\ \hline & & 0 & 0 & 0 \\ \hline & & & \circledast & \circledast \\ & & & & \circledast \end{array}\right]$$

Beispiel 6.43

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

Iteration	$\mathcal{O}(a_{21})$	$\mathcal{O}(a_{23})$	$\mathcal{O}(a_{34})$
1	10^0	10^0	10^0
2	10^0	10^0	10^0
3	10^0	10^0	10^0
4	10^0	10^{-1}	10^{-2}
5	10^0	10^{-3}	10^{-8}
6	10^0	10^{-1}	10^{-27}
7	10^0	10^{-1}	–
8	10^0	10^{-4}	–
9	10^{-1}	10^{-14}	–
10	10^{-1}	–	–
11	10^{-4}	–	–
12	10^{-12}	–	–
13	–	–	–

Also kubische Konvergenz.
Weiteres siehe Golub / van Loan.

6.5 Der Bisektionsverfahren zur Berechnung der Eigenwerte symmetrischer tridiagonaler Matrizen

Sei T_r die führende Hauptabschnittsmatrix von

$$T = \begin{bmatrix} a_1 & b_1 & & & \\ \bar{b}_1 & a_2 & b_2 & & \\ & \bar{b}_2 & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ & & & \bar{b}_{n-1} & a_n \end{bmatrix} = T^* \tag{6.44}$$

und sei $p_r(x) = \det(T_r - xI), \quad r = 1 : n$ dann gilt

$$p_r(x) = (a_r - x)p_{r-1}(x) - (b_{r-1})^2 p_{r-2}(x) \quad r = 2 : n \tag{6.45}$$

$\rightarrow$ Übung 6.59 Man kann $p_r(x)$ in $0(r)$ flops auswerten.

Algorithmus 6.46 *Bisektion zur Berechnung der Eigenwerte von T.*

Input: T wie in (6.44) symmetrisch, tridiagonal, Toleranzgrenze tol, $y, z \in \mathbb{R}, y < z, p_n(y)p_n(z) < 0$.

WHILE $|y - z| > tol\ (|y| + |z|)$

$x = \frac{y+z}{2}$
IF $p_n(x)p_n(x) < 0$
$z = x$
ELSE
$y = x$

END

END

Kosten: $0(h)$ *flops. Fehler halbiert sich pro Schritt, lineare Konvergenz.*
Parallel auswertbar.

Satz 6.47 (Sturm'sche Kette) *Sei T in (6.44) unreduziert dann separieren die Eigenwerte von T_{r-1} die Eigenwerte von T_r.*

$$\lambda_r(T_r) < \lambda_{r-1}(T_{r-1}) < \cdots < \lambda_2(T_r) < \lambda_1(T_{r-1} < \lambda_1(T_r) \tag{6.48}$$

Falls $a(\lambda)$ die Anzahl der Zeichenwechsel in der Folge $\{p_0(\lambda), \cdots, p_n(\lambda)\}$ ist, so ist $a(x)$ die Anzahl der Eigenwerte von $T <$ als x.

Beweis: z.B. Stoer □

Damit können wir einen weiteren Algorithmus zur Bisektion bekommen.

Algorithmus 6.49 (Sturm'sche Ketten, Bisektion) *Input: T wie in (6.44), Toleranz tol und Polynomfolge wie in 6.45, $[a, b]$, so daß alle Eigenwerte von T in $[a, b]$. Sei $x = b, y = a$.*

WHILE $|y-z| > tol\,(|y|+|z|)$

$x = (y+z)/2$
IF $a(x) \geqslant k$
$z = x$
ELSE
$y = x$
END

END

Man kann $y(\lambda)$ auch mittels Cholesky–Zerlegung bestimmen.

6.6 Übungen zu Kapitel 6

Übung 6.50 *a) Zeige den Satz von Gerschgorin: Sei $A = (a_{ij})_{i,j=1,\ldots,n} \in \mathbb{C}^{n,n}$. Dann sind die Eigenwerte von A enthalten in der Vereinigung der Kreisscheiben*

$$\bigcup_{i=1}^{n} K_i,\ K_i = \{z:\ |z-a_{ii}| \leqslant \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}|\}.$$

b) Zeige folgende verschärfte Variante des Satzes von Gerschgorin: Sind $i_1, \ldots, i_r$ (paarweise verschieden) so gewählt, daß

$$\left(\bigcup_{k=1}^{r} K_{i_k}\right) \cap \left(\bigcup_{\substack{i=1\\ i\neq i_1,\ldots,i_r}}^{n} K_i\right),$$

dann enthält $\bigcup_{k=1}^{r} K_{i_k}$ bereits r (nicht zwingend verschiedene) Eigenwerte von A
Hinweis: Die Eigenwerte einer Matrix hängen stetig von den Koeffizienten der Matrix ab.

Übung 6.51 *Zeige den Satz von Bauer und Fike: Sei $A = (a_{ij})_{i,j=1,\ldots,n} \in \mathbb{C}^{n,n}$ diagonalisierbar, d.h., nehme an, daß $A = X\Lambda X^{-1}$ ist, wobei $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ ist und X nicht singulär ist. Sei $E \in \mathbb{C}^{n,n}$ irgend eine Matrix. Dann existiert zu jedem Eigenwert μ von $A+E$ ein Eigenwert λ_j von A derart, daß*

$$|\lambda_j - \mu| \leqslant \|X\|_p \|X^{-1}\|_p \|E\|_p.$$

Dabei ist $\|\bullet\|_p$ die übliche induzierte Matrixnorm zur p–Norm für Vektoren.
Hinweis: Betrachte o.B.d.A. $\mu \neq \lambda_j$ und betrachte $(\mu I - \Lambda)X^{-1}v$ für ein geeignetes v.

Übung 6.52 *Schreibe ein* **MATLAB** *–Programm für den QR–Algorithmus für symmetrische Matrizen einmal mit Wilkinson–Shift und einmal ohne Shift (Der Wilkinson–Shift einer symmetrischen Matrix A ist der Eigenwert von $\begin{pmatrix} a_{n-1,n-1} & a_{n-1,n} \\ a_{n-1,n} & a_{nn} \end{pmatrix}$, welcher dichter bei a_{nn} gelegen ist). Für die Reduktion auf symmetrische Tridiagonalgestalt darf die entsprechende Routine 'hess' aus* **MATLAB** *verwendet werden. Für die Iteration reicht die explizite Variante.*
Vergleiche beide Varianten bezüglich Rechenzeit und Genauigkeit anhand der Matrix $A = \begin{pmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}$ für Dimension $n = 10, 20, \ldots, 100$.

Übung 6.53 *Sei $A \in \mathbb{C}^{n,n}$ eine nichtzerfallende obere Hessenbergmatrix und $\mu \in \mathbb{C}$. Zeige:*

1. *Falls $A - \mu I = QR$ eine QR–Zerlegung von $A - \mu I$ ist, dann ist Q eine obere Hessenbergmatrix (d.h. $q_{ij} = 0, i > j + 1$). Ist μ ein Eigenwert von A, dann ist $r_{nn} = 0$.*
2. *$H = Q^H AQ$ ist eine obere Hessenbergmatrix. Ist μ ein Eigenwert von A, dann ist $h_{n,n-1} = 0$. Was bedeutet das für die letzte Spalte von Q?*
3. *Nehme an A sei reell. Dann ist auch $(A - \mu I)(A - \bar{\mu} I)$ is reell.*
4. *Sei A reell und $\mu = \alpha + i\beta$ ein Eigenwert von A. Falls $A^2 - 2\alpha A + (\alpha^2 + \beta^2)I = QR$ eine QR–Zerlegung ist, dann ist $\begin{pmatrix} r_{n-1,n-1} & r_{n-1,n} \\ 0 & r_{nn} \end{pmatrix}$ is singulär.*

Übung 6.54 *Sei $A \in \mathbb{C}^{n,n}$ obere Hessenbergmatrix und p ein Polynom. Nehme an, dass $p(A)$ nichtsingulär ist. Zeige: Ist $p(A) = QR$ eine QR–Zerlegung von $p(A)$, dann ist $B = Q^H AQ$ eine obere Hessenbergmatrix. Gibt es eine Möglichkeit B zu berechnen, ohne das ganze Polynom $p(A)$ auszurechnen?*

Übung 6.55 *Schreibe ein* **MATLAB** *–Programm für die Reduktion auf obere Hessenberggestalt mittels Householder–Transformationen. Vergleiche das Programm bezüglich Genauigkeit und Rechenaufwand mit der* **MATLAB** *–Routine 'hess' anhand zufällig erzeugter Beispiele der Dimensionen $n = 10, 20, \ldots, 100$.*

Übung 6.56 *Schreibe ein* **MATLAB** *–Programm für die Reduktion symmetrischer Matrizen auf Tridiagonalgestalt mittels Householder–Transformationen. Vergleiche das Programm bezüglich Genauigkeit und Rechenaufwand mit der* **MATLAB** *–Routine 'hess' anhand zufällig erzeugter Beispiele der Dimensionen $n = 10, 20, \ldots, 100$.*

Übung 6.57 *Schreibe ein* **MATLAB** *–Programm für den QR–Algorithmus für symmetrische Matrizen einmal mit Wilkinson–Shift und einmal ohne Shift (Der Wilkinson–Shift einer symmetrischen Matrix A ist der Eigenwert von $\begin{pmatrix} a_{n-1,n-1} & a_{n-1,n} \\ a_{n-1,n} & a_{nn} \end{pmatrix}$, welcher dichter bei a_{nn} gelegen ist). Für die Reduktion auf symmetrische Tridiagonalgestalt darf die entsprechende Routine 'hess' aus* **MATLAB** *verwendet werden. Für die Iteration soll die implizite Variante verwendet werden. Vergleiche beide Varianten bezüglich Rechenzeit und Genauigkeit anhand der Matrix $A = \begin{pmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}$ für Dimension $n = 10, 20, \ldots, 100$.*

Übung 6.58 *Schreibe ein* **MATLAB** *–Programm für den QR–Algorithmus für allgemeine reelle Matrizen mit implizitem Francis–Shift (Eigenwerte der Matrix $\begin{pmatrix} a_{n-1,n-1} & a_{n-1,n} \\ a_{n,n-1} & a_{nn} \end{pmatrix}$). Für die Reduktion auf Hessenberggestalt darf die entsprechende Routine 'hess' aus* **MATLAB** *verwendet werden. Vergleiche das Verfahren bezüglich Rechenzeit und Genauigkeit mit der Funktion 'schur' aus* **MATLAB** *anhand zufällig erzeugter Matrizen der Dimensionen $n = 10, 20, \ldots, 100$.*

Übung 6.59 *Gegeben sei eine symmetrische Tridiagonalmatrix*

$$A = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_n \\ & & \beta_n & \alpha_n \end{pmatrix} \in \mathbb{R}^{n,n}.$$

Zeige: Das charakteristische Polynom $p_n(t) = \det(A - tI)$ von A lässt sich rekursiv berechnen durch

$$\begin{aligned} p_0(t) &= 1 \\ p_1(t) &= \alpha_1 - t \\ p_j(t) &= (\alpha_j - t)p_{j-1}(t) - \beta_j^2 p_{j-2}(t),\ j = 2, 3, \ldots, n. \end{aligned}$$