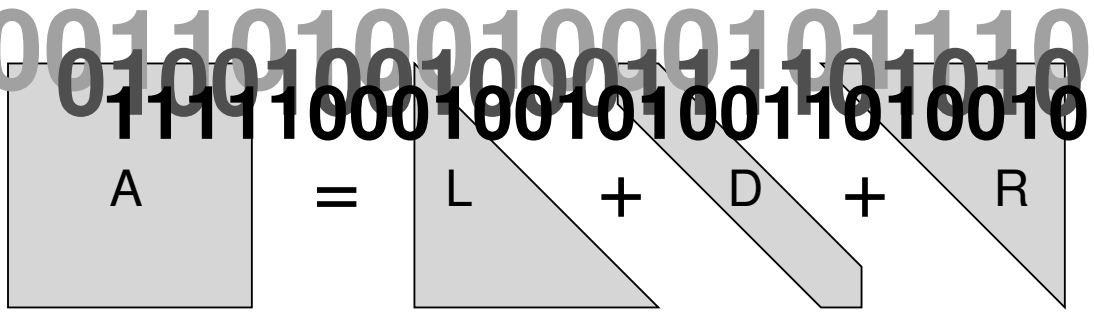
Stefan Funken

Numerik I

(Einführung in die Numerische Lineare Algebra)



SKRIPT, UNIVERSITÄT ULM, WINTERSEMESTER 2008/2009

Danksagung. Frau Kristin Kirchner und Herrn Moritz Reinhard bin ich für das sorgfältige Lesen des Manuskripts zu besonderem Dank verpflichtet. Ihre zahlreichen Kommentare, Vorschläge, Korrekturen und Hinweise haben die Qualität des Textes wesentlich verbessert. Insbesondere möchte ich Herrn Reinhard für die Ausarbeitung der Aufgaben und zugehöriger Lösungen danken.

Ganz besonderer Dank gebührt auch Frau Petra Hildebrand, die meine handschriftlichen Aufzeichnungen in LaTeX umgesetzt und zahlreiche Grafiken erstellt hat. Frau Brandner, Frau Serbine und Herrn Weithmann möchte ich für das LaTeX'en der Lösungen danken.

Die aufmerksame Leserin oder Leser sei ermuntert, mir Hinweise auf Druckfehler, sprachliche Unzulänglichkeiten oder inhaltliche Flüchtigkeiten per Email zukommen zu lassen (`stefan.funken@uni-ulm.de`).

Stand. Ulm, Oktober 2010.

Inhaltsverzeichnis

1 LINEARE GLEICHUNGSSYSTEME

1.1 EINFÜHRUNG, CRAMERSCHE REGEL

Wir beginnen mit dem klassischen Verfahren zur Lösung eines linearen Gleichungssystems (LGS), der Gaußschen[1] Eliminationsmethode.

Zu lösen ist ein System von n linearen Gleichungen mit n Unbekannten $x_1, \ldots, x_n \in \mathbb{R}$

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array} \tag{1.1}$$

oder kurz

$$Ax = b, \tag{1.2}$$

wobei $A \in \mathbb{R}^{n\times n}$ eine reelle $n \times n$-Matrix ist und $b, x \in \mathbb{R}^n$ reelle (Spalten-)Vektoren sind.

Wann ist ein lineares Gleichungssystem überhaupt lösbar? Aus der Linearen Algebra (siehe z.B. [Wille]) kennen wir das folgende Resultat, das die Lösbarkeit mit Hilfe der Determinante der Matrix A charakterisiert.

Satz 1.1.1 (Lösbarkeit) *Sei $A \in \mathbb{R}^{n\times n}$ mit $\det A \neq 0$ und $b \in \mathbb{R}^n$. Dann existiert genau ein $x \in \mathbb{R}^n$, so dass $Ax = b$.*

Falls $\det A \neq 0$, so lässt sich die Lösung $x = A^{-1}b$ mit der **Cramerschen Regel** berechnen, d.h.

$$x_i = \frac{1}{\det A} \begin{vmatrix} a_{11} & \ldots & b_1 & \ldots & a_{1n} \\ a_{21} & \ldots & b_2 & \ldots & a_{2n} \\ \vdots & & \vdots & & \vdots \\ a_{nn} & \ldots & b_n & \ldots & a_{nn} \end{vmatrix} = \frac{D_i}{D} \qquad (i = 1, \ldots, n).$$

Dabei geht die im Zähler stehende Determinante D_i dadurch aus $D = \det A$ hervor, dass man deren i-te Spalte durch den Vektor b der rechten Seite ersetzt.

Man beachte hier die Verbindung von Existenz- und Eindeutigkeitsaussage mit dem Rechenverfahren, was einen „guten" Algorithmus ausmacht. Dieser braucht dabei nicht unbedingt optimal zu sein!

Die Determinanten von $n \times n$-Matrizen lassen sich mittels der **Leibnizschen Darstellung** berechnen, d.h.

$$\det A := \sum_{\pi} (-1)^{j(\pi)} a_{1i_1}\, a_{2i_2} \cdots a_{ni_n}\,,$$

wobei die Summe über alle möglichen $n!$ Permutationen π der Zahlen $1, 2, \ldots, n$ zu berechnen ist. Der Wert des Ausdrucks $(-1)^{j(\pi)}$ ergibt sich aus der Anzahl $j(\pi)$ der Inversionen der Permutation $\pi = \begin{pmatrix} 1 & 2 & \ldots & n \\ i_1 & i_2 & \ldots & i_n \end{pmatrix}$.

[1] **Carl Friedrich Gauß**, 1777 - 1855. Lagrange hatte 1759 die Methode schon vorweggenommen und in China war sie schon vor dem ersten Jahrhundert bekannt. Näheres zu Gauß, Lagrange und weiteren Mathematikern findet man im Internet unter `www-groups.dcs.st-andrews.ac.uk/~history`.

Bemerkung 1.1.2 (Rechenoperationen) *Im Folgenden werden die Verknüpfungen Multiplikation, Addition, Division und Subtraktion in ihrem Rechenaufwand nicht unterschieden und unter dem Begriff* **Gleitkommaoperation** *zusammengefasst (1 Gleitkommaoperation $\simeq$ 1 FLOP[2]). Systematische Multiplikationen mit ± 1 bleiben im Allgemeinen unberücksichtigt. Anderweitige Operationen wie z.B. Wurzelziehen werden gesondert betrachtet.*

Satz 1.1.3 (Aufwand Leibnizsche Darstellung) *Sei $A \in \mathbb{R}^{n\times n}$. Der Aufwand zur Berechnung von $\det A$ mit der Leibnizschen Darstellung, d.h. als Summe über alle Permutationen der Menge $\{1,\dots,n\}$, beträgt*

$$\text{FLOP}(\det A) = n\,n! - 1\,.$$

Beweis. Der Aufwand zur Berechnung von $\det A$ für $A \in \mathbb{R}^{n\times n}$ in der Leibnizschen Darstellung (unter Vernachlässigung der Multiplikation mit $(-1)^{j(\pi)}$) ergibt sich wie folgt:

$$\begin{aligned}\text{FLOP}(\det A) &= \text{" } (n!-1) \text{ Additionen } + n! \text{ Produkte mit } n \text{ Faktoren "} \\ &= n! - 1 + n!(n-1) = n\,n! - 1\,.\end{aligned}$$

Damit ist die Aussage des Satzes bewiesen. □

Alternativ lässt sich die Determinante rekursiv mit Hilfe des **Laplaceschen Entwicklungssatzes** berechnen. Dieser lautet für eine quadratische Matrix $A \in \mathbb{R}^{n\times n}$

$$\det A = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}),$$

wobei A_{1j} diejenige Matrix ist, die aus A durch Streichen der ersten Zeile und der j-ten Spalte entsteht.

Satz 1.1.4 (Aufwand Laplacescher Entwicklungssatz) *Sei $A \in \mathbb{R}^{n\times n}$ ($n \geq 2$). Der Aufwand zur Berechnung von $\det A$ mit dem Laplaceschen Entwicklungssatz beträgt*

$$\text{FLOP}(\det A) = 2\,n! - 1\,.$$

Beweis. Wir zeigen zuerst induktiv, dass für $n \geq 2$

$$\sum_{k=1}^{n-1} \frac{k}{(k+1)!} = 1 - \frac{1}{n!} \tag{1.3}$$

gilt. Der Induktionsanfang für $n = 2$ ist offensichtlich erfüllt. Der Induktionsschritt $n \to n+1$ ergibt sich aus

$$\sum_{k=1}^{n} \frac{k}{(k+1)!} = \sum_{k=1}^{n-1} \frac{k}{(k+1)!} + \frac{n}{(n+1)!} = \frac{(n!-1)}{n!} + \frac{n}{(n+1)!} = 1 - \frac{1}{(n+1)!}\,.$$

Nun zeigen wir weiter, dass der Aufwand zur Berechnung der Determinante einer $n \times n$-Matrix mit dem Laplaceschen Entwicklungssatz

$$\text{FLOP}\left(\det(\mathbb{R}^{n\times n})\right) = n!\left(1 + \sum_{k=1}^{n-1} \frac{k}{(k+1)!}\right) \tag{1.4}$$

[2] **FLOP** ist nicht zu verwechseln mit **FLOP/s** oder **flops** (floating point operations per second), welches als Maßeinheit für die Geschwindigkeit von Computersystemen verwendet wird.

beträgt.
Induktionsanfang ($n = 2$): Die Determinante der Matrix $A = (a_{ij}) \in \mathbb{R}^{2\times 2}$ lautet $a_{11}a_{22} - a_{21}a_{12}$, d.h. der Aufwand beträgt 2 Multiplikationen und 1 Addition, also

$$\text{FLOP}\left(\det(\mathbb{R}^{2\times 2})\right) = 3 = 2!\left(1 + \frac{1}{2!}\right).$$

Induktionsschritt ($n \to n+1$): Für $n \geq 2$ gilt

$$\begin{aligned}
\text{FLOP}\left(\det(\mathbb{R}^{(n+1)\times(n+1)})\right) &= \text{"}n \text{ Additionen und } (n+1) \text{ Multiplikationen mit } \det(\mathbb{R}^{n\times n})\text{"} \\
&= n + (n+1)\cdot \text{FLOP}(\det(\mathbb{R}^{n\times n})) \\
&= \frac{(n+1)!\,n}{(n+1)!} + (n+1)!\left(1 + \sum_{k=1}^{n-1}\frac{k}{(k+1)!}\right) \\
&= (n+1)!\left(1 + \sum_{k=1}^{n-1}\frac{k}{(k+1)!} + \frac{n}{(n+1)!}\right) \\
&= (n+1)!\left(1 + \sum_{k=1}^{n}\frac{k}{(k+1)!}\right).
\end{aligned}$$

Aus den Gleichungen (1.3) und (1.4) ergibt sich die Behauptung des Satzes. □

Beispiel 1.1.5 Ein einfaches Beispiel soll zeigen, dass man die Determinante überhaupt nur für sehr kleine allgemeine Matrizen $n \ll 23$ mit dem Laplaceschen Entwicklungssatz berechnen kann.
Ein Jahr hat $365 \cdot 24 \cdot 60 \cdot 60 \approx 3 \cdot 10^7$ Sekunden. Geht man nun von einem schnellen Rechner[3] mit 180 TFlops aus, so benötigt man zur Berechnung der Determinante einer 20×20-Matrix mit dem Laplaceschen Entwicklungssatz

$$\frac{\text{Anzahl der Operationen}}{\text{Gleitkommaoperationen pro Sekunde}}\,[s] = \frac{2\cdot 20! - 1}{180\cdot 10^{12}}\,[s] \approx 27032\,[s] \approx 7.5\,[h]$$

bzw. für eine 23×23 -Matrix

$$\frac{2\cdot 23! - 1}{180\cdot 10^{12}}\,[s] \approx 2.872\cdot 10^8\,[s] \approx 9.1 \text{ Jahre}.$$

Bemerkung 1.1.6 *Bei der Cramerschen Regel ist zum Lösen eines linearen Gleichungssystems $Ax = b$ mit $A \in \mathbb{R}^{n\times n}$ und $b \in \mathbb{R}^n$ die Berechnung von $n+1$ Determinanten und n Quotienten notwendig. Der Aufwand zur Lösung eines linearen Gleichungssystems lässt sich somit zusammenfassen, wobei wir auf die Komplexität des Gauß-Verfahrens (siehe Seite 9) erst später eingehen werden.*

Cramersche Regel		*Gauß-Elimination*
Leibniz	*Laplace*	
$n\,(n+1)! - 1$	$2(n+1)! - 1$	$\dfrac{4n^3 + 9n^2 - n}{6}$

[3]Der weltweit auf Platz sechs rangierende und zugleich schnellste europäische Supercomputer JUGENE am nordrhein-westfälischen Forschungszentrum Jülich hat 180 TFlops (Stand Okt. 2008). Ein aktueller PC hat eine Leistung von 1-10 GFlops. T = Tera = 10^{12}, G = Giga = 10^9.

Bemerkung 1.1.7 *Für das Lösen eines LGS mit 20 Unbekannten mit der Cramerschen Regel und dem Laplaceschen Entwicklungssatz benötigt man auf einem der schnellsten Rechner etwa eine knappe Woche, d.h.* $21 \cdot 7.5[h] = 157.5[h] = 6.5625[d]$ *(20 verschiedene Determinanten im Zähler und eine im Nenner, vgl. Beispiel 1.1.5). Ein System mit 23 Unbekannten ist mit einem heutigen Supercomputer und der Cramerschen Regel nicht in einem Menschenleben zu lösen.*

Beispiel 1.1.8 (Vergleich Rechenaufwand) Aufwand zum Lösen eines linearen Gleichungssystems $Ax = b$ via Cramerscher Regel und Gauß-Verfahren.

Für einige n sei die Anzahl der notwendigen FLOPs wiedergegeben.

	Cramer/Leibniz	Cramer/Laplace	Gauß
n	$= n\,(n+1)! - 1$	$= 2(n+1)! - 1$	$= (4n^3 + 9n^2 - n)/6$
$n = 2$	11	11	11
$n = 3$	71	47	31
$n = 4$	479	239	66
$n = 5$	3599	1439	120
$n = 8$	2903039	725759	436
$n = 10$	399167999	79833599	815

Im Folgenden werden wir nun Verfahren beschreiben, die bereits für $n \geq 3$ effektiver als die Cramersche Regel sind. Mit Hilfe dieser Verfahren lassen sich auch Determinanten in polynomialer Zeit bestimmen. Daher wird die Cramersche Regel im Allgemeinen nur für $n = 2,\ 3$ verwendet. Der Vorteil der Cramerschen Regel ist jedoch die explizite Schreibweise, d.h. sie ist eine Formel für alle Fälle. Man spart sich bei ähnlichem Rechenaufwand (d.h. $n = 2, 3$) aufwendige Fallunterscheidungen (z.B. Pivotwahl) im Programm.

1.2 Gestaffelte Systeme

Betrachten wir zuerst besonders einfach zu lösende Spezialfälle. Am einfachsten ist sicherlich der Fall einer diagonalen Matrix A.

Diagonalmatrix

Man bezeichnet eine Diagonalmatrix $A = (a_{ij})$ als Diagonalmatrix, falls $a_{ij} = 0$ für $i \neq j$ gilt. Häufig schreibt man eine Diagonalmatrix A mit Diagonaleinträgen $a_{11}, \ldots, a_{nn}$ auch einfach $A = \text{diag}(a_{11}, \ldots, a_{nn})$. Die Anwendung der Inversen einer Diagonalmatrix A mit Diagonalelementen $a_{ii} \neq 0$ auf einen Vektor b lässt sich als Pseudocode wie folgt schreiben:

Algorithmus 1.2.1: Lösen von $Ax = b$ mit Diagonalmatrix A

Sei $A \in \mathbb{R}^{n\times n}$ eine invertierbare Diagonalmatrix und $b \in \mathbb{R}^n$

Input $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$

for $j = 1, \ldots, n$

$x_j = b_j / a_{jj}$

end

Output $x = (x_1, \cdots, x_n)^T$

Eine einfache Matlab Realisierung ist im Folgenden dargestellt.

MATLAB-Beispiel:

Man löse $Ax = b$ mit

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix}, \; b = \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix}.$$

```
>> A=diag([1,2,4]);
>> b=[5;3;2];
>> for j=1:3, x(j)=b(j)/A(j,j); end
>> x
x =
    5.0000    1.5000    0.5000
```

Dreiecksmatrix

Der nächstschwierigere Fall ist der einer **Dreiecksmatrix** A. Man spricht von einer oberen Dreiecksmatrix $A = (a_{ij})$, falls $a_{ij} = 0$ für $i > j$, und von einer unteren Dreiecksmatrix $A = (a_{ij})$, falls $a_{ij} = 0$ für $i < j$ gilt. Häufig verwenden wir auch R für eine obere Dreiecksmatrix und L für eine untere Dreiecksmatrix.
Betrachten wir nun das „gestaffelte“ Gleichungssystem

$$\begin{array}{ccccccccc} r_{11}x_1 & + & r_{12}x_2 & + & \cdots & + & r_{1n}x_n & = & z_1 \\ & & r_{22}x_2 & + & \cdots & + & r_{2n}x_n & = & z_2 \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & r_{nn}x_n & = & z_n \end{array} \tag{1.5}$$

oder in Matrix-Vektor-Schreibweise

$$Rx = z. \tag{1.6}$$

Offenbar erhalten wir x durch sukzessive Auflösung, beginnend mit der n-ten Zeile:

$$\begin{array}{lll} x_n := z_n/r_{nn} & , \text{ falls} & r_{nn} \neq 0 \\ x_{n-1} := (z_{n-1} - r_{n-1,n}x_n)/r_{n-1,n-1} & , \text{ falls} & r_{n-1,n-1} \neq 0 \\ \vdots \quad\quad\quad\quad \vdots & & \vdots \\ x_k := (z_k - \sum_{i=k+1}^{n} r_{ki}x_i)/r_{kk} & , \text{ falls} & r_{kk} \neq 0 \\ \vdots \quad\quad\quad\quad \vdots & & \vdots \\ x_1 := (z_1 - r_{12}x_2 - \ldots - r_{1,n}x_n)/r_{11} & , \text{ falls} & r_{11} \neq 0. \end{array} \tag{1.7}$$

Für die obere Dreiecksmatrix R gilt, dass

$$\det R = r_{11} \cdot r_{22} \cdot \ldots \cdot r_{nn} \tag{1.8}$$

und daher

$$\det R \neq 0 \Longleftrightarrow r_{ii} \neq 0 \quad (i = 1, \ldots, n). \tag{1.9}$$

Der angegebene Algorithmus ist also wiederum genau dann anwendbar, wenn det $R \neq 0$, also unter der Bedingung des Existenz- und Eindeutigkeitssatzes (Satz 1.1.1).

Analog zum obigen Vorgehen lässt sich auch ein gestaffeltes lineares Gleichungssystem der Form

$$Lx = z$$

mit einer unteren Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$ lösen. In diesem Fall beginnt man in der ersten Zeile mit der Berechnung von x_1 und arbeitet sich dann bis zur letzten Zeile zur Bestimmung von x_n vor.

Bemerkung 1.2.1 (Vorwärts-, Rückwärtssubstitution) *Das Lösen eines LGS mit oberer Dreiecksmatrix nennt man auch Rückwärtseinsetzen/-substitution (Index läuft „rückwärts“ von n nach 1), bzw. mit einer unteren Dreiecksmatrix auch Vorwärtseinsetzen/-substitution (Index läuft „vorwärts“ von 1 nach n).*

Satz 1.2.2 (Rechenaufwand Ab und $A^{-1}b$ (Dreiecksmatrix)) *Sei $A \in \mathbb{R}^{n\times n}$ eine reguläre obere oder untere Dreiecksmatrix und $b \in \mathbb{R}^n$. Dann gilt*

$$\text{FLOP}(Ab) = n^2 \quad \textit{und} \quad \text{FLOP}(A^{-1}b) = n^2\,.$$

Bei der Berechnung von $A^{-1}b$ ist im Gegensatz zu Ab die Reihenfolge, in der die Zeilen „abgearbeitet” werden, festgelegt.

Beweis. Es genügt den Fall einer regulären oberen Dreiecksmatrix $R \in \mathbb{R}^{n\times n}$ zu betrachten. Für den Rechenaufwand zur Lösung von $Rx = b$ (oder die Anwendung von R^{-1} auf einen Vektor b) ergibt sich:

i) für die i-te Zeile: je $(n-i)$ Additionen und Multiplikationen sowie eine Division

ii) insgesamt für die Zeilen n bis 1: Betrachten wir zuerst die Summenformel

$$\sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2-n}{2}\,.$$

Mit i) erhält man insgesamt einen Aufwand von $2\,\frac{n^2-n}{2} + n = n^2$ Operationen.

Der zweite Teil der Aussage bleibt Ihnen als Übungsaufgabe überlassen. □

Aufgabe 1.2.3 Es sei $A \in \mathbb{R}^{n\times n}$ eine obere (oder untere) Dreiecksmatrix und $b \in \mathbb{R}^n$. Man zeige, dass das Matrix-Vektor-Produkt Ab mit n^2 Operationen berechnet werden kann.

Eine Matlab Realisierung zur Bestimmung von $R^{-1}b$, bei der die Matrix R zeilenweise durchlaufen wird, und ein Anwendungsbeispiel sind im Folgenden dargestellt.

MATLAB-Funktion: Rinvb.m

```
function x = Rinvb(R,b)
% compute solution of R * x = b with upper triangular matrix R
n = size(R,1);
x = zeros(n,1);
for j = n:-1:1
  for k=j+1:n
    b(j) = b(j) - R(j,k) * x(k);
  end
  x(j) = b(j) / R(j,j);
end
```

MATLAB-Beispiel:

Man löse $Rx = b$ mit

$$R = \begin{pmatrix} 4 & 1 & 1 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 9 \\ 12 \\ 3 \end{pmatrix}.$$

```
>> R = [4,1,1;0,3,2;0,0,1];
>> b = [9;12;3];
>> x = Rinvb(R,b);
>> x'
ans =
     1     2     3
```

Eine alternative Realisierung, bei der die Matrix R spaltenweise durchlaufen wird, ist mit `Rinvb2.m` gegeben.

MATLAB-Funktion: Rinvb2.m

```
1  function x = Rinvb2(R,b)
2  % compute solution of R * x = b with upper triangular matrix R
3  n = size(R,1);
4  x = zeros(n,1);
5  for j = n:-1:1
6    x(j) = b(j) / R(j,j);
7    for k=1:j-1
8      b(k) = b(k) - R(k,j) * x(j);
9    end
10 end
```

Was ist der Unterschied? Machen Sie sich dies klar!

1.3 GAUSSSCHE ELIMINATIONSMETHODE

Gäbe es nun zu einer beliebigen Matrix A eine Zerlegung

$$A = L \cdot R, \tag{1.10}$$

so könnte ein beliebiges lineares Gleichungssystem $Ax = b$ durch eine Rückwärts- und Vorwärtssubstitution mittels der beiden Schritte

i) löse $Lz = b$,

ii) löse $Rx = z$,

gelöst werden. Die folgende Gaußsche Eliminationsmethode liefert gerade eine solche Zerlegung. Betrachten wir ein allgemeines lineares Gleichungssystem $Ax = b$ ($A \in \mathbb{R}^{n\times n}$ regulär, $b \in \mathbb{R}^n$)

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array} \tag{1.11}$$

und versuchen dies in ein gestaffeltes System umzuformen.

Durch die folgenden drei Äquivalenzoperationen

1. Vertauschung von Zeilen,
2. Multiplikation einer Zeile mit einem Skalar $\neq 0$,
3. Addition eines Vielfachen einer Zeile zu einer anderen,

wird die Lösungsmenge des Gleichungssystems (1.11) nicht verändert.
Wir setzen zunächst $a_{11} \neq 0$ voraus. Um (1.11) nun in ein gestaffeltes System umzuformen, muss die erste Zeile nicht verändert werden. Die restlichen Zeilen werden so modifiziert, dass die Koeffizienten vor x_1 verschwinden, d.h. die Variable x_1 aus den Gleichungen in den Zeilen 2 bis n eliminiert wird.
So entsteht ein System der Art

$$\begin{array}{rcccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}'x_2 & + & \cdots & + & a_{2n}'x_n & = & b_2' \\ & & \vdots & & & & & & \\ & & a_{n2}'x_2 & + & \cdots & + & a_{nn}'x_n & = & b_n'\,. \end{array} \tag{1.12}$$

Haben wir dies erreicht, so können wir dasselbe Verfahren auf die letzten $(n-1)$ Zeilen anwenden und so rekursiv ein gestaffeltes System erhalten. Mit dem Quotienten

$$l_{i1} = a_{i1}/a_{11} \qquad (i = 2, 3, \ldots, n) \tag{1.13}$$

sind die Elemente in (1.12) gegeben durch

$$\begin{aligned} a_{ik}' &= a_{ik} - l_{i1}a_{1k}\,, \\ b_i' &= b_i - l_{i1}b_1. \end{aligned} \tag{1.14}$$

Damit ist der erste Eliminationsschritt unter der Annahme $a_{11} \neq 0$ ausführbar.

Wenden wir auf diese Restmatrix die Eliminationsvorschrift erneut an, so erhalten wir eine Folge

$$A = A^{(1)} \to A^{(2)} \to \ldots A^{(n)} =: R \tag{1.15}$$

von Matrizen der speziellen Gestalt

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & & a_{2n}^{(2)} \\ & & \ddots & & \\ & & & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ & & & \vdots & & \vdots \\ & & & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} \end{pmatrix} \tag{1.16}$$

mit einer $(n-k+1, n-k+1)$-Restmatrix, auf die wir den Eliminationsschritt

$$\boxed{\begin{array}{lcll} l_{ik} & := & a_{ik}^{(k)}/a_{kk}^{(k)} & \text{für} \quad i = k+1, \ldots, n \\ a_{ij}^{(k+1)} & := & a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)} & \text{für} \quad i,j = k+1, \ldots, n \\ b_i^{(k+1)} & := & b_i^{(k)} - l_{ik}b_k^{(k)} & \text{für} \quad i = k+1, \ldots, n \end{array}} \tag{1.17}$$

ausführen können, wenn das **Pivotelement** (Dreh- und Angelpunkt) $a_{kk}^{(k)}$ nicht verschwindet. Da jeder Eliminationsschritt eine lineare Operation auf den Zeilen von A ist, lässt sich der Übergang

von $A^{(k)}$ und $b^{(k)}$ zu $A^{(k+1)}$ und $b^{(k+1)}$ als Multiplikation mit einer Matrix $L_k \in \mathbb{R}^{n\times n}$ von links darstellen, d.h.

$$A^{(k+1)} = L_k A^{(k)}, \quad b^{(k+1)} = L_k b^{(k)}. \tag{1.18}$$

Die Matrix

$$L_k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{n,k} & & 1 \end{pmatrix} = I - \begin{pmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1,k} \\ \vdots \\ l_{n,k} \end{pmatrix} \cdot e_k^T =: I - \ell_k \cdot e_k^T, \tag{1.19}$$

wobei e_k der k-te Einheitsvektor sei, hat die Eigenschaft, dass die Inverse L_k^{-1} aus L_k durch einen Vorzeichenwechsel in den Einträgen l_{ik} $(i > k)$ ensteht und für das Produkt der L_k^{-1} gilt

$$L := L_1^{-1} \cdot \ldots \cdot L_{n-1}^{-1} = \begin{pmatrix} 1 & 0 & \ldots & \ldots & 0 \\ l_{21} & 1 & \ddots & & \vdots \\ l_{31} & l_{32} & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ l_{n1} & \ldots & & l_{n,n-1} & 1 \end{pmatrix}. \tag{1.20}$$

Zusammengefasst erhalten wir auf diese Weise das zu $Ax = b$ äquivalente gestaffelte System $Rx = z$ mit der oberen Dreiecksmatrix

$$R = L^{-1}A \qquad \text{und} \qquad z = L^{-1}b. \tag{1.21}$$

Das **Gaußsche Eliminationsverfahren** schreibt sich dann wie folgt:
Sei $A \in \mathbb{R}^{n\times n}$ und $b \in \mathbb{R}^n$. Berechne nacheinander

i) $L \cdot R = A$ (Zerlegung mit R obere und L untere Dreiecksmatrix),

ii) $Lz = b$ (Vorwärtseinsetzen bzw. -substitution),

iii) $Rx = z$ (Rückwärtseinsetzen bzw. -substitution).

Definition 1.3.1 (unipotente Matrix) *Eine untere oder obere Dreiecksmatrix, deren Diagonalelemente alle gleich eins sind, heißt unipotent.*

Definition 1.3.2 (Gaußsche Dreieckszerlegung, LR-Zerlegung) *Die obige Darstellung $A = L \cdot R$ der Matrix A als Produkt einer unipotenten unteren Dreiecksmatrix L und einer oberen Dreiecksmatrix R heißt* **Gaußsche Dreieckszerlegung** *oder LR-***Zerlegung** *von A.*

Satz 1.3.3 *Existiert die LR-Zerlegung einer quadratischen Matrix A, so sind L und R eindeutig bestimmt.*

Beweis. Siehe Hausübung. □

Satz 1.3.4 (Rechenaufwand Gaußsche Dreieckszerlegung, Gaußsche Elimination)
Sei $A \in \mathbb{R}^{n\times n}$ regulär. Existiert die LR-Zerlegung, so gilt

$$\text{FLOP}(LR - Zerlegung) = \frac{4n^3 - 3n^2 - n}{6}.$$

Des Weiteren sei $b \in \mathbb{R}^n$. Dann gilt für die Gaußsche Elimination

$$\text{FLOP}(A^{-1}b) = \frac{4n^3 + 9n^2 - n}{6},$$

d.h. die Lösung von $Ax = b$ kann mit $(4n^3 + 9n^2 - n)/6$ FLOPs berechnet werden.

Bemerkung 1.3.5 *Die Aufwandsberechnung unterscheidet sich teilweise in der Literatur. Neuerdings werden $+$ und $\cdot$ Operationen nicht mehr unterschieden!*

Beweis von Satz 1.3.4. Für den Rechenaufwand der Gaußschen Dreieckszerlegung gilt Folgendes:

i) für den i-ten Eliminationsschritt:
je $(n-i)$ Divisionen zur Berchnung der ℓ_{ik} und je $(n-i)^2$ Multiplikationen und Subtraktionen zur Berechnung der $a_{jk}^{(i+1)}$, d.h. $2(n-i)^2 + (n-i)$ Operationen

ii) für alle $n-1$ Eliminationsschritte erhalten wir durch Umindizierung und Summenformeln aus Analysis 1 (Aufgabe 1.4.23 in [Analysis I])

$$\begin{aligned} & \sum_{i=1}^{n-1}(n-i) + 2(n-i)^2 = \sum_{i=1}^{n-1} i + 2\,i^2 = \frac{n(n-1)}{2} + \frac{(2n-1)n(n-1)}{3} \\ = \quad & \frac{2(2n-1)(n-1)n + 3n(n-1)}{6} = \frac{n(n-1)(4n-2+3)}{6} \\ = \quad & \frac{n(n-1)(4n+1)}{6} = \frac{4n^3 - 3n^2 - n}{6} \end{aligned}$$

iii) insgesamt benötigt man zum Lösen von $Ax = b$ mittels Gauß-Elimination, d.h. LR-Zerlegung und Vorwärts-, Rückwärtssubstitution (siehe Satz 1.2.2)

$$\frac{4n^3 - 3n^2 - n}{6} + 2n^2 = \frac{4n^3 + 9n^2 - n}{6} \quad \text{Operationen.}$$

Somit sind die Behauptungen des Satzes bewiesen. □

Das Speicherschema für die Gauß-Elimination orientiert sich an der Darstellung von $A^{(k)}$. In die erzeugten Nullen des Eliminationsschritts können die l_{ik} eingetragen werden. Da die Elemente mit den Werten 0 oder 1 natürlich nicht eigens gespeichert werden müssen, kann die LR-Zerlegung mit dem Speicherplatz der Matrix A bewerkstelligt werden:

$$A \to \left(\begin{array}{c|ccc} a_{11}^{(1)} & \dots & \dots & a_{1n}^{(1)} \\ \hline l_{21} & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & & \\ l_{n1} & a_{n2}^{(2)} & & a_{nn}^{(2)} \end{array}\right) \to \dots \to \left(\begin{array}{cccc} a_{11}^{(1)} & \dots & & a_{1n}^{(1)} \\ \hline l_{21} & a_{22}^{(2)} & & a_{2n}^{(2)} \\ \vdots & \ddots & & \\ \vdots & & \ddots & \\ l_{n1} & \dots & l_{n,n-1} & a_{nn}^{(n)} \end{array}\right)$$

$$\Longrightarrow L = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ l_{21} & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & 0 \\ l_{n1} & \dots & \dots & l_{n,n-1} & 1 \end{pmatrix}, \quad R = \begin{pmatrix} a_{11}^{(1)} & \dots & \dots & \dots & a_{1n}^{(1)} \\ & \ddots & & & \vdots \\ & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ & & & & a_{nn}^{(n)} \end{pmatrix}.$$

1.4 PIVOT-STRATEGIEN

Bisher haben wir uns noch nicht mit der Frage beschäftigt, ob eine LR-Zerlegung immer existiert und ob eine Vertauschung von Zeilen bei $Ax = b$ zu einem anderen Ergebnis führt, wenn der Algorithmus mit endlicher Rechengenauigkeit durchgeführt wird. (Bei exakter Arithmetik sind beide Ergebnisse gleich, bzw. Vertauschung von Zeilen in A führt zur gleichen Lösung). Betrachten wir hierzu zwei Beispiele:

Beispiel 1.4.1 i) **Permutation liefert LR-Zerlegung:** Sei $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Wie wir leicht sehen, gilt für die Eigenwerte $\lambda_{1,2} = \pm 1$ und somit $\det A \neq 0$. Falls eine Zerlegung existieren würde, gäbe es $a, b, c, d \in \mathbb{R}$ mit

$$A = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix} \begin{pmatrix} b & c \\ 0 & d \end{pmatrix} = \begin{pmatrix} b & c \\ ab & ac + d \end{pmatrix} .$$

Ein einfacher Vergleich der Koeffizienten zeigt jedoch (zuerst nach b und dann nach a), dass keine LR–Zerlegung existiert, obwohl nach Vertauschen der Zeilen trivialerweise eine LR–Zerlegung sofort gegeben ist. Die Frage, die sich nun stellt, lautet: Kann man für jede reguläre Matrix A durch Vertauschen ihrer Zeilen eine Matrix finden, für die dann eine LR-Zerlegung existiert?

ii) **Permutation liefert höhere Genauigkeit:** Berechnen wir die Lösung des folgenden linearen Gleichungssystems ($\epsilon > 0$)

$$\begin{aligned} \epsilon x_1 + x_2 &= 1 && (1.22) \\ x_1 + x_2 &= 2\,. && (1.23) \end{aligned}$$

Bei exakter Arithmetik erhalten wir

$$\tfrac{1}{\epsilon - 1} \begin{pmatrix} 1 & -1 \\ -1 & \epsilon \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \tfrac{1}{1-\epsilon} \begin{pmatrix} 1 \\ 1 - 2\epsilon \end{pmatrix}$$

d.h. $x_1 = \frac{1}{1-\epsilon}$, $x_2 = \frac{1-2\epsilon}{1-\epsilon}$. Für $2\epsilon <$ Rechengenauigkeit (d.h. $1 \pm 2\epsilon$ und 1 werden im Rechner durch dieselbe Zahl dargestellt) gilt nun

$$x_1 = 1, \quad x_2 = 1.$$

Für die Gauß-Elimination erhalten wir, wenn wir den Koeffizienten vor x_1 in (1.23) eliminieren, d.h. das $1/\epsilon$-fache von (1.22) subtrahieren

$$\begin{aligned} \epsilon x_1 + x_2 &= 1 \\ \left(1 - \tfrac{1}{\epsilon}\right) x_2 &= 2 - \tfrac{1}{\epsilon}\,. \end{aligned}$$

Somit ergibt sich für $2\epsilon <$ Rechengenauigkeit aus der letzten Gleichung $x_2 = 1$ und damit aus der ersten Gleichung $x_1 = 0$. Vertauschen wir jedoch 1. und 2. Zeile (bzw. eliminieren den Koeffizienten vor x_1 in (1.22)) so erhalten wir

$$\begin{aligned} x_1 + x_2 &= 2 \\ (1 - \epsilon) x_2 &= 1 - 2\epsilon\,. \end{aligned}$$

Dies liefert $x_1 = x_2 = 1$ bei $2\epsilon <$ Rechengenauigkeit.

MATLAB-Beispiel:

Dieses scheinbar theoretische Ergebnis aus Beispiel 1.4.1 können wir direkt in Matlab umsetzen. Da die Recheneinheiten heutiger Rechner meist 2 Stellen genauer rechnen als unsere bisherigen theoretischen Untersuchungen vermuten lassen, muss hier $\epsilon \leq$ Maschinengenauigkeit$/8$ gelten.

```
>> e = eps/8;  % eps/2 genügt nicht!
>> x(2) = (2 - 1/e) / (1 - 1/e);
>> x(1) = (1 - x(2)) / e
x =
     0     1
>> x(2) = (1 - 2*e) / (1 - e);
>> x(1) = 2 - x(2)
x =
     1     1
```

Das Vertauschen kann folglich nötig sein. Zum einen, damit eine LR-Zerlegung überhaupt existiert, zum anderen, um den „Rechenfehler“ bedingt durch Rundungsfehler möglichst klein zu halten. Daraus leiten sich besondere Strategien zur Wahl des Pivotelements ab.

Betrachten wir nun die Gauß-Elimination mit **Spaltenpivotstrategie**, welche theoretisch nur dann nicht zielführend sein kann, falls die Matrix A singulär ist.

Algorithmus 1.4.1: Gauß-Elimination mit Spaltenpivotstrategie

a) Wähle im Eliminationsschritt $A^{(k)} \to A^{(k+1)}$ ein $p \in \{k, \ldots, n\}$, so dass

$$|a_{pk}^{(k)}| \geq |a_{jk}^{(k)}| \qquad \text{für } j = k, \ldots, n\,.$$

Die Zeile p soll die Pivotzeile werden.

b) Vertausche die Zeilen p und k

$$A^{(k)} \to \widetilde{A}^{(k)} \quad \text{mit} \quad \widetilde{a}_{ij}^{(k)} = \begin{cases} a_{kj}^{(k)} & \text{falls} \quad i = p \\ a_{pj}^{(k)} & \text{falls} \quad i = k \\ a_{ij}^{(k)} & \text{sonst.} \end{cases}$$

Nun gilt

$$\left|\widetilde{l}_{ik}\right| = \left|\frac{\widetilde{a}_{ik}^{(k)}}{\widetilde{a}_{kk}^{(k)}}\right| = \left|\frac{\widetilde{a}_{ik}^{(k)}}{a_{pk}^{(k)}}\right| \leq 1.$$

c) Führe den nächsten Eliminationsschritt angewandt auf $\widetilde{A}^{(k)}$ aus,

$$\widetilde{A}^{(k)} \to A^{(k+1)}.$$

Bemerkung 1.4.2 *Anstelle der Spaltenpivotstrategie mit Zeilentausch kann man auch eine Zeilenpivotstrategie mit Spaltentausch durchführen. Beide Strategien benötigen im schlimmsten Fall $\mathcal{O}(n^2)$ zusätzliche Operationen. In der Praxis werden im Gegensatz zum oben genannten Algorithmus die Zeile bzw. Spalte nicht umgespeichert, sondern besondere Indexvektoren verwendet (siehe Matlabfunktion* `mylu.m`*). Die Kombination von Spalten- und Zeilenpivotstrategie führt zur vollständigen Pivotsuche, bei der die gesamte Restmatrix nach dem betragsgrößten Eintrag durchsucht wird. Wegen des Aufwands $\mathcal{O}(n^3)$ wird dies jedoch so gut wie nie angewandt.*

MATLAB-Funktion: mylu.m

```
function [L,R,P] = mylu(A)
n = size(A,1);                          % get leading dimension of A
p = 1 : n;                              % pivot element vector
for k = 1 : n-1                         % consider k-th column
  [m,mptr] = max(abs(A(p(k:end),k)));   % find pivot element
  tmp = p(k);                           % interchange in vector p
  p(k) = p(k-1+mptr);
  p(k-1+mptr) = tmp;

  for j = k+1 : n                       % modify entries in
    A(p(j),k) = A(p(j),k)/A(p(k),k);    % compute  l_jk, store in A
    for i = k+1 : n                     % (n-k-1)*(n-k-1) submatrix
      A(p(j),i) = A(p(j),i) ...
                  - A(p(j),k)*A(p(k),i);
    end
  end
end
L = tril(A(p,:),-1)+eye(n);             % these lines could be
R = triu(A(p,:));                       % neglected, all information
P = eye(n);                             % allready in A and p
P(:,p) = P;
```

Satz 1.4.3 *Es sei $A \in \mathbb{R}^{n\times n}$ regulär. Dann existiert vor dem k-ten Eliminationsschritt des Gauß-Algorithmus stets eine Zeilen-/Spaltenpermutation derart, dass das k-te Diagonalelement von Null verschieden ist. Bei Zeilenpermutation, d.h. Spaltenpivotisierung, sind alle Einträge von L vom Betrag kleiner oder gleich eins.*

Beweis. Nach Voraussetzung gilt $\det A \neq 0$. Angenommen, es sei $a_{11} = 0$. Dann existiert eine Zeilenvertauschung, so dass das erste Pivotelement $a_{11}^{(1)}$ von Null verschieden und das betragsgrößte Element in der Spalte ist, d.h.

$$0 \neq \left|a_{11}^{(1)}\right| \geq |a_{i1}^{(1)}| \qquad \text{für} \quad i = 1, \ldots, n,$$

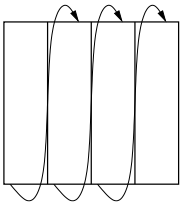
denn andernfalls wäre die Determinante von A in Widerspruch zur Voraussetzung gleich Null. Die Zeilenvertauschung hat dabei nur einen Vorzeichenwechsel in der Determinante zur Folge. Die Überlegungen für den ersten Eliminationsschritt übertragen sich sinngemäß auf die folgenden, reduzierten Systeme, bzw. ihre zugehörigen Determinanten. Diese Schlussfolgerungen gelten analog bei Zeilenpivotisierung. □

Eine unmittelbare Folge aus der Beweisführung des letzten Satzes ist das folgende Lemma.

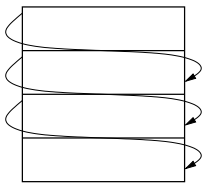
Lemma 1.4.4 *Erfolgen im Verlauf des Gauß-Algorithmus total m Zeilenvertauschungen (Spaltenvertauschungen), so ist die Determinante von A gegeben durch*

$$\det A = (-1)^m \prod_{k=1}^{n} r_{kk}.$$

Bemerkung 1.4.5 *Numerisch sind Spalten- und Zeilenpivotisierungen äquivalent. Die Auswahl hängt von der Speichermethode der Matrix A ab, d.h. man favorisiert die Spaltenpivotisierung, wenn die Einträge der Matrix spaltenweise im Speicher abgelegt sind, z.B. bei den Programmiersprachen Matlab, Fortran und die Zeilenpivotisierung u.a. bei C, denn hier sind die Matrizen als Folge von Zeilenvektoren abgelegt.*



Spaltenpivotisierung, bzw.



Zeilenpivotisierung.

Die genannten Pivotisierungen bzw. Zeilenvertauschungen lassen sich formal durch eine geeignete Permutationsmatrix P beschreiben. Diese ist eine quadratische Matrix, welche in jeder Zeile und in jeder Spalte genau eine Eins und sonst Nullen enthält. Die Determinante ist $\det P = \pm 1$. Aus dem letzten Satz ergibt sich folgendes Resultat.

Satz 1.4.6 (Existenz einer Zerlegung $PA = LR$) *Zu jeder regulären Matrix A existiert eine Permutationsmatrix P, so dass $P \cdot A$ in ein Produkt $L \cdot R$ zerlegbar ist, d.h.*

$$P \cdot A = L \cdot R\,.$$

Ist nun immer eine Pivotisierung notwendig? Wir nennen im Folgenden ein Kriterium, welches leicht zu überprüfen ist.

Definition 1.4.7 (Diagonaldominanz) *Eine Matrix heißt diagonaldominant, falls gilt*

$$|a_{ii}| \geq \sum_{k=1,\, k\neq i}^{n} |a_{ik}| \qquad \forall\, i = 1, \ldots, n\,.$$

Sie heißt strikt diagonaldominant, falls für alle i „$>$" anstatt „$\geq$" gilt.

Satz 1.4.8 *Es sei $A \in \mathbb{R}^{n\times n}$ eine diagonaldominante und reguläre Matrix. Dann existiert eine Zerlegung*

$$A = L \cdot R.$$

Beweis. Einen Beweis dieser Aussage (in leicht abgewandelter Form) findet man z.B. bei [Schwarz]. Nach Voraussetzung ist entweder $|a_{11}| \geq \sum_{k=2}^{n} |a_{1k}| > 0$ oder es gilt $\sum_{k=2}^{n} |a_{1k}| = 0$ und $|a_{11}| > 0$; dies folgt aus der Regularität von A. Somit ist $a_{11} \neq 0$ ein zulässiges Pivotelement für den ersten Eliminationsschritt. Man muss nun zeigen, dass sich die Eigenschaft der diagonalen Dominanz auf das reduzierte Gleichungssystem überträgt. □

Zum Schluss dieses Abschnitts betrachten wir noch folgendes Beispiel zur konkreten Berechnung einer LR-Zerlegung einer Matrix A.

Beispiel 1.4.9 (Berechnung von $PA = LR$) Sei

$$A = \begin{pmatrix} 0 & 2 & -1 & -2 \\ 2 & -2 & 4 & -1 \\ 1 & 1 & 1 & 1 \\ -2 & 1 & -2 & 1 \end{pmatrix}.$$

Diese Matrix ist offensichtlich nicht diagonaldominant, also wenden wir die Gauß-Elimination mit Spaltenpivotisierung an. Hierzu sei $P(i,j)$ diejenige Permutationsmatrix, die aus der Einheitsmatrix durch Vertauschen der i-ten und j-ten Zeile entsteht. Das Vertauschen der ersten mit der zweiten Zeile von A entspricht der Multiplikation der Matrix $P_1 := P(1,2)$ mit A von links, also

$$\widetilde{A}^{(1)} = P_1 A = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 1 & 1 & 1 & 1 \\ -2 & 1 & -2 & 1 \end{pmatrix}.$$

Der erste Schritt der LR-Zerlegung liefert

$$L_1 = \begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ -\frac{1}{2} & & 1 & \\ 1 & & & 1 \end{pmatrix}, \qquad L_1 P_1 A = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 0 & 2 & -1 & \frac{3}{2} \\ 0 & -1 & 2 & 0 \end{pmatrix}.$$

Bei der nächsten Spaltenpivotisierung sind keine Zeilenvertauschungen notwendig, also $P_2 = I$. Es folgt

$$L_2 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & -1 & 1 & \\ & \frac{1}{2} & & 1 \end{pmatrix}, \qquad L_2 P_2 L_1 P_1 A = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 0 & 0 & 0 & \frac{7}{2} \\ 0 & 0 & \frac{3}{2} & -1 \end{pmatrix}.$$

Um nun auf eine obere Dreiecksgestalt zu kommen, müssen lediglich noch die dritte und vierte Zeile vertauscht werden, also formal $P_3 = P(3,4)$, $L_3 = I$ und damit

$$L_3 P_3 L_2 P_2 L_1 P_1 A = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 0 & 0 & \frac{3}{2} & -1 \\ 0 & 0 & 0 & \frac{7}{2} \end{pmatrix} =: R\,.$$

Aber wie sehen nun P und L aus, sodass

$$P \cdot A = L \cdot R \quad ?$$

Wir betrachten hierzu im allgemeinen Fall mit $A \in \mathbb{R}^{n\times n}$ invertierbar für $k = 1, \ldots, n-1$ die Matrizen

$$\widetilde{L}_k := P_n \cdot \ldots \cdot P_{k+1} L_k P_{k+1} \cdot \ldots \cdot P_n\,,$$

wobei $P_n := I$ gesetzt wird und $P_1, \ldots, P_{n-1}$ wie in Beispiel 1.4.9 gewählt werden. Dann gilt

$$\begin{aligned} \widetilde{L}_k &= P_n \cdot \ldots \cdot P_{k+1} L_k P_{k+1} \cdot \ldots \cdot P_n \\ &\overset{(1.19)}{=} P_n \cdot \ldots \cdot P_{k+1} (I - \ell_k e_k^T) P_{k+1} \cdot \ldots \cdot P_n \\ &= I - \underbrace{P_n \cdot \ldots \cdot P_{k+1} \ell_k}_{=:\widetilde{\ell}_k} \underbrace{e_k^T P_{k+1} \cdot \ldots \cdot P_n}_{=e_k^T} \\ &= I - \widetilde{\ell}_k e_k^T\,. \end{aligned}$$

Da die Multiplikation mit den Matrizen $P_{k+1}, \ldots, P_n$ von links bzw. rechts nur Zeilen- bzw. Spaltenvertauschungen innerhalb der Zeilen bzw. Spalten $k+1, \ldots, n$ bewirkt, besitzt die Matrix $\widetilde{L}_k$ dieselbe Struktur wie L_k. Aus der Definition der $\widetilde{L}_k$ folgt nun

$$\widetilde{L}_{n-1} \cdot \ldots \cdot \widetilde{L}_1 P_{n-1} \cdot \ldots \cdot P_1 = L_{n-1} P_{n-1} \cdot \ldots \cdot L_2 P_2 L_1 P_1 \,.$$

Nach Konstruktion der L_k und P_k gilt schließlich

$$\widetilde{L}_{n-1} \cdot \ldots \cdot \widetilde{L}_1 P_{n-1} \cdot \ldots \cdot P_1 A = R \,.$$

Somit ergibt sich mit

$$L := \left(\widetilde{L}_{n-1} \cdot \ldots \cdot \widetilde{L}_1\right)^{-1} = \widetilde{L}_1^{-1} \cdot \ldots \cdot \widetilde{L}_{n-1}^{-1}$$

und

$$P := P_{n-1} \cdot \ldots \cdot P_1$$

eine Zerlegung $PA = LR$.

Bemerkung 1.4.10 *Man beachte, dass mit den obigen Überlegungen ein alternativer, konstruktiver Beweis des Satzes 1.4.6 vorliegt.*

Führen wir nun Beispiel 1.4.9 weiter fort:

Beispiel 1.4.11 Zu der Matrix A aus Beispiel 1.4.9 sollen die Matrizen L und P der Zerlegung $PA = LR$ bestimmt werden. Wir erhalten

$$P = P_3 P_2 P_1 = P(1,2)\, I\, P(3,4) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

$$\widetilde{L}_1 = P_3 P_2 L_1 P_2 P_3 = P(3,4) \begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ -\frac{1}{2} & & 1 & \\ 1 & & & 1 \end{pmatrix} P(3,4) = \begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ 1 & & 1 & \\ -\frac{1}{2} & & & 1 \end{pmatrix},$$

$$\widetilde{L}_2 = P_3 L_2 P_3 = P(3,4) \begin{pmatrix} 1 & & & \\ & 1 & & \\ & -1 & 1 & \\ & \frac{1}{2} & & 1 \end{pmatrix} P(3,4) = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & \frac{1}{2} & 1 & \\ & -1 & & 1 \end{pmatrix},$$

$$\widetilde{L}_3 = P_4 L_3 P_4 = I \,,$$

$$L = \widetilde{L}_1^{-1} \widetilde{L}_2^{-1} \widetilde{L}_3^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -\frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 1 & 0 & 1 \end{pmatrix}$$

und somit

$$PA = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ -2 & 1 & -2 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -\frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 0 & 0 & \frac{3}{2} & -1 \\ 0 & 0 & 0 & \frac{7}{2} \end{pmatrix} = LR \,.$$

1.5 NACHITERATION

Die oben diskutierten Pivotstrategien schließen offenbar nicht aus, dass die so berechnete Lösung x immer noch „ziemlich ungenau" ist. Wie kann man nun x ohne großen Aufwand verbessern?

Häufig lässt sich dies durch eine **Nachiteration** mit Hilfe einer expliziten Auswertung des **Residuums** $r := b - A\widetilde{x}$ erreichen. Dabei bezeichne $\widetilde{x}$ die durch ein numerisches Verfahren berechnete Näherung an x. Ausgehend von $\widetilde{x}$ soll die exakte Lösung mittels des Korrekturansatzes

$$x = \widetilde{x} + s$$

ermittelt werden. Der Korrekturvektor ist so zu bestimmen, dass die Gleichungen erfüllt sind, d.h.

$$Ax - b = A(\widetilde{x} + s) - b = A\widetilde{x} + As - b = 0.$$

Der Korrekturvektor s ergibt sich somit als Lösung von

$$As = b - A\widetilde{x} = r.$$

Bei der numerischen Lösung dieser Korrekturgleichung erhalten wir im Allgemeinen eine wiederum fehlerhafte Korrektur $\widetilde{s} \neq s$. Trotzdem erwarten wir, dass die Näherungslösung

$$\widetilde{x} + \widetilde{s}$$

„besser" ist als $\widetilde{x}$.

Bemerkung 1.5.1 *Wie wir in Kapitel 3 zeigen werden, genügt bei Spalten- oder Zeilenpivotisierung eine einzige Nachiteration, um eine Lösung auf eine dem Problem angepasste Genauigkeit zu erhalten.*

1.6 CHOLESKY-VERFAHREN

Wir wollen nun die Gauß-Elimination auf die eingeschränkte Klasse von Gleichungssystemen mit symmetrisch positiv definiten Matrizen anwenden. Es wird sich herausstellen, dass die Dreieckszerlegung in diesem Fall stark vereinfacht werden kann und dass dieser vereinfachte Algorithmus für große Matrizen nur den halben Aufwand an Operationen benötigt. Wir erinnern hier nochmals an die

Definition 1.6.1 *Eine symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ heißt genau dann positiv definit, wenn $x^T Ax > 0$ für alle $x \in \mathbb{R}^n$ mit $x \neq 0$ ist.*

Positiv definite Matrizen haben folgende Eigenschaften.

Satz 1.6.2 (Eigenschaften positiv definiter Matrizen) *Für jede positiv definite Matrix $A \in \mathbb{R}^{n\times n}$ gilt:*

i) A *ist invertierbar,*

ii) $a_{ii} > 0 \qquad$ *für* $\quad i = 1, \ldots, n\,,$

iii) $\max\limits_{i,j=1,\ldots,n} |a_{ij}| = \max\limits_{i=1,\ldots,n} a_{ii}\,,$

iv) Bei der Gauß-Elimination ohne Pivotsuche ist jede Restmatrix wiederum positiv definit.

Beweis. Wäre A nicht invertierbar, gäbe es einen Eigenwert $\lambda = 0$, da $\det A = 0$ nach Annahme. Dies steht aber im Widerspruch zur Voraussetzung, dass A positiv definit ist, da es ansonsten einen Eigenvektor $x \neq 0$ zu $\lambda = 0$ mit $Ax = \lambda x$ gäbe und somit dann auch $x^T Ax = 0$ gälte. Somit ist i) bewiesen. Setzt man für x einen kanonischen Einheitsvektor e_i ein, so folgt gerade $a_{ii} = e_i^T A e_i > 0$ und daher gilt die Aussage ii). Behauptung iii) sei als Hausübung überlassen. Um die verbleibende Behauptung iv) zu beweisen, schreiben wir $A = A^{(1)}$ in der Form

$$A^{(1)} = \left(\begin{array}{c|c} a_{11} & z^T \\ \hline z & B^{(1)} \end{array}\right),$$

wobei $z = (a_{12}, \ldots, a_{1n})^T$ sei. Nach einem Eliminationsschritt ergibt sich

$$A^{(2)} = L_1 A^{(1)} = \left(\begin{array}{c|c} a_{11} & z^T \\ \hline 0 & \\ \vdots & B^{(2)} \\ 0 & \end{array}\right) \quad \text{mit} \quad L_1 = \begin{pmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{n1} & & & 1 \end{pmatrix}.$$

Multipliziert man nun $A^{(2)}$ von rechts mit L_1^T, so wird auch z^T in der ersten Zeile eliminiert und die Teilmatrix $B^{(2)}$ bleibt unverändert, d.h.

$$L_1 A^{(1)} L_1^T = \left(\begin{array}{c|ccc} a_{11} & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & B^{(2)} & \\ 0 & & & \end{array}\right).$$

Damit ist bewiesen, dass die Restmatrix $B^{(2)}$ symmetrisch ist. Können wir nun noch zeigen, dass $B^{(2)}$ auch wiederum positiv definit ist, so können wir unsere Argumentation sukzessive fortsetzen. Es sei $y \in \mathbb{R}^{n-1}$ mit $y \neq 0$. Da L_1 regulär ist, gilt $x^T := \big(0 \,\vdots\, y^T\big) L_1 \neq 0$. Nach Voraussetzung ist A positiv definit, also gilt

$$0 < x^T Ax = \big(0 \,\vdots\, y^T\big) L_1 A L_1^T \begin{pmatrix} 0 \\ \cdots \\ y \end{pmatrix} = \big(0 \,\vdots\, y^T\big) \left(\begin{array}{c|c} a_{11} & 0 \\ \hline 0 & B^{(2)} \end{array}\right) \begin{pmatrix} 0 \\ \cdots \\ y \end{pmatrix} = y^T B^{(2)} y\,.$$

Somit haben wir gezeigt, dass auch $B^{(2)}$ wieder positiv definit ist. □

Aufgabe 1.6.3 Man beweise Aussage iii) in Satz 1.6.2.

Mit Hilfe des letzten Satzes über die LR-Zerlegung können wir jetzt die **rationale** Cholesky-Zerlegung für symmetrische, positiv definite Matrizen herleiten.

Satz 1.6.4 (rationale Cholesky-Zerlegung) *Für jede* **symmetrische***, positiv definite Matrix A existiert eine eindeutig bestimmte Zerlegung der Form*

$$A = LDL^T,$$

wobei L eine unipotente untere Dreiecksmatrix und D eine positive Diagonalmatrix ist.

Beweis. Wir setzen die Konstruktion im Beweis von Satz 1.6.2.iv für $k = 2, \ldots, n-1$ fort und erhalten so unmittelbar L als Produkt der $L_1^{-1}, \ldots, L_{n-1}^{-1}$ und D als Diagonalmatrix der Pivotelemente. □

Bemerkung 1.6.5 *Man beachte die Eigenschaften der L_k bei der wirklichen Berechnung von L, d.h. das Produkt $L_1^{-1} \cdot \ldots \cdot L_{n-1}^{-1}$ ist nicht wirklich durch Multiplikation auszurechnen (vgl.* (1.20)*).*

Bemerkung 1.6.6 *Da alle Einträge der Diagonalmatrix D positiv sind, existiert $D^{1/2} = \mathrm{diag}(\pm\sqrt{d_i})$ und daher die Cholesky-Zerlegung*

$$A = \overline{L}\,\overline{L}^T,$$

wobei $\overline{L}$ die untere Dreiecksmatrix $\overline{L} := LD^{1/2}$ ist.

Bemerkung 1.6.7 *Die Cholesky-Zerlegung ist nicht eindeutig, da $D^{1/2}$ nur bis auf Vorzeichen der Elemente in der Diagonalen eindeutig bestimmt ist und somit auch $\overline{L} = LD^{1/2}$ nicht eindeutig ist.*

Zur Herleitung des Algorithmus zur Berechnung einer Cholesky-Zerlegung betrachten wir einfach die Gleichung

$$\begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & & \ddots & \\ l_{n1} & \cdots & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & & \vdots \\ & & \ddots & \vdots \\ & & & l_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & & \vdots \\ \vdots & & & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix} =$$

$$\begin{pmatrix} l_{11}^2 & l_{11}l_{21} & l_{11}l_{31} & \cdots & l_{11}l_{n1} \\ l_{11}l_{21} & l_{21}^2 + l_{22}^2 & l_{21}l_{31} + l_{22}l_{32} & \cdots & l_{21}l_{n1} + l_{22}l_{n2} \\ l_{11}l_{31} & l_{21}l_{31} + l_{22}l_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 & \cdots & l_{31}l_{n1} + l_{32}l_{n2} + l_{33}l_{n3} \\ \vdots & \vdots & \vdots & & \vdots \\ l_{11}l_{n1} & l_{21}l_{n1} + l_{22}l_{n2} & l_{31}l_{n1} + l_{32}l_{n2} + l_{33}l_{n3} & \cdots & \sum_{k=1}^{n} l_{nk}^2 \end{pmatrix}$$

d.h.

$$\text{für } i = k \qquad \text{gilt } a_{kk} = \sum_{j=1}^{k} l_{kj}^2 \quad \text{und}$$

$$\text{für } i \neq k \qquad \text{gilt } a_{ik} = \sum_{j=1}^{k} l_{ij} \cdot l_{kj} \qquad (k+1 \leq i \leq n)$$

und werten diese in einer geschickten Reihenfolge aus.

Cholesky-Verfahren zur Berechnung von L mit $A = LL^T$
Spaltenweise berechnet man für $k = 1, 2, \ldots, n$

$$\begin{aligned} l_{kk} &= \Big(a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2\Big)^{1/2} \\ l_{ik} &= \frac{1}{l_{kk}}\Big(a_{ik} - \sum_{j=1}^{k-1} l_{ij} \cdot l_{kj}\Big) \qquad (k+1 \leq i \leq n). \end{aligned}$$

Bemerkung 1.6.8 *i) Aus der Cholesky-Zerlegung $A = L \cdot L^T$ ergibt sich für alle $1 \le k \le n$ die Abschätzung*

$$\sum_{j=1}^{k} l_{kj}^2 \le \max_{1\le j\le n} |a_{jj}|.$$

Folglich sind alle Elemente der Matrix L betragsweise durch $\max_{1\le j\le n} \sqrt{|a_{jj}|}$ beschränkt. Die Elemente können damit nicht allzu stark anwachsen, was sich günstig auf die Stabilität des Verfahrens auswirkt.

ii) Da A symmetrisch ist, wird nur Information oberhalb und einschließlich der Hauptdiagonalen benötigt. Wenn man die Diagonalelemente l_{kk} separat in einem Vektor der Länge n speichert, und die Elemente $l_{jk},\ k < j$ unterhalb der Diagonale, so kann man die Information der Matrix A bewahren.

iii) Bei der algorithmischen Durchführung der Cholesky-Zerlegung liefert das Verfahren auch die Information, ob die Matrix positiv definit ist. Man mache sich dies als Übungaufgabe klar!

Aufgabe 1.6.9 Man beweise Aussage iii) in Bemerkung 1.6.8.

Satz 1.6.10 (Rechenaufwand Cholesky-Zerlegung) *Sei $A \in \mathbb{R}^{n\times n}$ positiv definit. Dann gilt*

$$\text{FLOP}(\textit{Cholesky-Zerl. von } A) = \frac{2n^3+3n^2-5n}{6} + n\,\text{Wurzelberechnung} = \frac{1}{3}n^3 + \mathcal{O}(n^2)\,.$$

Beweis. Untersuchen wir nun die Komplexität der Cholesky-Zerlegung zuerst für einen k-ten Zerlegungsschritt und betimmen dann den Gesamtaufwand.

i) Für den k-ten Zerlegungsschritt, d.h. die Berechnung der Elemente l_{ik} für festen Spaltenindex, sind jeweils $(k-1)$ Multiplikationen, $(k-1)$ Subtraktionen und eine Wurzelberechnung zur Berechnung des Diagonalelements nötig. Für jedes Nebendiagonalelement werden $(k-1)$ Multiplikationen, $(k-1)$ Subtraktionen und eine Division benötigt, d.h. bei $(n-k)$ Elementen unterhalb des k-ten Diagonalelements sind dies in der Summe

$$\begin{aligned} & \big(2(k-1)+(n-k)(2k-1)\big)\text{FLOP und 1 Wurzelberechnung} \\ = \ & \big(-2k^2+(3+2n)k-(n+2)\big)\text{FLOP und 1 Wurzelberechnung} \end{aligned}$$

ii) Insgesamt ergibt sich dann für die Summe über alle Zerlegungschritte

$$\begin{aligned} & n\,\text{Wurzelberechnungen} + \sum_{k=1}^{n}\big(-2k^2+(3+2n)k-(n+2)\big) \\ = \ & n\,\text{Wurzelberechnungen} + \Big(-2\frac{(2n+1)(n+1)n}{6} + (3+2n)\frac{(n+1)n}{2} - (n+2)n\Big) \\ = \ & n\,\text{Wurzelberechnungen} + \Big(\frac{-(4n^3+6n^2+2n)+(6n^3+15n^2+9n)-(6n^2+12n)}{6}\Big) \\ = \ & n\,\text{Wurzelberechnungen} + \frac{2n^3+3n^2-5n}{6}\,. \end{aligned}$$

Da eine einzelne Wurzelberechnung an Aufwand ein konstantes Vielfaches einer Gleitkommaoperation benötigt, kann man den Term n Wurzelberechnungen $+(3n^2-5n)/6$ zu $\mathcal{O}(n^2)$ Operationen zusammenfassen. □

Bemerkung 1.6.11 *Für große n ist der Aufwand des Cholesky-Verfahrens im Vergleich zum Gauß-Algorithmus ungefähr halb so groß.*

1.7 BANDGLEICHUNGEN

Eine weitere Klasse spezieller Matrizen, welche beim Lösen linearer Gleichungssysteme eine besondere Rolle spielen, sind Bandmatrizen. Man spricht von einer Bandmatrix A, falls alle von Null verschiedenen Elemente a_{ik} in der Diagonale und in einigen dazu benachbarten Nebendiagonalen stehen. Für die Anwendungen sind insbesondere die symmetrischen, positiv definiten Bandmatrizen wichtig.

Definition 1.7.1 *Unter der* **Bandbreite** m *einer symmetrischen Matrix* $A \in \mathbb{R}^{n\times n}$ *versteht man die kleinste natürliche Zahl* $m < n$*, so dass gilt*

$$a_{ik} = 0 \qquad \text{für alle } i \text{ und } k \text{ mit } |i-k| > m.$$

Bemerkung 1.7.2 *Die Bandbreite* m *gibt somit die Anzahl der Nebendiagonalen unterhalb bzw. oberhalb der Diagonalen an, welche die im Allgemeinen von Null verschiedenen Matrixelemente enthalten.*

Bemerkung 1.7.3 *In verallgemeinerter Form spricht man auch von unterer und oberer Bandbreite, welche sich auf suggestive Weise definieren.*

Satz 1.7.4 *Die untere Dreiecksmatrix* L *der Cholesky-Zerlegung* $A = LL^T$ *einer symmetrischen, positiv definiten Bandmatrix mit der Bandbreite* m *besitzt dieselbe Bandstruktur, denn es gilt*

$$l_{ik} = 0 \qquad \text{für alle } i,k \quad \text{mit } i-k > m\,.$$

Beweis. Ergibt sich direkt aus einer Hausübung zur Hülle einer Matrix. □

Aufgabe 1.7.5 Man beweise Satz 1.7.4.

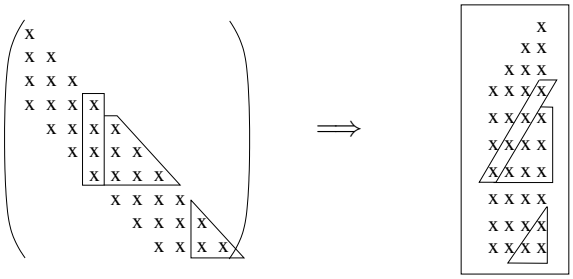


Abb. 1.1: Reduktion und Speicherung einer symmetrischen, positiv definiten Bandmatrix.

Analog zur Herleitung des Cholesky-Verfahrens auf Seite 19 erhalten wir die Rechenvorschrift des Cholesky-Verfahrens für Bandmatrizen wobei nun die Koeffizienten zeilenweise bestimmt werden. Für $n = 5$ und $m = 2$ sieht die Situation wie folgt aus:

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ 0 & l_{42} & l_{43} & l_{44} & 0 \\ 0 & 0 & l_{53} & l_{54} & l_{55} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} & 0 & 0 \\ 0 & l_{22} & l_{32} & l_{42} & 0 \\ 0 & 0 & l_{33} & l_{43} & l_{53} \\ 0 & 0 & 0 & l_{44} & l_{54} \\ 0 & 0 & 0 & 0 & l_{55} \end{pmatrix}$$

$$= \begin{pmatrix} l_{11}^2 & & \text{sym.} & & \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 & & & \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 & & \\ & l_{42}l_{22} & l_{42}l_{32} + l_{43}l_{33} & l_{42}^2 + l_{43}^2 + l_{44}^2 & \\ & & l_{53}l_{33} & l_{53}l_{43} + l_{54}l_{44} & l_{53}^2 + l_{54}^2 + l_{55}^2 \end{pmatrix}.$$

Cholesky-Verfahren zur Berechnung von L mit $A = LL^T$, wobei $A \in \mathbb{R}^{n\times n}$
mit Bandbreite $0 \le m < n$.
Zeilenweise berechnet man für $k = 1, 2, \ldots, n$

$$\begin{aligned} l_{ki} &= \frac{1}{l_{ii}}\Big(a_{ki} - \sum_{j=\max\{1,k-m\}}^{i-1} l_{kj}\cdot l_{ij}\Big) \qquad (\max\{1, k-m\} \le i \le k-1) \\ l_{kk} &= \Big(a_{kk} - \sum_{j=\max\{1,k-m\}}^{k-1} l_{kj}^2\Big)^{1/2} \end{aligned}$$

Satz 1.7.6 *Es sei $A \in \mathbb{R}^{n\times n}$ eine positiv definite Bandmatrix mit Bandbreite $1 \le m \le n$. Dann beträgt der Aufwand zur Berechnung der Cholesky-Zerlegung von A*

$$\text{FLOP}(\textit{Cholesky-Zerl. von } A) = n(m^2+2m) - \frac{4m^3+9m^2+5m}{6} + n \text{ Wurzelberechnungen}\,.$$

Beweis. Wendet man die Cholesky-Zerlegung auf eine positiv definite Bandmatrix A an, so ist die resultierende untere Dreiecksmatrix L mit $A = LL^T$ wieder eine Bandmatrix mit der gleichen Bandbreite, wie A sie hat.

i) Gehen wir davon aus, dass die Matrix L zeilenweise berechnet wird und betrachten zuerst den Aufwand für die ersten m Zeilen. Hier kann man das Ergebnis aus Satz 1.6.10 verwenden. Es sind hierfür somit

$$m \text{ Wurzelberechnungen} + \frac{2m^3+3m^2-5m}{6}$$

Operationen notwendig.

ii) In den verbleibenden $n-m$ Zeilen, in denen jeweils $m+1$ Koeffizienten zu bestimmen sind, ergibt sich Folgendes:
Um den j-ten von Null verschiedenen Nebendiagonaleintrag von L in der k-ten Zeile $m+1 \le k \le n$ zu berechnen, sind jeweils $j-1$ Multiplikationen, $j-1$ Subtraktionen und eine Division notwendig. Zur Berechnung des Diagonalelements werden m Multiplikationen, m Subtraktionen und eine Wurzelberechnung benötigt. Bei m Nebendiagonalelementen sind dies in der Summe

$$\begin{aligned} & 1 \text{ Wurzelberechnung} + 2m + \sum_{j=1}^{m}(2j-1) = 1\sqrt{\cdot} + m + 2\sum_{j=1}^{m} j \\ = \;& 1\sqrt{\cdot} + m + m(m+1) = 1\sqrt{\cdot} + m(m+2)\,. \end{aligned}$$

iii) Der gesamte Aufwand beträgt also

$$\begin{aligned} & n \text{ Wurzelberechnungen} + \frac{2m^3+3m^2-5m}{6} + (n-m)m(m+2) \\ = \;& n \text{ Wurzelberechnungen} + n(m^2+2m) - \frac{4m^3+9m^2+5m}{6}\,. \end{aligned}$$

□

Bemerkung 1.7.7 *Ist für eine Klasse von positiv definiten Bandmatrizen aus $\mathbb{R}^{n\times n}$ für beliebiges n die Bandbreite beschränkt, so wächst der Aufwand zur Berechnung der Cholesky-Zerlegung asymptotisch nur* ***linear*** *in n.*

1.8 VANDERMOND-MATRIZEN

Bei vielen Approximations- und Interpolationsproblemen treten sogenannte Vandermond-Matrizen auf, d.h. Matrizen $V \in \mathbb{R}^{(n+1)\times(n+1)}$ von der Form

$$V = \quad V(x_0, \ldots, x_n) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \cdots & & & \\ x_0^n & x_1^n & \cdots & x_n^n \end{pmatrix}.$$

Sei z.B. zu $(n+1)$-Daten (x_i, f_i) ein Polynom n-ten Grades $p \in \mathbb{P}_n$ gesucht mit der Eigenschaft $p(x_i) = f_i$ $(i = 0, \ldots, n)$. Wählt man den Ansatz

$$p(x) = \sum_{j=0}^{n} a_j x^j \,,$$

so führt dies zu einem LGS

$$V^T a = f$$

mit $a = (a_0, \ldots, a_n)^T$ und $f = (f_0, \ldots, f_n)^T$. Man mache sich dies an einem einfachen Beispiel klar!

Diesen Zusammenhang zwischen dem Interpolationsproblem und dem System $V^T a = f$ macht man sich auch beim Lösen zu Nutze. Die Bestimmung des Koeffizientenvektors a bzw. das Lösen des Gleichungssystems gliedert sich dabei in zwei Teile.

- Zuerst bestimmt man das Interpolationspolynom p in der Newtondarstellung, d.h.

$$\begin{aligned} p(x) &= \sum_{k=0}^{n} c_k \left(\prod_{j=0}^{k-1} (x - x_j) \right) \\ &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_n(x - x_0) \cdots (x - x_{n-1}) \end{aligned}$$

  Die Konstanten c_k werden als dividierte Differenzen bezeichnet, sie lösen das lineare Gleichungsystem

$$\begin{pmatrix} 1 & & & & \\ 1 & (x_1 - x_0) & & & \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & & \\ \vdots & & & \ddots & \\ 1 & (x_n - x_0) & (x_n - x_0)(x_n - x_1) & \ldots & \prod_{j=0}^{n-1}(x_n - x_j) \end{pmatrix} \begin{pmatrix} c_0 \\ \\ \vdots \\ \\ c_n \end{pmatrix} = \begin{pmatrix} f_0 \\ \\ \vdots \\ \\ f_n \end{pmatrix}$$

  und können wie folgt bestimmt werden.

```
c(0:n) = f(0:n)
for k = 0 : n − 1
    for i = n : −1 : k + 1
        c_i = (c_i − c_{i−1})/(x_i − x_{i−k−1})
    end
end
```
(1.24)

- Im zweiten Schritt gewinnt man nun die a_i's aus den c_i's durch eine Basistransformation von $(1,(x-x_0),(x-x_0)(x-x_1),\ldots,\prod_{j=0}^{n-1}(x-x_j))$ nach $(1,x,x^2,\ldots,x^n)$. Die Transformationsmatrix U ergibt sich aus entsprechendem Koeffizientenvergleich

$$\begin{pmatrix} 1 & -x_0 & x_0x_1 & \ldots & \prod_{j=0}^{n-1}(-x_j) \\ & 1 & -(x_0+x_1) & & \\ & & 1 & & \vdots \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} c = a\,. \tag{1.25}$$

  Anstatt die Matrix U zu bestimmen, kann a wie folgt bestimmt werden.

```
a(0:n) = c(0:n)
for k = n − 1 : −1 : 0
    for i = k : n − 1
        a_i = a_i − x_k a_{i+1}
    end
end
```
(1.26)

Die Kombination beider Schritte liefert den folgenden Algorithmus.

Algorithmus 1.8.1: Polynominterpolation: $V^T a = f$

Gegeben seien $x(0:n) \in \mathbb{R}^{n+1}$ mit paarweise verschiedenen Einträgen und $f = f(0:n) \in \mathbb{R}^{n+1}$. Der folgende Algorithmus überschreibt f mit der Lösung $a = a(0:n)$ zu dem Vandermondschen System

$$V(x_0,\ldots,x_n)^T a = f$$

```
for k = 0 : n − 1
    for i = n : −1 : k + 1
        f(i) = (f(i) − f(i − 1))/(x(i) − x(i − k − 1))
    end
end
for k = n − 1 : −1 : 0
    for i = k : n − 1
        f(i) = f(i) − f(i + 1)x(k)
    end
end
```

Satz 1.8.1 *Das Verfahren 1.8.1 benötigt zum Lösen des Vandermondschen System*

$$V(x_0,\ldots,x_n)^T a = f$$

nur $5(n^2+n)/2$ *Operationen.*

Beweis. Die Anweisung in der ersten inneren FOR-Schleife von Algorithmus 1.8.1 wird für ein $k \in \{0,\ldots,n-1\}$ $(n-k)$-mal ausgeführt, insgesamt also $\sum_{k=0}^{n-1}(n-k) = \sum_{k=1}^{n} k = n(n+1)/2$

-mal. Hierbei werden jedesmal 3 Gleitkommaoperationen benötigt. Für die zweite Doppelschleife ergeben sich, analog zu der obigen Betrachtung, $n(n+1)/2$ Aufrufe der inneren Anweisung, für die jeweils 2 Gleitkommaoperationen benötigt werden. □

Bemerkung 1.8.2 *Man beachte, dass für spezielle vollbesetzte Matrizen Verfahren existieren, die weniger als $\mathcal{O}(n^3)$ Operationen zur Berechnung der Lösung benötigen. Auch die Matrix braucht nicht vollständig gespeichert werden.*

1.8.1 Das System $Vz = b$

Der Vollständigkeit wegen sei hier auch ein Verfahren für $Vz = b$ aufgeführt.
Es sei $L_k(\alpha) \in \mathbb{R}^{(n+1)\times(n+1)}$ eine untere bidiagonale Matrix definiert durch

$$L_k(\alpha) = \left(\begin{array}{c|ccccccc} I_k & & & & 0 & & & \\ \hline & 1 & & & \dots & & 0 & \\ & -\alpha & 1 & & & & & \\ & & \ddots & \ddots & & & & \\ 0 & \vdots & & \ddots & \ddots & & & \vdots \\ & & & & \ddots & 1 & & \\ & 0 & & & \dots & -\alpha & 1 & \end{array}\right)$$

und die Diagonalmatrix D_k sei definiert durch

$$D_k = \text{diag}(\underbrace{1,\dots,1}_{(k+1)\text{-mal}}, x_{k+1} - x_0, \dots, x_n - x_{n-k-1}) .$$

Mit diesen Definitionen kann man leicht aus (1.24) nachvollziehen, falls $f = f(0:n)$ und $c = c(0:n)$ der Vektor der dividierten Differenzen ist, dass $c = Lf$ gilt, wobei L die Matrix sei, die wie folgt definiert ist:

$$L = D_{n-1}^{-1} L_{n-1}(1) \cdots D_0^{-1} L_0(1) .$$

Ähnlich erhält man aus (1.26), dass

$$a = Uc \tag{1.27}$$

gilt, wobei die durch (1.27) definierte Matrix U sich auch definieren lässt durch

$$U = L_0(x_0)^T \cdots L_{n-1}(x_{n-1})^T .$$

Mit diesen Definitionen ergibt sich sofort die Aussage $a = ULf$ mit $V^{-T} = UL$. Die Lösung von $Vz = b$ ergibt sich somit durch

$$\begin{aligned} z &= V^{-1}b = L^T(U^T b) \\ &= \left(L_0(1)^T D_0^{-1} \cdots L_{n-1}(1)^T D_{n-1}^{-1}\right) \left(L_{n-1}(x_{n-1}) \cdots L_0(x_0)\right) b . \end{aligned}$$

Diese Erkenntnis führt zu dem folgenden Algorithmus.

Algorithmus 1.8.2: Vandermondsches System: $Vz = b$

Gegeben seien $x(0:n) \in \mathbb{R}^{n+1}$ mit paarweise verschiedenen Einträgen und $b = b(0:n) \in \mathbb{R}^{n+1}$. Der folgende Algorithmus uberschreibt b mit der Lösung $z = z(0:n)$ zu dem Vandermondschen System

$$V(x_0, \ldots, x_n)z = f\,.$$

```
for k = 0 : n − 1
  for i = n : −1 : k + 1
      b(i) = b(i) − x(k)b(i − 1)
   end
end
for k = n − 1 : −1 : 0
   for i = k + 1 : n
      b(i) = b(i)/(x(i) − x(i − k − 1))
   end
   for i = k + 1 : n
      b(i) = b(i) − b(i + 1))
   end
end
```

Satz 1.8.3 *Das Verfahren 1.8.2 benötigt zum Lösen des Vandermondschen System*

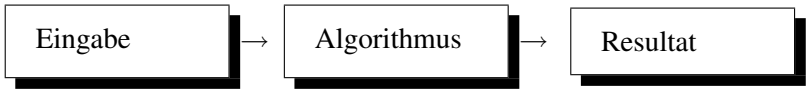
$$V(x_0, \ldots, x_n)a = f$$

nur $5(n^2 + n)/2$ *Operationen.*

Beweis. Analog zu den Betrachtungen im Beweis von Satz 1.8.1 erkennt man, dass die Anweisungen in den inneren FOR-Schleifen von Algorithmus 1.8.2 jeweils $n(n+1)/2$ -mal aufgerufen werden. Jedesmal werden 2 bzw. 1 Gleitkommaoperation benötigt. □

2 ZAHLENDARSTELLUNG, FEHLERARTEN UND KONDITION

Im letzten Kapitel haben wir aus den Eingabegrößen (A, b) das Resultat $f(A, b) := A^{-1}b$ berechnet. Etwas abstrakter formuliert, wir haben eine Abbildung $f : U \subset X \to Y$ an einer Stelle $x \in U$ ausgewertet.



Bei exakter Arithmetik würden wir mit dem Gauß-Verfahren ein exaktes Ergebnis erhalten. Im Allgemeinen treten jedoch sowohl Fehler in der Eingabe als auch im Algorithmus auf, z.B. lässt sich die Zeit nur auf eine Genauigkeit von $10^{-14}s$ und die Masse auf $10^{-8}g$ genau bestimmen[1], Zahlen nur näherungsweise im Rechner als Gleitkommazahl (auch Fließkommazahl genannt, engl. floating point number) darstellen und z. B. auch die Darstellung

$$\sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k\, x^{2k+1}}{(2k+1)!} \tag{2.1}$$

nur näherungsweise auswerten. Welchen Einfluss haben nun beide Fehlerarten

- Eingabefehler (lassen sich nicht vermeiden) und
- Fehler im Algorithmus (wobei keine algorithmischen Fehler gemeint sind)

auf den Fehler im Resultat? Die Unterscheidung beider Arten von Fehlern führt uns zu den Begriffen

Kondition eines Problems und

Stabilität eines Algorithmus.

2.1 ZAHLENDARSTELLUNG

Ein erstes Beispiel soll motivieren, wieso wir uns mit der Frage, wie Zahlen im Rechner dargestellt werden, beschäftigen sollten.

Beispiel 2.1.1 Eine einfache Überlegung zeigt, dass das Ergebnis von $4 \cdot 13860^4 - 19601^4 + 2 \cdot 19601^2$ ungerade ist, eine erste Rechnung, dass an der letzten Stelle eine 1 oder 9 stehen muss. (Dabei könnte sich die 9 z.B. durch $-1 \cdot 10 + 1$ ergeben.)

$$\begin{aligned} & 4 \cdot 13860^4 - 19601^4 + 2 \cdot 19601^2 = 4(1386 \cdot 10)^4 - (1960 \cdot 10 + 1)^4 + 2(1960 \cdot 10 + 1)^2 \\ = \;& 4\,1386^4\,10^4 - (1960^4\,10^4 + 4\,1960^3\,10^3 + 6\,1960^2\,10^2 + 4\,1960\,10 + 1) + 2(1960^2\,10^2 + 2\,1960\,10 + 1) \\ = \;& 10^4(\cdots) + 10^3(\cdots) + 10^2(\cdots) + 10^1(\cdots) + 10^0(-1 + 2) \end{aligned}$$

Rechnet man auch die anderen Stellen aus (per Hand oder mit Maple) so ergibt sich

$$4 \cdot 13860^4 - 19601^4 + 2 \cdot 19601^2 = 1\,.$$

[1] siehe www.ptb.de Physikalisch-technische Bundesanstalt

MATLAB-Beispiel:

Überraschenderweise liefert Matlab aber ein anderes Resultat.

```
>> myfun = @(x,y) 4*x^4-y^4+2*y^2;
>> myfun(13860,19601)
ans =
     2
```

Auch die folgende Realisierung mittels quadratischer Ergänzung liefert ein (anderes) falsches Ergebnis.

```
>> x = 13860; y = 19601;
>> u = 4 * x * x * x * x + 1;
>> v = y * y - 1;
>> u - v * v
ans =
     0
```

⚠ Und Vorsicht beim Umgang mit Maple! Schreibt man dort versehentlich nach einer Zahl noch einen Punkt, so ergibt es das scheinbar überraschende folgende Ergebnis. (Man beachte dabei, dass Maple, wenn man den Punkt hinzufügt, nicht mehr mit exakter Arithmetik rechnet, sondern Gleitkommazahlen verwendet, deren Genauigkeit man einstellen kann. Die Anweisung `Digits:=18` führt dazu, dass Maple Gleitkommazahlen mit 18 Stellen verwendet.

```
> restart:
  4  *13860^4-19601^4+2*19601^2;

  4. *13860^4-19601^4+2*19601^2;
  Digits:=15:
  4. *13860^4-19601^4+2*19601^2;
  Digits:=16:
  4. *13860^4-19601^4+2*19601^2;
  Digits:=17:
  4. *13860^4-19601^4+2*19601^2;
  Digits:=18:
  4. *13860^4-19601^4+2*19601^2;

                        1
                   68398402.
                      402.
                       2.
                       2.
                       1.
```

MATLAB-Beispiel:

Aus dem Topf der mathematischen Mirakel ziehen wir noch folgendes Beispiel:

```
>> floor(114)
ans =
   114
>> floor(1.14*100)
ans =
   113
```

Die Routine `floor` rundet dabei auf die nächstkleinere ganze Zahl. 114 auf die nächstkleinere ganze Zahl gerundet sollte also in beiden Fällen 114 ergeben!

Ein Rechner verwendet als einfachste Form der Zahlendarstellung Festkommazahlen. Dabei wird eine (meistens) begrenzte Ziffernfolge gespeichert und an festgelegter Stelle das Komma angenommen. Bei vielen Anwendungen reicht der darstellbare Zahlenbereich jedoch nicht aus, so dass es naheliegend ist, bei jedem Wert die genaue Stelle des Kommas zusätzlich zu speichern. Das bedeutet mathematisch nichts anderes als die Darstellung der Zahl x mit zwei Werten, der Mantisse m und dem Exponenten e. Wir bekommen somit die Darstellung $x = m \cdot 10^e$. Diese Darstellung ist jedoch nicht eindeutig, siehe z.B. $1014 = 1.014 \cdot 10^3 = 10140 \cdot 10^{-1}$. Legt man jedoch den Wertebereich der Mantisse m geschickt fest, z. B. $1 \leq m < 10$, so erhält man eine eindeutige Darstellung (wenn man endlich viele Stellen hat). Diesen Schritt bezeichnet man als **Normalisierung** der Mantisse. Wir haben soeben stillschweigend vorausgesetzt, dass wir die Basis 10 verwenden[1]. Darauf ist man jedoch nicht festgelegt. Pascal[2] erkannte, dass man jede Basis $b \geq 2$ verwenden kann. Heutige Rechner verwenden meist binäre Zahldarstellungen, d.h. zur Basis 2.
Betrachten wir nun die Zahldarstellung noch etwas allgemeiner.

Definition 2.1.2 (b-adischer Bruch) *Es sei $b \geq 2$ eine natürliche Zahl, $n \in \mathbb{N}_0$, $a_k \in \mathbb{N}_0$ mit $0 \leq a_k < b$ und $a_n \neq 0$. Eine Reihe der Gestalt*

$$\pm \sum_{k=-\infty}^{n} a_k b^k$$

wird als b-adischer Bruch bezeichnet. Falls die Basis festgelegt ist, kann man einen b-adischen Bruch auch durch die Reihung der Ziffern a_k angeben:

$$\pm a_n a_{n-1} \cdots a_1 a_0 \,.\, a_{-1} a_{-2} a_{-3} \cdots .$$

Bemerkung 2.1.3

i.) Für $b = 10$ spricht man von **Dezimalbrüchen**, *für $b = 2$ von* **dyadischen Brüchen** *und die Babylonier[3] verwendeten ein System zur Basis $b = 60$, was sich heute noch im Verhältnis von Sekunde, Minute und Stunde wiederfindet.*

ii.) Die Basis wird auch als Grundzahl bezeichnet, weswegen man auch die Bezeichnung **g-adisch** *findet.*

iii.) Vorsicht, nicht **b-adisch** *mit* **p-adisch** *verwechseln ($p \sim$ Primzahl)!*

iv.) Der Null kommt immer eine Sonderrolle zu. Sie wird gesondert betrachtet, um die Darstellung einfacher zu halten. Ohne den Sonderfall der Null jeweils aufzuführen, schließen wir ihn indirekt mit ein.

Die nächsten beiden Sätze besagen, dass sich jede reelle Zahl durch einen b-adischen Bruch darstellen lässt und umgekehrt.

Satz 2.1.4 (Konvergenz b-adischer Bruch) *Jeder b-adische Bruch stellt eine Cauchy-Folge dar, d.h. dass jeder b-adische Bruch gegen eine reele Zahl konvergiert.*

[1] **Dezimalsystem** Unter dem Einfluss der Schrift „De Thiende“ (Leiden 1585) von Simon Stevin (1548-1620) setzte sich im 16. Jahrhundert das Rechnen mit Dezimalbrüchen in Westeuropa durch.

[2] **Blaise Pascal**, 1623-1662

[3] **Sexagesimalsystem** Zählt man mit den Fingern der linken Hand wie üblich von 1 bis 5 und registriert die an der linken Hand gezählten 5-er Gruppen durch das Drücken des Daumens der rechten Hand auf das passende Fingerglied an der rechten Hand, so kommt man zum $5 \times 12 = 60$-System. Diese Zählweise wurde schon von den Vorfahren der Sumerer im 4 Jahrtausend v. Chr. angewandt. Die Babylonier (1750 v. Chr.) verwendeten dann das Sexagesimalsystem in dem Sinne, dass sie 60 Herzschläge zu einer Minute zusammenzufassten und 60 Minuten zu einer Stunde.

Beweis. Wir beschränken uns auf den Fall eines nicht-negativen b-adischen Bruchs $\sum_{k=-\infty}^{n} a_k b^k$. Wir definieren für $-\ell \le n$ die Partialsummen

$$s_\ell := \sum_{k=-\ell}^{n} a_k b^k \,.$$

Es ist nun zu zeigen, dass die Folge $(s_\ell)_{-\ell \le n}$ eine Cauchy-Folge ist. Sei $\varepsilon > 0$ und $L \in \mathbb{N}$ so groß, dass $b^{-L} < \varepsilon$ gelte. Dann gilt für $\ell \ge m \ge L$

$$\begin{aligned} |s_\ell - s_m| &= \sum_{k=-\ell}^{n} a_k b^k - \sum_{k=-m}^{n} a_k b^k = \sum_{k=-\ell}^{-m-1} a_k b^k \le \sum_{k=-\ell}^{-m-1} (b-1) b^k \\ &= (b-1) b^{-m-1} \sum_{k=-\ell+m+1}^{0} b^k = (b-1) b^{-m-1} \sum_{k=0}^{\ell-m-1} b^{-k} \\ &< (b-1) b^{-m-1} \frac{1}{1-b^{-1}} = b^{-m} \le b^{-L} < \varepsilon \,. \end{aligned}$$

□

Satz 2.1.5 (Reelle Zahl als b-adischer Bruch) *Sei $b \ge 2$ eine natürliche Zahl. Dann lässt sich jede reelle von Null verschiedene Zahl in einen b-adischen Bruch entwickeln.*

Beweis. Es genügt wieder, nur den Fall $x \ge 0$ zu zeigen. Zuerst zeigen wir, dass es zu jeder Basis $b \ge 2$ mindestens eine Zahl $m \in \mathbb{N}$ gibt mit $x < b^m$. Nach dem Archimedischen Axiom gibt es ein $m \in \mathbb{N}$ mit $m > (x-1)/(b-1) \in \mathbb{R}$. Mit diesem m und der Bernoullischen Ungleichung erhalten wir

$$b^m = (1 + (b-1))^m \ge 1 + m(b-1) > 1 + x - 1 = x \,.$$

Sei $n := \min\{k \in \mathbb{N}_0 \,|\, 0 \le x < b^{k+1}\}$. Durch vollständige Induktion konstruieren wir nun eine Folge $(a_{-\ell})_{\ell \ge -n}$ natürlicher Zahlen $0 \le a_{-\ell} < b$, so dass für alle $m \ge -n$ gilt

$$x = \sum_{\ell=-m}^{n} a_\ell b^\ell + \nu_{-m} \quad \text{mit } 0 \le \nu_{-m} < b^{-m} \,.$$

Da $\lim_{m\to\infty} \nu_{-m} = 0$ gilt, folgt $x = \sum_{\ell=-\infty}^{n} a_\ell b^\ell$ und somit die Behauptung.
Induktionsanfang $-m = n$: Es gilt $0 \le x b^{-n} < b$. Somit existiert ein $a_n \in \{0, 1, \ldots, b-1\}$ und ein $\delta \in \mathbb{R}$ mit $0 \le \delta < 1$, so dass $x b^{-n} = a_n + \delta$ gilt. Mit $\nu_n := \delta b^n$ gewinnt man

$$x = a_n b^n + \nu_n \quad \text{mit } 0 \le \nu_n < b^n \,.$$

Induktionsschritt $(-m) \to (-m-1)$: Es gilt $0 \le \nu_{-m} b^{m+1} < b$, also gibt es ein $a_{-m-1} \in \{0, 1, \ldots, b-1\}$ und ein $\delta \in \mathbb{R}$ mit $0 \le \delta < 1$, so dass $\nu_{-m} b^{m+1} = a_{-m-1} + \delta$ gilt. Mit $\nu_{-m-1} := \delta b^{-m-1}$ gewinnt man

$$x = \sum_{\ell=-m}^{n} a_\ell b^\ell + \nu_{-m} = \sum_{\ell=-m}^{n} a_\ell b^\ell + (a_{-m-1} + \delta) b^{-m-1} = \sum_{\ell=-m-1}^{n} a_\ell b^\ell + \nu_{-m-1} \,,$$

wobei $0 \le \nu_{-m-1} < b^{-m-1}$ gilt. □

Bemerkung 2.1.6 *Die b-adische Darstellung ist in der gewählten Form nicht eindeutig, siehe z. B. $1 = 0.\overline{9} = 0.999\ldots$, falls $b = 10$ gewählt ist. Durch die zusätzliche Bedingung „$a_k < b-1$ für unendlich viele $k \le n$“ erhält man eine eindeutige Darstellung. Somit wäre z.B. für $b = 10$ die Darstellung $0.\overline{9} = 0.999\ldots$ nicht zugelassen.*

Bemerkung 2.1.7 *Da es unmöglich ist, reelle Zahlen als unendliche b-adische Brüche zu speichern, werden reelle Zahlen auf Rechnern durch eine endliche Entwicklung approximiert.*

Definition 2.1.8 (Festpunktzahl) *Zu gegebener Basis b ist*

$$F(\ell, n) := \left\{ \pm \sum_{k=-\ell}^{n} a_k b^k = \pm (a_n \dots a_1 a_0 \,.\, a_{-1} \dots a_{-\ell})_b \right\}$$

die Menge der Festpunktzahlen mit $(n+1)$ Vor- und ℓ Nachkommastellen.

Bemerkung 2.1.9 *i.) Typischerweise werden ganze Zahlen (engl. integers) auf Computern dargestellt mit $\ell = 0$ und $b = 2$.*

ii.) In modernen Rechnern werden negative Zahlen häufig als 2-er-Komplement [4] abgespeichert.

Beispiel 2.1.10 Binärdarstellung ($b = 2$) von $x = (14.625)_{10} = (1110.101)_2$

$$\begin{array}{rllll} 14 = & 7 \cdot 2 & +0 & & \Rightarrow a_0 = 0 \\ & 7 & = \quad 3 \cdot 2 & +1 & \Rightarrow a_1 = 1 \\ & & 3 = \quad 1 \cdot 2 & +1 & \Rightarrow a_2 = 1 \\ & & & 1 = 0 \cdot 2 + 1 & \Rightarrow a_3 = 1 \end{array}$$

Im Folgenden bezeichnet $\lfloor \cdot \rfloor$ die untere Gaußklammer, d.h.

$$\lfloor x \rfloor := \max \{ j \in \mathbb{Z} : j \leq x \}.$$

$$\begin{aligned} a_{-1} &= \lfloor 2 * 0.625 \rfloor = \lfloor 1.25 \rfloor = 1 \\ a_{-2} &= \lfloor 2 * 0.25 \rfloor = \lfloor 0.5 \rfloor = 0 \\ a_{-3} &= \lfloor 2 * 0.5 \rfloor = \lfloor 1 \rfloor = 1 \end{aligned}$$

Bemerkung 2.1.11 *Da im 2-er-Komplement der Wert 0 den positiven Zahlen zugeordnet wird, erhält man für 8-Bit[5], 16-Bit und 32-Bit, bzw. 1-, 2- und 4-Byte[6] Darstellungen die Zahlenbereiche*

$$\begin{aligned} 8 - bit, n = 7: \quad & 0, \dots, 255 \text{ ohne Vorzeichen} \\ \text{bzw. } & -128, \dots, 127 \ (8 \text{ Stellen}) \end{aligned}$$

$$\begin{aligned} 16 - bit, n = 15: \quad & 0, \dots, 65535 \text{ ohne Vorzeichen} \\ \text{bzw. } & -32768, \dots, 32787 \ (16 \text{ Stellen}) \end{aligned}$$

$$\begin{aligned} 32 - bit, n = 31: \quad & 0, \dots, 4.294.967.295, \dots, \text{ ohne Vorzeichen} \\ \text{bzw. } & -2.147.483.648, \dots, 2.147.483.647 \ (32 \text{ Stellen}) \end{aligned}$$

[4] **Zweierkomplement** ist eine Möglichkeit, negative Zahlen im Dualsystem darzustellen. Dabei werden keine zusätzlichen Symbole wie + und - benötigt. Da im Zweierkomplement der Wert 0 den positiven Zahlen zugerechnet wird, umfasst der Wertebereich bei n binären Stellen allgemein den Bereich $-2^{n-1}, \dots, 0, \dots 2^{n-1} - 1$. Zum Beispiel bei 4 Bit von -8 bis 7. Negative Zahlen gewinnt man aus positiven Zahlen, indem sämtliche binären Stellen negiert werden und zu dem Ergebnis der Wert 1 addiert wird. Somit wird aus $(4)_{10} = (0100)_2$ durch Invertieren $Not(0100) = 1011$ und Addition der 1 $(1011)_2 + (0001)_2 = (1100)_2$. Addiert man nun 4 und -4 so gilt $(4)_{10} + (-4)_{10} = (0100)_2 + (1100)_2 = (0000)_2 = (0)_{10}$, wobei die vorderste Stelle verworfen wurde.

[5] **Bit** $\sim$ (engl.: binary digit, lat. digitus $\sim$ Finger)

[6] **Byte** $\sim$ Maßeinheit für eine Datenmenge von 8 Bit

Bemerkung 2.1.12 *Als Problem stellt sich dabei heraus, dass damit nur ein kleiner Zahlenbereich darstellbar ist.*

Definition 2.1.13 (Gleitpunkt-Darstellung) *Die normalisierte Gleitpunkt-Darstellung einer reellen Zahl $x \in \mathbb{R}$ hat die Form $x = f \cdot b^{\ell}$, wobei*

- *die* **Mantisse** f *ist eine feste Zahl, die m Stellen hat und*

$$b^{-1} \leq |f| < 1, \text{ falls } x \neq 0 \text{ und } f = 0 \text{ falls, } x = 0$$

genügt, d.h.

$$f = \begin{cases} \pm (0.d_1 \dots d_m)_b = \pm \sum_{j=1}^{m} d_j b^{-j} & , d_1 \neq 0 \\ 0 & , d_1 = 0 \end{cases}$$

- *der* **Exponent** ℓ *ist eine ganze Zahl innerhalb fester Schranken*

$$r \leq \ell \leq R\,,$$

welche sich in der Form

$$\ell = \pm (\ell_{n-1} \cdots \ell_0)_b = \pm \sum_{j=0}^{n-1} \ell_j b^j$$

darstellen lässt. (Für 8-stelligen binären Exponenten wird z.B. $r = -128$ und $R = 127$ gewählt.)

Die Menge der Maschinenzahlen mit Basis b, m-stelliger Mantisse und n-stelligem Exponenten wird als $\mathbb{M}(b, m, n)$ bezeichnet.

Beispiel 2.1.14 $\mathbb{M}(2, 3, 1)$ besteht aus den Zahlen

$$x = \pm(d_{-1} \cdot 2^{-1} + d_{-2} \cdot 2^{-2} + d_{-3} \cdot 2^{-3}) \cdot 2^{\pm \ell_0},$$

wobei $d_{-1} \neq 0 \vee x = d_{-1} = d_{-2} = d_{-3} = 0$ gilt. Somit

$$\mathbb{M}(2,3,1) = \left\{0, \pm\frac{1}{4}, \pm\frac{5}{16}, \pm\frac{3}{8}, \pm\frac{7}{16}, \pm\frac{1}{2}, \pm\frac{5}{8}, \pm\frac{3}{4}, \pm\frac{7}{8}, \pm 1, \pm\frac{5}{4}, \pm\frac{3}{2}, \pm\frac{7}{4}\right\}.$$

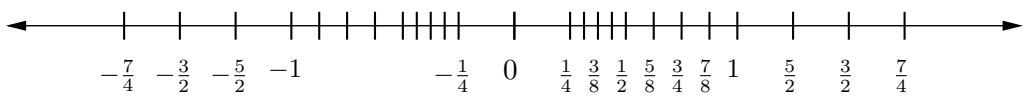


Abb. 2.1: Graphische Darstellung von $\mathbb{M}(2, 3, 1)$.

Man beachte den kleiner werdenden Abstand zur Null hin und den Abstand zwischen den betragsmäßig kleinsten Zahlen und der Null.

Bemerkung 2.1.15 *Die betragsmäßig kleinste und größte Zahl in $\mathbb{M}(b, m, n)$ bezeichnen wir mit $x_{\min}$ bzw. $x_{\max}$,*

$$\begin{array}{ll} x_{\min} & := \min\{|z| \in \mathbb{M}(b,m,n)\}\,, \quad |x| < x_{\min} \Rightarrow x = 0 \\ x_{\max} & := \max\{|z| \in \mathbb{M}(b,m,n)\}\,, \quad |x| > x_{\max} \Rightarrow \textbf{overflow} \end{array} \tag{2.2}$$

Beispiel 2.1.16

$$\begin{array}{lll} \mathbb{M}(10,4,2), & x = -51.34 & \rightsquigarrow -0.5134 \cdot 10^2 \\ \mathbb{M}(2,4,2), & x = 3.25 = (11.01)_2 & \rightsquigarrow (0.1101)_2 \cdot 2^{(10)_2} \\ \mathbb{M}(2,4,2), & x = 14.625 = (1110.101)_2 & \rightsquigarrow (0.1110101)_2 \cdot 2^{(100)_2} \notin \mathbb{M}(2,4,2) \end{array}$$

Bemerkung 2.1.17 *Üblicherweiser werden 8 Byte (64 Bit) Speicherplatz für eine Gleipunktzahl wie folgt aufgeteilt,*

- *1 Bit für das Vorzeichen,*
- *11 Bit für den Exponenten, d.h. Werte sind darstellbar im Bereich -1022 ... 1023 ($1 \cdots 2046$ - Bias[7]-Wert 1023) . Die Werte -1023 und 1024, bzw. 0 und $2047 = 2^{11} - 1$ als Charakteristik sind reserviert für die speziellen Zahlenwerte „Null“, „Unendlich“ und „NaN“,*
- *52 Bit für die Mantisse. Dies sind eigentlich Werte zwischen $(0 \ldots 01)_2$ und $(1 \ldots 1)_2$, da die Null gesondert abgespeichert wird. Da die erste Null jedoch redundant ist, gewinnt man noch eine Stelle hinzu, wenn man in der normalisierten Darstellung ist. Also muss auch noch irgendwie festgehalten werden, ob man in der normalisierten Darstellung ist oder nicht.*

Exponent e	Mantisse	Bedeutung	Bezeichnung
$e = 0$	$m = 0$	± 0	Null
$e = 0$	$m > 0$	$\pm 0, m \times 2^{1-Bias}$	denormalisierte Zahl
$0 < e < e_{max}$	$m \geq 0$	$\pm 1, m \times 2^{e-Bias}$	normalisierte Zahl
$e = e_{max}$	$m = 0$	$\pm\infty$	Unendlich (Inf)
$e = e_{max}$	$m > 0$	keine Zahl	**N**ot **a** **N**umber (NaN)

Tab. 2.1: Interpretation des Zahlenformats nach IEEE-754

Typ	Größe	Exponent	Mantisse	Werte des Exponenten (e)
single	32 Bit	8 Bit	23 Bit	-126 $\leq$ e $\leq$ 127
double	64 Bit	11 Bit	52 Bit	-1022 $\leq$ e $\leq$ 1023
	rel. Abstand zweier Zahlen	Dezimalstellen	betragsmäßig kleinste Zahl	größte Zahl
single	$2^{-(23+1)}$	7-8	$2^{-23} \cdot 2^{-126}$	$(1 - 2^{-24})2^{128}$
double	$2^{-(52+1)}$	15-16	$2^{-52} \cdot 2^{-1022}$	$(1 - 2^{-53})2^{1024}$

Tab. 2.2: Zahlenformat nach IEEE-754

Bemerkung 2.1.18 *Der Standardtyp in Matlab, wenn nichts Weiteres angegeben wird, ist double.*

[7](oder auch Offset) hier zu verstehen als ein konstanter Wert, der zu etwas hinzuaddiert bzw. subtrahiert wird

Mit Hilfe der folgenden mex-Routine `digits.c` lassen sich auf einem 32-Bit Rechner unter Matlab die einzelnen Bits einer double precision-Variablen auslesen. (Kompilieren der c-Routine auf der Kommandozeilenebene von Matlab mit `mex digits.c`.)

C-Funktion: digits.c

```c
#include "mex.h"
bool get_bit_at(double y[], int pos)
{
  unsigned long bitmask;
  unsigned long *cy = (unsigned long *) y;

  if (pos >= 0){
    if (pos < 32){
        bitmask = 1 << pos;       // bitmask  setzen
        return (cy[0] & bitmask)!= 0;
    } else if (pos < 64){
        bitmask = 1 << (pos-32); // bitmask  setzen
                                 // unsigned long ist nur
                                 // 4 Byte = 32 Bit lang
        return (cy[1] & bitmask)!= 0;
    }
  }
  mexErrMsgTxt("No valid position for Bit.");
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
  double *x,*y;
  mwSize mrows,ncols;
  int k;
  /* Check for proper number of arguments. */
  if(nrhs!=1) {
    mexErrMsgTxt("One input required.");
  } else if(nlhs>1) {
    mexErrMsgTxt("Too many output arguments");
  }
  /* The input must be a noncomplex scalar double.*/
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
      !(mrows==1 && ncols==1) ) {
    mexErrMsgTxt("Input must be a noncomplex scalar double.");
  }
  /* Create matrix for the return argument. */
  plhs[0] = mxCreateDoubleMatrix(1,64, mxREAL);
  /* Assign pointers to each input and output. */
  x = mxGetPr(prhs[0]);
  y = mxGetPr(plhs[0]);
  /* Call the digit subroutine. */
  for (k=0; k<64; k++) y[63-k] = (double) get_bit_at(x,k);
}
```

Versuchen Sie sich die Ergebnisse der folgenden Matlab-Ausgaben zu erklären bzw. testen Sie es auf Ihrem eigenen Rechner.

```
>> A=digits(-1);A(1)
ans =
     1
>> A=digits(1);A(1)
ans =
     0
>> A=digits(2^(1023)*(1.9999999999999998));
>> A(1),A(2:12),[A(13:29); A(30:46); A(47:63)],A(64)
ans =
     0
ans =
     1  1  1  1  1  1  1  1  1  1  0
ans =
     1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
     1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
     1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
ans =
     1
>> A=digits(2^(-971)*(2-1.9999999999999995));
>> A(1),A(2:12),[A(13:29); A(30:46); A(47:63)],A(64)
ans =
     0
ans =
     0  0  0  0  0  0  0  0  0  0  1
ans =
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
ans =
     0
>> A=digits(2^(-972)*(2-1.9999999999999995));
>> A(1),A(2:12),[A(13:29); A(30:46); A(47:63)],A(64)
ans =
     0
ans =
     0  0  0  0  0  0  0  0  0  0  0
ans
     1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
ans =
     0
>> A=digits(realmax)        % try by yourself
>> A=digits(realmin)
>> A=digits(realmin/2^52)
>> A=digits(2^0+2^(-1)); A(13:29)
>> A=digits(2^0+2^(-1)+2^(-2)); A(13:29)
>> A=digits(2^0+2^(-1)+2^(-2)+2^(-3)); A(13:29)
>> A=digits(2^0+2^(-1)+2^(-2)+2^(-3)+2^(-17))A(13:29)1
```

Bemerkung 2.1.19 *i) Die Werte von b, m, ℓ hängen in der Regel vom Typ des jeweiligen Computers ab, ebenso r, R. Der IEEE[8]-Standard versucht dies zu vereinheitlichen.*

ii) Bei der Umwandlung „exakter" Eingaben in Maschinenzahlen können Fehler auftreten, z.B.

$$x = 0.1 = \left(0.000\overline{1100}\right)_2$$

2.2 Rundung und Maschinengenauigkeit

Aufgabe: Approximiere $x \in \mathbb{R}$ durch $fl(x) \in \mathbb{M}(b, m, \ell)$ („float")

$$fl : \mathbb{R} \to \mathbb{M}(b, m, \ell)$$

bzw.

$$fl : [-x_{\max}, x_{\max}] \to \mathbb{M}(b, m, \ell)$$

Dies wird durch geeignete Rundungsstrategien gewährleistet. Wir betrachten zwei konkrete Beispiele:

2.2.1 Standardrundung, kaufmännische Rundung

(entspricht „normalem Runden" $1.4 \to 1$, $1.6 \to 2$)

Die **Standardrundung** ist definiert durch

$$fl(x; b, m, \ell) = fl(x) := \pm \sum_{j=1}^{m} d_j \, b^{\ell_j} + \begin{cases} 0 & \text{, falls } d_{m+1} < \frac{b}{2} \\ b^{\ell_m} & \text{, falls } d_{m+1} \geq \frac{b}{2} \end{cases}$$

und

$$|x| < x_{\min} \Rightarrow fl\,(x) := 0$$
$$|x| > x_{\max} \Rightarrow \textbf{overflow}\ .$$

Bemerkung 2.2.1 *Man mache sich klar, dass die Rundung im Falle $d_{m+1} = b/2$ (b gerade, $b/2$ ungerade) rein willkürlich ist.*

2.2.2 „Round to even" (oder auch Banker's Rule oder mathematisch unverzerrte Rundung)

Die unverzerrte Rundung (englisch „round to even"), auch als Banker's Rule bekannt, ist eine Rundungsregel, welche sich dann von der Standardrundung unterscheidet, wenn der zu rundende Rest genau zwischen zwei Zahlen mit der gewählten Anzahl von Ziffern liegt. Voraussetzung zur Anwendung der Banker's Rule ist, dass die Basis b gerade und $b/2$ ungerade ist, dies ist z.B. für die interessanten Fälle $b = 2,\ 10$ erfüllt. Die Standardrundung erzeugt statistische „Fehler", da das Aufrunden um $b/2$ vorkommt, das Abrunden um $b/2$ jedoch nie. Rundet man z.B. bei großen Mengen von Daten, bei denen der Trend untersucht werden soll, immer auf, falls der zu rundende Rest genau $b/2$ ist, so verschiebt sich der Trend aufwärts. Außerdem ist das Verhalten bei positiven und negativen Zahlen unterschiedlich, wenn die zu rundende Ziffer $b/2$ ist. Das „Round to Even" vermeidet dieses Verhalten. Es rundet von der genauen Mitte zwischen zwei Ziffern immer zur nächsten geraden Zahl auf oder ab. Ansonsten entspricht es der Standardrundung.

[8] IEEE∼ Institute of Electrical and Electronics Engineers

Bemerkung 2.2.2 *Man könnte genauso gut auch zur nächsten ungeraden Zahl runden, dann würde man jedoch nie auf Null runden.*

Banker's Rule: Voraussetzung: b gerade, $b/2$ ungerade. Es sei m die vorgegebene Mantissenlänge und $x = \pm(0.d_1d_2d_3\ldots)_b b^{e_1+1}$ ($d_1 \neq 0, d_k < b-1$ für unendlich viele $k \in \mathbb{N}$). Dann ist die Banker's Rule definiert durch

$$fl(x) := \pm \sum_{j=1}^{m} d_j\, b^{e_j} + \begin{cases} 0 & \text{, falls } d_{m+1} < b/2 \\ b^{e_m} & \text{, falls } d_{m+1} > b/2 \text{ oder} \\ & \quad d_{m+1} = b/2 \text{ und es ex. ein } k > m+1 \text{ mit } d_k \neq 0 \\ 0 & \quad \sum_{k=m+1}^{\infty} d_k\, b^{e_k} = b^{e_m}/2 \text{ und } d_m \text{ gerade} \\ b^{e_m} & \quad \sum_{k=m+1}^{\infty} d_k\, b^{e_k} = b^{e_m}/2 \text{ und } d_m \text{ ungerade} \end{cases}$$

Beispiel 2.2.3 Es sei $b = 10$. Dann erhält man mit der Banker's Rule folgende Ergebnisse:

aus 2.33 wird 2.3,	aus -2.33 wird -2.3,
aus 2.35 wird 2.4,	aus -2.45 (exakt) wird -2.4,
aus 2.45 (exakt) wird 2.4,	aus -2.4500001 wird -2.5,
aus 2.4500001 wird 2.5 und	aus -2.449 wird -2.4.

Bemerkung 2.2.4 *i.) Der Vorteil der Banker's Rule liegt in einer „besseren" statistischen Verteilung der Rundungsfehler.*

ii.) Zur Basis 10 lautet die Regel: Folgt auf die letzte beizubehaltende Ziffer lediglich eine 5 (oder eine 5, auf die nur Nullen folgen), so wird derart gerundet, dass die letzte beizubehaltende Ziffer gerade wird.

Beispiel 2.2.5 (Wiederholtes Runden)

Standard	$2.445 \to 2.45 \to 2.5 \to 3$	(Fehler 0.555)
	$1.445 \to 1.45 \to 1.5 \to 2$	(Fehler 0.555)
Banker's Rule	$2.445 \to 2.44 \to 2.4 \to 2$	(Fehler 0.445)
	$1.445 \to 1.44 \to 1.4 \to 1$	(Fehler 0.445)

Definition 2.2.6 (absoluter/relativer Fehler) *Es sei $x \in \mathbb{R}$ und $\tilde{x} \in \mathbb{R}$ eine Approximation.*

(i) Die Größe $|x - \tilde{x}|$ heißt **absoluter Fehler**.

(ii) Die Größe $\frac{|x-\tilde{x}|}{|x|}$ heißt **relativer Fehler**.

Lemma 2.2.7 *Die Basis b sei gerade. Es sei $x = \pm b^{\ell} \sum_{k=1}^{\infty} d_k\, b^{-k}$ ($0 < d_1 < b$) und $fl(x)$ die Standardrundung von x auf m Stellen. Dann gilt*

$$|fl(x) - x| \leq \frac{b^{1-m}}{2} \cdot b^{\ell-1}$$

und

$$\frac{|fl(x) - x|}{|x|} \leq \frac{b^{1-m}}{2}\,.$$

Beweis. O.B.d.A sei x positiv. Sei $n \in \mathbb{N}$. Für $b > 1$ und $0 \leq d_k \leq b-1$ ($k = n, n+1, \ldots$) gilt

$$\sum_{k=n}^{\infty} d_k\, b^{-k} \leq \sum_{k=n}^{\infty} (b-1) b^{-k} = \sum_{k=n-1}^{\infty} b^{-k} - \sum_{k=n}^{\infty} b^{-k} = b^{-(n-1)} \tag{2.3}$$

Betrachten wir zuerst die Situation des Abrundens bei der Standardrundung, was der Situation $d_{m+1} < b/2$ entspricht. Da b nach Voraussetzung gerade ist, gilt somit $b/2 \in \mathbb{Z}$. Somit folgt aus $d_{m+1} < b/2$, dass auch $d_{m+1} + 1 \leq b/2$ gilt. Da $0 < d_1 < b$ vorausgesetzt wurde (was der normierten Gleitpunktdarstellung 2.1.13 entspricht), hat man $b^{-1} \leq \sum_{k=1}^{\infty} d_k\, b^{-k} \leq 1$ und somit $b^{\ell-1} \leq x \leq b^{\ell}$. Beachtet man $x - fl(x) = b^{\ell} \sum_{m+1}^{\infty} d_k\, b^{-k}$, so schließt man nun für $d_{m+1} < b/2$ mit (2.3)

$$\begin{aligned} |fl(x) - x| &= b^{\ell}\left[d_{m+1}\, b^{-(m+1)} + \sum_{k=m+2}^{\infty} d_k\, b^{-k}\right] \leq b^{\ell}\left[d_{m+1}\, b^{-(m+1)} + b^{-(m+1)}\right] \\ &\leq b^{\ell-(m+1)}\,(d_{m+1}+1) \leq b^{\ell-(m+1)}\frac{b}{2} = \frac{b^{1-m}}{2} b^{\ell-1} \end{aligned} \tag{2.4}$$

Für das Aufrunden, d.h. für den Fall $d_{m+1} \geq \frac{b}{2}$ gilt nun $d_{m+1}\, b^{-1} \geq \frac{1}{2}$ und analog zu (2.4)

$$|f\ell\,(x) - x| = b^{\ell}\left[b^{-m} - \sum_{k=m+1}^{\infty} d_k\, b^{-k}\right] \leq b^{\ell}\left[b^{-m} - d_{m+1}\, b^{-(m+1)}\right]$$

$$= b^{\ell-m}\underbrace{|\,1 - \underbrace{d_{m+1}\, b^{-1}}_{\geq 1/2}\,|}_{\leq 1/2} \leq \frac{b^{1-m}}{2} b^{\ell-1}\,.$$

Damit ist die Abschätzung für den absoluten Fehler bewiesen und die Abschätzung für den relativen Fehler erhält man sofort mit $b^{\ell-1} \leq x$. □

Definition 2.2.8 *Die Zahl* $eps := \frac{b^{1-m}}{2}$ *heißt (relative)* **Maschinengenauigkeit**.

Definition 2.2.9 *Wir definieren*

$$\mathbb{M}' := \{x \in \mathbb{R} : x_{\min} \leq |x| \leq x_{\max}\} \cup \{0\}$$

Bemerkung 2.2.10 *(i) Es gilt* $eps = \inf\{\delta > 0 : f\ell\,(1+\delta) > 1\}$

(ii) In Matlab liefert die Anweisung eps die Maschinengenauigkeit, die sich aber auch durch folgende Routine mineps.m bestimmen lässt. Unter anderen Programmiersprachen geht dies ähnlich, man muss sich jedoch vergewissern, dass der Compiler die Anweisung `while 1 < 1 + value` *nicht zu* `while 0 < value` *optimiert!*

(iii) $\forall x \in \mathbb{M}', x \neq 0\ \exists \varepsilon \in \mathbb{R}$ *mit* $|\varepsilon| < eps$ *und* $\frac{|f\ell(x)-x|}{|x|} = \varepsilon$, *d.h.*

$$f\ell\,(x) = x\,(1+\varepsilon) \tag{2.5}$$

MATLAB-Funktion: mineps.m

```
function value = mineps
value = 1;
while 1 < 1 + value
  value = value / 2;
end
value = value * 2;
```

MATLAB-Beispiel:

Die Funktion `mineps` liefert das gleiche Ergebnis wie die Matlab-Konstante `eps`. Ergänzend sei noch die Matlab-Konstante `realmin` wiedergegeben. Sie ist die kleinste positive darstellbare Maschinenzahl.

```
>> eps
ans =
    2.220446049250313e-016
>> mineps
ans =
    2.220446049250313e-016
>> realmin
ans =
    2.225073858507201e-308
```

Voraussetzung 2.2.11 *Wir unterstellen im Folgenden, dass für die Addition $\oplus$ im Rechner gilt*

$$x \oplus y = fl(x+y) \quad \text{für alle } x, y \in \mathbb{M} \text{ mit } |x+y| \in \mathbb{M}' \tag{2.6}$$

und entsprechendes auch für die anderen Grundrechenarten. Man sagt $\oplus$ ist **exakt gerundet**.

Folgerung 2.2.12 *Für alle $x, y \in \mathbb{M}$ mit $x + y \in \mathbb{M}'$ gilt*

$$x \oplus y = (x+y)(1+\varepsilon) \tag{2.7}$$

für ein geeignetes $|\varepsilon| \leq$ eps, und entsprechendes auch für die anderen Grundrechenarten.

Bemerkung 2.2.13 *Die Maschinenaddition und -multiplikation sind nicht assoziativ, z.B. gilt für $b = 10$ und $m = 2$*

$$(100 \oplus 4) \oplus 4 = 100 \neq 110 = 100 \oplus (4 \oplus 4)\,. \tag{2.8}$$

Man beachte dabei $100 \oplus 4 = 100$, da $100 + 4$ auf 2 Stellen gerundet 100 ergibt, und dass $100 + 8$ auf 2 Stellen gerundet 110 ergibt.

Der unangenehmste Effekt der Gleitpunktarithmetik ist die sogenannte Auslöschung.

Beispiel 2.2.14 (Auslöschung) Wir betrachten die Rechnung

$$0.1236 + 1.234 - 1.356 = 0.0016 = 0.1600 \cdot 10^{-2}. \tag{2.9}$$

Gleitpunktrechnung mit $b = 10, m = 4$ liefert

$$(0.1236 \oplus 1.234) \ominus 1.356 = 1.358 \ominus 1.356 = 0.2000 \cdot 10^{-2}, \tag{2.10}$$

d.h. schon die erste Stelle des berechneten Ergebnisses ist falsch! Der Rundungsfehler der ersten Addition wird durch die nachfolgende Subtraktion extrem verstärkt. (Dieser Effekt tritt nicht bei Multiplikation und Division auf.)

Bemerkung 2.2.15 *Auch wenn alle Summanden positiv sind, kann etwas schiefgehen: Sei wie im vorigen Beispiel $b = 10$ und $m = 4$. Wir wollen berechnen*

$$\sum_{n=0}^{N} a_n,\ a_0 = 1. \tag{2.11}$$

Falls $0 \leq a_n < 10^{-4}$ für alle $n > 0$, und falls wir der Reihe nach summieren, ist das berechnete Ergebnis 1, egal wie groß die Summe wirklich ist. Eine erste Abhilfe ist es, der Größe nach zu summieren (zuerst die kleinste Zahl). In unserem Beispiel (2.11) hilft das allerdings nicht wirklich, das berechnete Ergebnis ist dann höchstens 2. Besser ist jedoch folgendes Vorgehen: Man ersetzt die zwei kleinsten Zahlen durch ihre Summe und wiederholt diesen Vorgang so lange, bis nur noch eine Zahl, nämlich die gesuchte Summe, übrigbleibt.

2.3 GLEITKOMMAARITHMETIK

Beispiel 2.3.1 (Guard Digit) . Sei $\mathbb{M} = \mathbb{M}(10, 3, 1)$ und betrachte $\ominus$. Sei weiter $x = 0.215 \cdot 10^9$ und $y = 0.125 \cdot 10^{-9}$. Naive Realisierung von $x \ominus y = \mathrm{rd}(x - y)$ erfordert **schieben** von y auf den größeren Exponenten $y = 0.125 \cdot 10^{-18} \cdot 10^9$ und **subtrahieren** der Mantissen:

$$\begin{array}{rcl} x & = & 0.215000000000000000000 \cdot 10^9 \\ y & = & 0.000000000000000000125 \cdot 10^9 \\ \hline x - y & = & 0.214999999999999999875 \cdot 10^9 \end{array} \tag{2.12}$$

Runden auf drei Stellen liefert dann $x \ominus y = 0.215 \cdot 10^9$. Dies erfordert einen Addierer mit $2(b^\ell - 1) + m = 21$ Stellen. In diesem Fall hätten wir das Ergebnis auch durch die Abfolge **Schieben**, **Runde** y, **Subtrahiere** erhalten. Im Allgemeinen ist das aber nicht gut wie folgendes Beispiel zeigt:

$$\begin{array}{rcl} x & = & 0.101 \cdot 10^1 \\ y & = & 0.993 \cdot 10^0 \end{array} \quad \longrightarrow \quad \begin{array}{rcll} x & = & 0.101 & \cdot 10^1 \\ \mathrm{y} & = & 0.099 & \cdot 10^1 \\ \hline x \ominus y & = & 0.002 & \cdot 10^1 \end{array} .$$

Für den relativen Fehler im Ergebnis gilt dann

$$\frac{(x \ominus y) - (x - y)}{(x - y)} = \frac{0.02 - 0.017}{0.017} \approx 0.176 \approx 35\mathrm{eps}\,,$$

wobei $\mathrm{eps} = 1/2 \cdot 10^{-3+1} = 0.005$ sei.
Verwenden wir nun einen $m + 1$-stelligen Addierer, dann erhalten wir

$$\begin{array}{rcl} x & = & 0.1010 \cdot 10^1 \\ y & = & 0.0993 \cdot 10^1 \\ \hline x - y & = & 0.0017 \cdot 10^1 \end{array} .$$

Das Ergebnis $x \ominus y = 1.7 \cdot 10^{-2}$ ist exakt!

Bemerkung 2.3.2 *Allgemein kann man zeigen (siehe [Knuth]): Mit einer zusätzlichen Stelle (sogenannter* **Schutzziffer** *oder* **Guard Digit***) gilt*

$$\frac{(x \ominus y) - (x - y)}{(x - y)} \leq 2\ \mathit{eps}\,.$$

Mit nur 2 Guard Digits erhält man sogar genau das gerundete Ergebnis der exakten Rechnung, d.h. das gerundete Ergebnis aus (2.12) *ohne einen* $2(b^\ell - 1) + m$*-stelligen Addierer.*

Völlig unkalkulierbar sind Gleitkommaoperationen aber nicht. Es gilt z.B. folgendes Resultat: Sind u, v Gleitkommazahlen und

$$\begin{array}{rclcrcl} u' & = & (u \oplus v) \ominus v\,, & & v' & = & (u \oplus v) \ominus u\,, \\ u'' & = & (u \oplus v) \ominus v'\,, & & v'' & = & (u \oplus v) \ominus u'\,, \end{array}$$

dann folgt

$$u + v = (u \oplus v) + ((u \ominus u') \oplus (v \ominus v'))\,.$$

Dies erlaubt eine Berechnung des Fehlers mittels Gleitkommaarithmetik. (Siehe [Knuth, 4.2.2, Theorem B].)

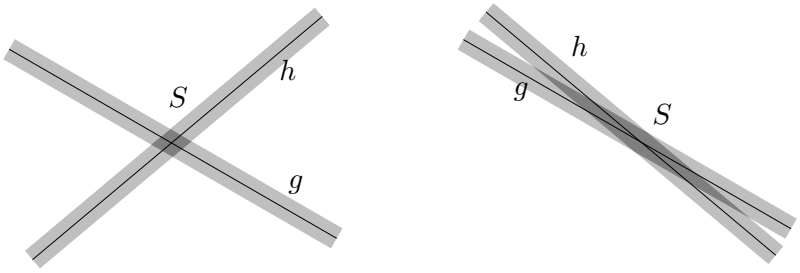


Abb. 2.2: Schnittpunkt S zweier Geraden g und h. Links gut konditioniert, rechts schlecht konditioniert.

2.4 KONDITION EINES PROBLEMS

Diskutieren wir am Anfang zuerst folgendes geometrische Problem: die zeichnerische Bestimmung des Schnittpunkts S zweier Geraden g und h in der Ebene. Schon beim Zeichnen haben wir Schwierigkeiten, die Geraden ohne Fehler darzustellen. Die Frage, die wir näher untersuchen wollen, lautet, wie stark der Schnittpunkt S (Output) von den Zeichenfehlern (Fehler im Input) abhängt.
Wie wir der Grafik direkt entnehmen können, hängt der Fehler in der Ausgabe stark davon ab, in welchem Winkel $\measuredangle(g,h)$ sich die Geraden schneiden. Stehen g und h annähernd senkrecht aufeinander, so variiert der Schnittpunkt S etwa im gleichen Maße wie der Fehler beim Zeichnen von g und h. Man bezeichnet das Problem, den Schnittpunkt S zeichnerisch zu bestimmen, als gut konditioniert. Ist jedoch der Winkel $\measuredangle(g,h)$ sehr klein (g und h sind fast parallel), so kann man schon mit dem Auge keinen genauen Schnittpunkt S ausmachen, und eine kleine Lageänderung von g oder h liefert einen gänzlich anderen Schnittpunkt. Man spricht hier von einem schlecht konditionierten Problem.
Kommen wir nun zu einer mathematischen Präzisierung des Konditionsbegriffs.

Problem 2.4.1 *Seien X, Y Mengen und $\varphi : X \to Y$. Wir betrachten das Problem:*

Gegeben sei $x \in X$, gesucht $y = \varphi(x)$.

Wir untersuchen, wie sich Störungen in den Daten x auf das Ergebnis y auswirken. Wir betrachten den Spezialfall $\varphi : \mathbb{R}^n \to \mathbb{R}$,

$$\varphi(x) = \langle a, x\rangle + b, \;\; a \in \mathbb{R}^n, \;\; b \in \mathbb{R}, \tag{2.13}$$

wobei $\langle \cdot, \cdot \rangle$ das (euklidische) Skalarprodukt bezeichne. Seien $x, \tilde{x} \in \mathbb{R}^n$. Der relative Fehler von $\varphi(\tilde{x})$ bezüglich $\varphi(x)$ lässt sich folgendermaßen abschätzen (falls $\varphi(x) \neq 0$ und $x_j \neq 0$ für alle j):

$$\left|\frac{\varphi(x)-\varphi(\tilde{x})}{\varphi(x)}\right| = \frac{|\langle a, x-\tilde{x}\rangle|}{|\varphi(x)|} \leq \frac{\sum_{j=1}^n |a_j||x_j-\tilde{x}_j|}{|\varphi(x)|} = \sum_{j=1}^n \frac{|x_j|}{|\varphi(x)|}|a_j| \cdot \frac{|x_j-\tilde{x}_j|}{|x_j|}. \tag{2.14}$$

Setzen wir $\tilde{x}_j = fl(x_j)$ in (2.14) ein, so ergibt sich mit Lemma 2.2.7, dass die Zahl

$$eps \cdot \sum_{j=1}^n \frac{|x_j|}{|\varphi(x)|}|a_j| \tag{2.15}$$

den unvermeidlichen Fehler (d.h. selbst bei exakter Rechnung) bei der Berechnung von φ repräsentiert. Sei nun $\varphi : \mathbb{R}^n \to \mathbb{R}$ differenzierbar in x. Dann ist

$$\varphi(x) - \varphi(\tilde{x}) = \langle \nabla\varphi(x), x-\tilde{x}\rangle + o(\|x-\tilde{x}\|), \tag{2.16}$$

und (2.14) wird zu

$$\left|\frac{\varphi(x)-\varphi(\tilde{x})}{|\varphi(x)|}\right| \leq \sum_{j=1}^{n} \frac{|x_j|}{|\varphi(x)|}\left|\frac{\partial\varphi}{\partial x_j}(x)\right| \cdot \frac{|x_j-\tilde{x}_j|}{|x_j|} + o(\|x-\tilde{x}\|). \tag{2.17}$$

Ist schließlich $\varphi : \mathbb{R}^n \to \mathbb{R}^m$, so können wir die bisherigen Überlegungen auf jede Komponente von φ einzeln anwenden.

Definition 2.4.2 (Konditionszahlen) *Sei $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ differenzierbar in $x \in \mathbb{R}^n$, sei $\varphi_i(x) \neq 0,\ 1 \leq i \leq m$. Die Zahlen*

$$k_{ij}(x) = \frac{|x_j|}{|\varphi_i(x)|}\left|\frac{\partial\varphi_i}{\partial x_j}(x)\right|, \quad 1 \leq i \leq m,\ 1 \leq j \leq n, \tag{2.18}$$

heißen die Konditionszahlen von φ in x.

Beispiel 2.4.3

1. Multiplikation: $\varphi : \mathbb{R}^2 \to \mathbb{R}, \varphi(x_1, x_2) = x_1 \cdot x_2$,

$$k_1(x) = \frac{|x_1|}{|x_1 x_2|}\left|\frac{\partial\varphi}{\partial x_1}(x)\right| = 1, \ \ k_2(x) = 1. \tag{2.19}$$

   Keine Verstärkung des relativen Fehlers; die Multiplikation ist „gut konditioniert“.

2. Addition: $\varphi : \mathbb{R}^2 \to \mathbb{R}, \varphi(x_1, x_2) = x_1 + x_2$,

$$k_1(x) = \frac{|x_1|}{|x_1+x_2|}\left|\frac{\partial\varphi}{\partial x_1}(x)\right| = \frac{|x_1|}{|x_1+x_2|}, \ \ k_2(x) = \frac{|x_2|}{|x_1+x_2|}. \tag{2.20}$$

   Im Falle der Auslöschung, d.h. wenn $|x_j| \gg |x_1 + x_2|$, große Verstärkung des relativen Fehlers; die Addition ist in diesem Fall „schlecht konditioniert“.

3. Lösen der quadratischen Gleichung $x^2 + 2px - q = 0$ im Fall $p, q > 0$, Berechnung der größeren der beiden Nullstellen:

$$\varphi(p, q) = -p + \sqrt{p^2+q}, \ \varphi : \mathbb{R}^2 \to \mathbb{R}, \tag{2.21}$$

   Eine Rechnung ergibt

$$k_p = \frac{p}{\sqrt{p^2+q}}, \ \ k_q = \frac{p+\sqrt{p^2+q}}{2\sqrt{p^2+q}}, \tag{2.22}$$

   also $k_p \leq 1, \ k_q \leq 1$; das Problem ist ebenfalls gut konditioniert.

2.5 Numerische Stabilität eines Algorithmus

Jeder Algorithmus zur Lösung von Problem 2.4.1 lässt sich auffassen als eine Abbildung $\tilde{\varphi} : X \to Y$. Von einem guten Algorithmus wird man erwarten, dass er die relativen Fehler nur unwesentlich mehr verstärkt als die Konditionszahlen $k_{ij}(x)$ des Problems φ es erwarten lassen. Für $\tilde{\varphi} : \mathbb{R}^n \to \mathbb{R}^m$ sollte also eine Abschätzung der Form

$$\left|\frac{\tilde{\varphi}_i(\tilde{x})-\tilde{\varphi}_i(x)}{\tilde{\varphi}_i(x)}\right| \leq C_1 \underbrace{\sum_{j=1}^{n} k_{ij}(x)\frac{|\tilde{x}_j - x_j|}{|x_j|}}_{\geq |\varphi(\tilde{x})-\varphi(x)|\ |\varphi(x)| - o(\|\tilde{x}-x\|)} + C_2\, n\, \text{eps}, \tag{2.23}$$

(oder so ähnlich) gelten mit Konstanten C_i, welche nicht viel größer als 1 sind.

Definition 2.5.1 (Numerische Stabilität eines Algorithmus) *Ein Algorithmus $\tilde{\varphi} : \mathbb{R}^n \to \mathbb{R}^m$ zur Lösung eines Problems $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ heißt numerisch stabil (oder gut konditioniert), falls (2.23) oder etwas Ähnliches gilt mit vernünftigen Konstanten C_i. Andernfalls heißt der Algorithmus numerisch instabil (oder schlecht konditioniert).*

Bemerkung 2.5.2 (Konsistenz eines Algorithmus) *Aus* (2.23) *ist nicht ersichtlich, was das exakte Datum $\varphi(x)$ mit $\tilde{\varphi}(x)$ zu tun hat. (Z.B. ist $\tilde{\varphi}(x) = 0$ ein stabiles Verfahren, aber im Allgemeinen nutzlos.) Für diesen Zusammenhang hat man in der Numerik den Begriff der* **Konsistenz**. *Dies ist die Eigenschaft eines numerischen Verfahrens, die bedeutet, dass der Algorithmus (in einer noch näher zu bestimmenden Art und Weise) tatsächlich das gegebene Problem löst und nicht ein anderes. Wir werden dies später noch insbesondere im Zusammenhang mit der numerischen Lösung von gewöhnlichen Differentialgleichungen untersuchen, aber man mag schon jetzt festhalten, dass Stabilität und Konsistenz für die Konvergenz eines Verfahrens wichtig sind.*

Bemerkung 2.5.3 (Standardfehler) *Typisch für das Entstehen numerischer Instabilität ist, dass man das Ausgangsproblem φ in zwei Teilschritte $\varphi = \varphi^{(2)} \circ \varphi^{(1)}$ zerlegt, von denen einer erheblich schlechter konditioniert ist als das Ausgangsproblem.*

Beispiel 2.5.4 (Quadratische Gleichung) Wir untersuchen zwei verschiedene Verfahren zur Lösung von (2.21) in Beispiel 2.4.3

$$\varphi(p, q) = -p + \sqrt{p^2 + q}, \;\; x^2 + 2px - q = 0$$

Methode 2.5.5

$$u = \sqrt{p^2 + q}, \;\; y = \chi_1(p, u) = -p + u. \tag{2.24}$$

Falls $u \approx p$, d.h. falls $p \gg q$, sind die Konditionszahlen von χ_1 erheblich größer als 1 (Auslöschung).

Methode 2.5.6

$$u = \sqrt{p^2 + q}, \;\; y = \chi_2(p, q, u) = \frac{q}{p + u}. \tag{2.25}$$

Die Konditionszahlen von χ_2 sind kleiner als 1. (Das Verfahren beruht darauf, dass $-p - u$ die andere Lösung und das Produkt der beiden Lösungen gleich $-q$ ist (Satz von Viëta).)
Es stellt sich heraus, dass Algorithmus 2.5.5 numerisch instabil, aber Algorithmus 2.5.6 numerisch stabil ist.

Bemerkung 2.5.7 *Man beachte beim Algorithmus 2.5.5 in Beispiel 2.5.4, dass die numerische Auswertung der Funktion χ_1 für sich genommen (trotz Auslöschung) nicht numerisch instabil ist! Schlecht konditioniert ist hier das Problem, $\chi_1(p, \sqrt{p^2 + q})$ zu berechnen. Umgekehrt kann man daran sehen, dass das Zusammensetzen zweier gut konditionierter Algorithmen sehr wohl einen schlecht konditionierten Algorithmus für das Gesamtproblem ergeben kann. Diese Tatsache steht in unangenehmem, aber unvermeidlichem Konstrast zu dem Wunsch, ein Problem in unabhängig voneinander zu bearbeitende Teilprobleme zu zerlegen.*

2.5.1 Vorwärtsanalyse

Die sogenannte Vorwärtsanalyse besteht darin, die Rundungsfehler im Laufe einer Rechnung durch Anwendung der Formeln (2.5) und (2.7) (und weiterer entsprechender) für jeden einzelnen Rechenschritt abzuschätzen. Man erhält dadurch eine obere Schranke für den gesamten durch die Rundungen bedingten Fehler. Kennt man außerdem die Konditionszahlen des Problems, so kann man auf diese Weise (theoretisch wenigstens) feststellen, ob der Algorithmus gut oder schlecht konditioniert ist.

Beispiel 2.5.8 Berechnung des Skalarprodukts $s = \langle x, y\rangle$ für $x, y \in \mathbb{R}^n$ mit dem Verfahren

$$s := 0; \textbf{ for } \quad k := 1 \textbf{ to } \ n \textbf{ do } \ s := s + x_k \cdot y_k; \tag{2.26}$$

Notation 2.5.9 *Für $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ definieren wir $|x| \in \mathbb{R}^n$ durch*

$$|x| = (|x_1|, \ldots, |x_n|). \tag{2.27}$$

Für die Matrix $A \in \mathbb{R}^{m\times n}$ definieren wir die Matrix $|A| \in \mathbb{R}^{m\times n}$ durch

$$|A|_{ij} = |a_{ij}|, \ \ 1 \le j \le n. \tag{2.28}$$

Satz 2.5.10 (Vorwärtsanalyse des Rundungsfehlers für das Skalarprodukt)
Sei $0 < \ eps < 1/n$. Dann gilt

$$|s - \langle x, y\rangle| \le \frac{n\, eps}{1 - n\, eps}\, \langle |x|, |y|\rangle. \tag{2.29}$$

Beweis. Wegen (2.5) lassen sich die Zwischenergebnisse s_k schreiben als

$$s_1 = x_1 y_1 (1 + \delta_1) \qquad , |\delta_1| \le \text{eps}, \tag{2.30}$$

$$s_k = [s_{k-1} + x_k y_k (1 + \delta_k)]\,(1 + \varepsilon_k) \quad , |\delta_k|,\ |\varepsilon_k| \le \text{eps}, \tag{2.31}$$

($x_i \odot y_i = x_i y_i (1 + \delta_i)$, $x_i \oplus y_i = (x_i + y_i)(1 + \varepsilon_i)$) also gilt für $s = s_n$

$$s - \langle x, y\rangle = \sum_{k=1}^{n} x_k y_k \left[\left((1 + \delta_k)\Pi_{j=k}^{n}(1 + \varepsilon_j)\right) - 1\right],\ \varepsilon_1 = 0. \tag{2.32}$$

Die Behauptung folgt aus (2.32) mit der Bernoullischen-Ungleichung

$$|\left((1 + \delta_k)\Pi_{j=k}^{n}(1 + \varepsilon_j)\right) - 1| \le |(1 + eps)^n - 1| \le \left|\frac{1}{(1 - eps)^n} - 1\right| \tag{2.33}$$

$$\le \left|\frac{1}{1 - n\, eps} - 1\right| \le \left|\frac{n\, eps}{1 - n\, eps}\right|.$$

□

Aus der Formel (2.29) ergibt sich

$$\left|\frac{s - \langle x, y\rangle}{\langle x, y\rangle}\right| \le \frac{\langle |x|, |y|\rangle}{|\langle x, y\rangle|}\, |\frac{n\, eps}{1 - n\, eps}|, \tag{2.34}$$

d.h. bei gegebenem $n\, eps$ wird der relative Fehler groß, falls der Winkel zwischen x und y nahe bei 90 Grad ist.

Bemerkung 2.5.11 (Intervallarithmetik) *In der Intervallarithmetik wird jede Zahl $x \in \mathbb{M}'$ durch ein (mittels zweier Gleitpunktzahlen) definiertes Intervall dargestellt, welches x einschließt. Die arithmetischen Operationen werden für solche Intervalle so definiert, dass das resultierende Intervall das exakte Ergebnis der Operation einschließt (und zwar für jede Kombination von Operationen in den Ausgangsintervallen). Falls die Hardware die Intervallarithmetik nicht unterstützt, muss sie softwareseitig simuliert werden (was die Rechnung natürlich stark verlangsamt). In der eben beschriebenen einfachen Form ist sie allerdings für die meisten Probleme der Praxis unbrauchbar, da die Intervalle unrealistisch groß werden, d.h. das mit normaler Gleitpunktarithmetik berechnete Ergebnis ist erheblich genauer.*

Bemerkung 2.5.12 (Hochgenauigkeitsarithmetik) *Die Grundlage der sogenannten Hochgenauigkeitsarithmetik besteht darin, das Skalarprodukt bis auf Maschinengenauigkeit auszurechnen, d.h. zu erreichen, dass der berechnete Wert s des Skalarprodukts die Eigenschaft*

$$s = fl(\langle x, y\rangle) \tag{2.35}$$

hat. Hierzu wird hardwareseitig ein langer Akkumulator benötigt, d.h. ein spezielles Register, welches einen festen Dezimalpunkt und so viele Stellen hat, dass jede Gleitkommazahl in $\mathbb{M}$ dargestellt werden kann. Hält man die Zwischensummen s aus (2.26) in diesem Akkumulator, so kann $\langle x, y\rangle$ exakt berechnet werden, falls $x_i, y_i \in \mathbb{M}$. Der Rundungsfehler entsteht dann beim Abspeichern von s als Gleitkommazahl.
In Verbindung mit Intervallarithmetik lassen sich Algorithmen konstruieren, die bereits für einen recht breiten Anwendungsbereich sehr gute Einschließungen und damit genaue und sichere numerische Ergebnisse liefern. Ein wesentliches Problem ist die benötigte Hardwareunterstützung. Die prinzipiellen Schwierigkeiten schlecht konditionierter Probleme werden natürlich nicht beseitigt, insbesondere auch nicht die Auswirkungen des „Standardfehlers" (Zerlegung eines gut konditionierten Problems in schlecht konditionierte Teilschritte).

2.5.2 Rückwärtsanalyse

Während in der Vorwärtsanalyse die Kondition eines Algorithmus durch Vergleich der Fehlerverstärkung des Algorithmus (durch Abschätzung der akkumulierten Rundungsfehler) mit den Konditionszahlen des Problems ermittelt wird, geht man in der Rückwärtsanalyse folgendermaßen vor : Sei $\varphi : X \to Y$ ein Problem und $\tilde{\varphi} : X \to Y$ ein Algorithmus. Zu gegebenen $x \in X$ sucht man ein $\tilde{x} \in X$ mit

$$\tilde{\varphi}(x) = \varphi(\tilde{x}). \tag{2.36}$$

Gilt dann

$$\left|\frac{\tilde{x} - x}{x}\right| \leq Cn \text{ eps}, \tag{2.37}$$

mit einer kleinen Konstante C (n repräsentiert in einem geeigneten Sinn die Problemgröße), so ist man zufrieden, da sich das berechnete Ergebnis $\tilde{\varphi}(x)$ interpretieren lässt als das Ergebnis einer exakten Rechnung auf mit kleinem relativen Fehler behafteten Daten.

Definition 2.5.13 (Alternative Definition der numerischen Stabilität) *Ein Algorithmus heißt gut konditioniert (oder numerisch stabil), falls er (2.36), (2.37) erfüllt mit einer kleinen Konstanten C.*

Beispiel 2.5.14 (Rückwärtsanalyse des Skalarprodukts) Die Rechnung im Beweis von Satz 2.5.10 zeigt, dass für das mit dem Verfahren (2.26) berechnete Skalarprodukt s gilt

$$s = \langle x, \tilde{y}\rangle,\ \tilde{y}_k = y_k(1+\delta_k)\Pi_{j=k}^{n}(1+\varepsilon_k), \tag{2.38}$$

also mit (2.33)

$$\left|\frac{\tilde{y}_k - y_k}{y_k}\right| \leq \frac{n\,eps}{1 - n\,eps},\ \text{ falls }\ eps \leq 1/n. \tag{2.39}$$

Bemerkung 2.5.15

i.) Also ist das Verfahren (2.26) gut konditioniert (nochmal: das Problem, das Skalarprodukt zu berechnen, kann schlecht konditioniert sein, wenn die Konditionszahlen des Skalarprodukts groß sind).

ii.) Man beachte, dass die Rückwärtsanalyse prinzipiell nichts über die Kondition des Problems aussagt.

iii.) Die Rückwärtsanalyse hat sich als recht brauchbares Werkzeug der Rundungsfehleranalyse erwiesen, und zwar vor allem bei numerischen Verfahren der Linearen Algebra.

3 Fehlerabschätzung mittels Kondition

3.1 Absolute und relative Fehler

Es sei $\widetilde{x} \in \mathbb{R}^n$ eine Approximation von $x \in \mathbb{R}^n$. Als **Fehler** bezeichnet man $e = x - \widetilde{x}$ und zu einer gegebenen Vektornorm $\|\cdot\|$ ist

$$\|x - \widetilde{x}\|$$

der **absolute Fehler** von $\widetilde{x}$. Es sei $x \neq 0$, dann beschreibt

$$\frac{\|x - \widetilde{x}\|}{\|x\|}$$

den **relativen Fehler** von $\widetilde{x}$. Relative Fehler in der ∞-Norm können durch eine Aussage über die Anzahl korrekter Stellen von $\widetilde{x}$ ausgedrückt werden, d.h.

$$\frac{\|x - \widetilde{x}\|_\infty}{\|x\|_\infty} \approx 10^{-p},$$

falls die betragsgrößte Komponente von $\widetilde{x}$ näherungsweise p korrekte signifikante Stellen hat.

3.2 Fehlerabschätzung & Kondition

Wir wollen nun zwei wichtige Fragestellungen untersuchen, welche bei dem numerischen Lösen von linearen Gleichungssystemen (LGS) auftreten. Zum einen betrachten wir eine Näherungslösung $\widetilde{x}$ zu der Lösung x von $Ax = b$. Welche Rückschlüsse kann man von der Größe des **Residuums** $r := b - A\widetilde{x}$ auf den Fehler $e = x - \widetilde{x}$ ziehen? Des Weiteren rechnet man im Allgemeinen mit endlicher Genauigkeit auf einem Rechner. Welche Einflüsse haben dabei Störungen der Ausgangsdaten auf die Lösung x?

Sei $\|A\|$ eine beliebige Matrixnorm und $\|x\|$ eine dazu verträgliche Vektornorm (d.h. $\|Ax\| \leq \|A\|\,\|x\|$). Nach Definition des Fehlers e und Residuums r gilt

$$Ae = Ax - A\widetilde{x} = b - A\widetilde{x} = r.$$

Daraus folgt

$$\|e\| = \|A^{-1}r\| \leq \|A^{-1}\|\,\|r\|.$$

Mit

$$\|b\| = \|Ax\| \leq \|A\|\,\|x\|$$

folgt für den relativen Fehler die Abschätzung

$$\frac{\|e\|}{\|x\|} \leq \frac{\|A^{-1}\|\,\|r\|}{\|b\| \cdot \frac{1}{\|A\|}} = \|A^{-1}\|\,\|A\| \cdot \frac{\|r\|}{\|b\|}\,. \tag{3.1}$$

Dies motiviert den Begriff der Konditionszahl.

Definition 3.2.1 (Konditionszahl) *Man bezeichnet*

$$\kappa(A) = \|A\| \, \|A^{-1}\| \tag{3.2}$$

als die **Konditionszahl** *der Matrix A bezüglich der verwendeten Matrixnorm.*

Beispiel 3.2.2 Man betrachte das LGS $Ax = b$ mit

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix} \qquad b := (b_j) \qquad b_j := \sum_{k=1}^{5} \frac{1}{k+j-1} \qquad (1 \le j \le 5)\,.$$

Für die Matrixnormen und die Konditionszahlen erhält man in den Fällen $p = 1, 2, \infty$

$$\begin{array}{lcllcllcl} \|A\|_1 &=& 2.28\overline{3}\,, & \|A^{-1}\|_1 &=& 413280\,, & \kappa_1(A) &=& 943656\,, \\ \|A\|_2 &=& 1.567051\,, & \|A^{-1}\|_2 &=& 304142.84\,, & \kappa_2(A) &=& 476607.34\,, \\ \|A\|_\infty &=& 2.28\overline{3}\,, & \|A^{-1}\|_\infty &=& 413280\,, & \kappa_\infty(A) &=& \kappa_1(A)\,. \end{array}$$

Sei die exakte Lösung $x = (1, \ldots, 1)^T$. Ist nun $\widetilde{x} = (1-\varepsilon)x$, dann gilt

$$\frac{\|e\|}{\|x\|} = \frac{\|\varepsilon x\|}{\|x\|} = |\varepsilon|$$

und

$$r = b - A\widetilde{x} = Ax - A\widetilde{x} = A(x - (1-\varepsilon)x) = \varepsilon Ax = \varepsilon b,$$

d.h.

$$\frac{\|r\|}{\|b\|} = |\varepsilon|\,.$$

Obwohl die Konditionzahl sehr groß ist, verhält sich bei dieser Störung $\frac{\|e\|}{\|x\|}$ wie $\frac{\|r\|}{\|b\|}$. Obige Abschätzung ist offensichtlich eine „worst case“ Abschätzug. Betrachten wir nun

$$\widetilde{x} = (0.993826,\ 1.116692,\ 0.493836,\ 1.767191,\ 0.623754)^T. \tag{3.3}$$

(A habe die Eigenwerte $0 \le \lambda_1 < \cdots < \lambda_5$ mit den zugehörigen Eigenvektoren $\varphi_1, \ldots, \varphi_5$ ($\|\varphi_j\|_2 = 1$). Man beachte, dass $\widetilde{x}$ in (3.3) so gewählt ist, dass $x - \widetilde{x} \approx \varphi_1$ gilt und x näherungsweise φ_5 entspricht.) Dann erhalten wir

$$\frac{\|e\|_1\|b\|_1}{\|x\|_1\|r\|_1} \approx 0.41\ \kappa_1(A), \quad \frac{\|e\|_2\|b\|_2}{\|x\|_2\|r\|_2} \approx 0.89\ \kappa_2(A), \quad \frac{\|e\|_\infty\|b\|_\infty}{\|x\|_\infty\|r\|_\infty} \approx 0.74\ \kappa_\infty(A)$$

und schätzen dann in der richtigen Größenordung ab!

Aufgabe 3.2.3 Es sei $A \in \mathbb{R}^{n\times n}$ eine positiv definite Matrix. Die Eigenwerte von Matrix A seien $0 < \lambda_1 \le \ldots \le \lambda_n$ mit den zugehörigen Eigenvektoren $\varphi_1, \ldots, \varphi_n$ ($\|\varphi_j\|_2 = 1$).

i) Es sei $x \in \mathbb{R}^n \setminus \{0\}$, $b := Ax$ und $\widetilde{x} = cx$ ($0 \ne c \in \mathbb{R}$). Man zeige

$$\frac{\|x - \widetilde{x}\|}{\|x\|} \, \frac{\|b\|}{\|b - A\widetilde{x}\|} = 1\,.$$

ii) Man zeige

$$\kappa_2(A) = \lambda_n/\lambda_1\,.$$

iii) Es sei $x = \varphi_n$, $b := Ax$ und $\widetilde{x} = x + \varphi_1$. Man zeige

$$\frac{\|x - \widetilde{x}\|}{\|x\|} \frac{\|b\|}{\|b - A\widetilde{x}\|} = \kappa_2(A)\,.$$

Untersuchen wir nun, welchen Einfluss kleine Störungen in den Ausgangsdaten A, b auf die Lösung x des linearen Gleichungssystems haben können, d.h. wir sind interessiert an der **Empfindlichkeit** der Lösung x auf Störungen in den Koeffizienten. Die genaue Frage lautet:
Wie groß kann die Änderung δx der Lösung x von $Ax = b$ sein, falls die Matrix um δA und b durch δb gestört sind. Dabei seien δA und δb kleine Störungen, so daß $A + \delta A$ immer noch regulär ist. Es sei $x + \delta x$ die Lösung zu

$$(A + \delta A)(x + \delta x) = (b + \delta b).$$

Teilweises Ausmultiplizieren liefert

$$Ax + \delta Ax + (A + \delta A)\delta x = b + \delta b$$

und somit mit $Ax = b$

$$\begin{aligned}\delta x &= (A + \delta A)^{-1}(\delta b - \delta Ax)\\ &= (I + A^{-1}\delta A)^{-1}A^{-1}(\delta b - \delta Ax).\end{aligned}$$

Für eine submultiplikative Matrixnorm und eine dazu verträgliche Vektornorm ergibt sich somit

$$\|\delta x\| \le \|(I + A^{-1}\delta A)^{-1}\|\ \|A^{-1}\|\ (\|\delta b\| + \|\delta A\|\ \|x\|)$$

und wir erhalten die Abschätzung

$$\frac{\|\delta x\|}{\|x\|} \le \|(I + A^{-1}\delta A)^{-1}\|\ \|A^{-1}\| \left(\frac{\|\delta b\|}{\|x\|} + \|\delta A\|\right). \tag{3.4}$$

Wir zeigen in Lemma 3.2.6, dass für $\|B\| < 1$ die Abschätzung

$$\|(I + B)^{-1}\| \le \frac{1}{1 - \|B\|} \qquad \text{existiert.}$$

Setzen wir nun $B = A^{-1}\delta A$ und $\|A^{-1}\|\,\|\delta A\| < 1$ (d.h. δA ist eine kleine Störung von A) voraus, erweitern auf der rechten Seite mit $\|A\|$, so erhalten wir unter Ausnutzung der Vertäglichkeit der Normen und $Ax = b$

$$\begin{aligned}\frac{\|\delta x\|}{\|x\|} &\le \frac{\|A^{-1}\|\ \|A\|}{1 - \|A^{-1}\delta A\|} \left(\frac{\|\delta b\|}{\|A\|\ \|x\|} + \frac{\|\delta A\|}{\|A\|}\right)\\ &\le \frac{\|A^{-1}\|\ \|A\|}{1 - \|A^{-1}\|\,\|\delta A\|} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right).\end{aligned}$$

Wir halten das Ergebnis nun in einem Satz fest.

Satz 3.2.4 *Es sei $A \in \mathbb{R}^{n\times n}$ regulär und $x \in \mathbb{R}^n$ die exakte Lösung von $Ax = b$. Die rechte Seite b sei um δb gestört und für die Störung von A gelte $\kappa(A)\,\|\delta A\|/\|A\| = \|A^{-1}\|\,\|\delta A\| < 1$ und $A + \delta A$ ist immer noch regulär. Dann gilt für die Lösung $\widetilde{x}$ des gestörten Systems mit Koeffizientenmatrix $A + \delta A$ und rechter Seite $b + \delta b$*

$$\frac{\|x - \widetilde{x}\|}{\|x\|} \le \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right). \tag{3.5}$$

Bemerkung 3.2.5 *Die Konditionszahl $\kappa(A)$ der Koeffizientenmatrix A ist folglich die entscheidende Größe, welche die Empfindlichkeit der Lösung bzgl. der Störungen δA und δb beschreibt.*

Es bleibt folgendes Lemma zu beweisen:

Lemma 3.2.6 (Neumann-Reihe) *Sei $A \in \mathbb{R}^{n\times n}$, $\|\cdot\|$ eine submultiplikative Matrixnorm und $\|A\| < 1$. Dann ist $(I-A)$ regulär und*

$$(I-A)^{-1} = \sum_{k=0}^{\infty} A^k \qquad \textit{(Neumann-Reihe)} \tag{3.6}$$

mit

$$\|(I-A)^{-1}\| \leq \frac{1}{1-\|A\|}\,. \tag{3.7}$$

Beweis. Mit $\|A\| < 1$, der Submultiplikativität und der Summenformel für die geometrische Reihe erhält man

$$\sum_{k=0}^{m} \|A^k\| \leq \sum_{k=0}^{m} \|A\|^k \leq \sum_{k=0}^{\infty} \|A\|^k = \frac{1}{1-\|A\|} < \infty \quad (m \in \mathbb{N})\,. \tag{3.8}$$

Der Raum $\mathbb{R}^{n\times n}$ ist isomorph zu $\mathbb{R}^{n^2}$ (siehe Anhang A.2). In [Analysis II, Beispiel 8.4.6] wurde gezeigt, dass $\mathbb{R}^{n^2}$ vollständig ist bezüglich irgendeiner Norm auf $\mathbb{R}^{n^2}$. Somit ist $\mathbb{R}^{n\times n}$ vollständig bezüglich $\|\cdot\|$ und aus der absoluten Konvergenz folgt die Konvergenz von $\sum_{k=0}^{\infty} A^k$. Ebenso folgt aus $\|A^k\| \leq \|A\|^k \to 0$ für $k \to \infty$ die Konvergenz von $\lim\limits_{k\to\infty} A^k = 0$. Weiter gilt die Gleichung („Teleskopsumme “)

$$\left(\sum_{k=0}^{m} A^k\right)(I-A) = I - A^{m+1} \quad (m \in \mathbb{N})\,. \tag{3.9}$$

Der Grenzübergang von (3.9) führt zur Gleichung

$$\left(\sum_{k=0}^{\infty} A^k\right)(I-A) = I. \tag{3.10}$$

Das bedeutet, $I-A$ ist regulär und $(I-A)^{-1} = \sum_{k=0}^{\infty} A^k$. Zusammen mit 3.8 wäre damit die Behauptung bewiesen. Mit der Summenformel für die geometrische Reihe erhält man nun

$$\|(I-A)^{-1}\| \leq \lim_{N\to\infty} \sum_{k=0}^{N} \|A^k\| \leq \lim_{N\to\infty} \sum_{k=0}^{N} \|A\|^k = \frac{1}{1-\|A\|}\,, \tag{3.11}$$

womit der Satz bewiesen ist. □

Zum Ende des Kapitels wollen wir nun den praktischen Nutzen der Abschätzung (3.5) in einer Faustregel festhalten.

Bei einer d-stelligen dezimalen Gleitkommarechnung können die relativen Fehler der Ausgangsgrößen für beliebige, kompatible Normen von der Größenordnung

$$\frac{\|\delta A\|}{\|A\|} \approx 5\cdot 10^{-d} \qquad \frac{\|\delta b\|}{\|b\|} \approx 5\cdot 10^{-d}$$

sein. Ist die Konditionszahl $\kappa(A) \approx 10^{\alpha}$ mit $5\cdot 10^{\alpha-d} \ll 1$, so ergibt die Abschätzung

$$\frac{\|\delta x\|}{\|x\|} \leq 10^{\alpha-d+1}$$

Mit der Aussage zu (3.1) besagt die Schätzung, dass $\|\delta x\|$ maximal in der Größenordnung der $(d-\alpha-1)$-ten Dezimalstelle von $\|x\|$ liegen kann und dies motiviert folgende Daumenregel:

Bemerkung 3.2.7 (Daumenregel zur Genauigkeit) *Wird $Ax = b$ mit d-stelliger dezimaler Gleitkommarechnung gelöst, und beträgt die Konditionszahl $\kappa(A) \approx 10^{\alpha}$, so sind, bezogen auf die betragsgrößte Komponente, nur $(d - \alpha - 1)$ Dezimalstellen sicher.*

MATLAB-Beispiel:

Es ist bekannt, dass die Gauß-Elemination selbst mit Spaltenpivotstrategie zu überraschend ungenauen Ergebnissen beim Lösen von linearen Gleichungssystemen führen kann, obwohl die Matrix gut konditioniert ist.

Betrachten wir hierzu die von Wilkinson angegebene pathologische Matrix

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

```
>> A=toeplitz([1,-ones(1,59)], ...
              [1,zeros(1,59)]);
>> A(:,60)=1;
>> cond(A)
ans =
   26.8035  % rel. gut konditioniert
>> randn('state', 3383)
>> x=randn(60,1);
>> b=A*x;
>> x1=A\b;
>> norm(x-x1)/norm(x)
ans =
    0.3402  % großer rel. Fehler
```

Bemerkung 3.2.8 *Das Beispiel lässt vermuten, dass das Gauß-Verfahren über die ganze Menge der invertierbaren Matrizen betrachtet nicht stabil ist. Für die in der Praxis auftretenden Matrizen, ist das Gauß-Verfahren mit Spaltenpivotierung jedoch „in der Regel“ stabil. Für eine weitere Stabilitätsanalyse des Gauß-Verfahrens sei auf [Deuflhard/Hohmann], die grundlegenden Artikel von Wilkinson [Wilkinson65, Wilkinson69] sowie auf die Übersichtsartikel [Higham, Discroll/Maki] verwiesen.*

4 ITERATIVE LÖSUNG LINEARER GLEICHUNGSSYSTEME

Gegeben sei eine reguläre Matrix $A \in \mathbb{R}^{n\times n}$ und ein lineares Gleichungssystem

$$Ax = b$$

mit der exakten Lösung x. Wir betrachten Iterationsverfahren der Form

$$x^{(i+1)} = \phi(x^{(i)}),\ \ i = 0, 1, \ldots$$

Mit Hilfe einer beliebigen regulären Matrix $B \in \mathbb{R}^{n\times n}$ erhält man solche Iterationsvorschriften aus der Gleichung

$$Bx + (A - B)x = b\,,$$

indem man

$$Bx^{(i+1)} + (A - B)x^{(i)} = b$$

setzt und nach $x^{(i+1)}$ auflöst

$$\begin{aligned} x^{(i+1)} &= x^{(i)} - B^{-1}(Ax^{(i)} - b) \\ &= (I - B^{-1}A)x^{(i)} + B^{-1}b\,. \end{aligned} \tag{4.1}$$

Jede Wahl einer nichtsingulären Matrix B führt zu einem möglichen **Iterationsverfahren**. Es wird umso brauchbarer, je besser B die folgenden Kriterien erfüllt

i) B ist leicht zu invertieren (**einfache Realisierbarkeit**)

ii) die Eigenwerte von $(I - B^{-1}A)$ sollen möglichst kleine Beträge haben. (**Konvergenzeigenschaft**)

Wir wollen hier nun einige Beispiele angeben. Dazu verwenden wir folgende (additive) Standardzerlegung

$$A = L + D + R\,,$$

wobei $\boldsymbol{D}$ eine **Diagonalmatrix**, $\boldsymbol{L}$ strikte **untere Dreiecksmatrix** und $\boldsymbol{R}$ strikte **obere Dreiecksmatrix** seien. Die Wahl

i) $B = \gamma I$ liefert das **Richardson**-Verfahren

ii) $B = D$ liefert das **Jacobi**-Verfahren (**Gesamtschrittverfahren**)

iii) $B = L + D$ oder $B = D + R$ liefert das **Gauß-Seidel**-Verfahren (**Einzelschrittverfahren**)

Was zeichnet nun die einzelnen Verfahren aus? Betrachten wir dazu ein Beispiel.

Beispiel 4.0.1 Zu gegebenem $n \in \mathbb{N}$ und

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}, b = \vec{1} \in \mathbb{R}^n, x^{(0)} = \vec{0} \in \mathbb{R}^n$$

bestimmen wir die **Konvergenzrate**

$$c = \max_k \frac{\|x - x^{(k+1)}\|_2}{\|x - x^{(k)}\|_2},$$

für das Jacobi- und das Gauß-Seidel-Verfahren, d.h. den Faktor, um den der **Fehler** in der 2-Norm in jedem Iterationsschritt **mindestens reduziert** wird.

MATLAB-Funktion: runKonvergenz.m

```
n = 10;
e = ones(n,1);
A = spdiags([e -2*e e], -1:1, n, n);
x_ex = rand(n,1);          % exakte Loesung
b = A * x_ex;              % exakte rechte Seite
x{1} = rand(n,1);          % zufaelliger Startv.
x{2}=x{1};
W{1} = triu(A);            % Gauss-Seidel
W{2} = diag(diag(A));      % Jacobi
for j = 1:length(x)
  error_old = norm(x{j}-x_ex);
  for k = 1 : 20
      x{j} = x{j} + W{j} \ (b-A*x{j});
      error_new = norm(x{j}-x_ex);
      quot{j}(k) = error_new/error_old;
      error_old = error_new;
  end
end
plot(1:20,quot{1},'m-s',1:20,quot{2},'k:*');
xlabel('Anzahl der Iterationen'), ylabel('Kontraktion')
legend({'Gauss-Seidel-Verf.','Jacobi-Verfahren'},4)
```

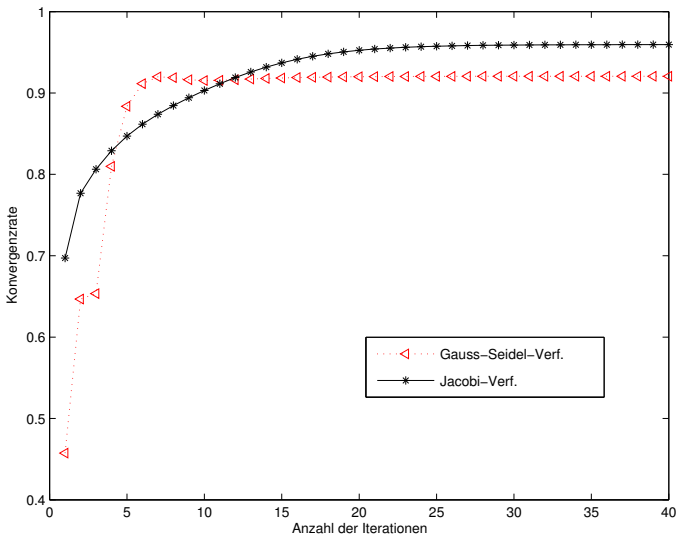


Abb. 4.1: *Kontraktionszahlen für Gesamt- und Einzelschrittverfahren.*

Dem numerischen Experiment kann man entnehmen, dass beide Verfahren konvergieren, da die Konvergenzerate jeweils kleiner 1 ist, und dass das Gauß-Seidel-Verfahren schneller konvergiert, da die Konvergenzrate hier kleiner ist.

Bemerkung 4.0.2 *Iterationsverfahren finden besonders Anwendung bei schwachbesetzten Matrizen, d.h. die Matrix hat „viele Nulleinträge". Ist die Hülle jedoch groß und in jeder Zeile treten nur wenige Nichtnulleinträge auf, dann ist ein Bandlöser somit „teuer"und die folgenden Verfahren liefern gute Alternativen.*

Bei dem Gauß-Seidel-Verfahren verwendet man in der praktischen Anwendung folgende Formulierung um die Anzahl der Operatoren zu reduzieren

$$x^{(i+1)} = (L+D)^{-1}(b - Rx^{(i)})$$

da

$$\begin{aligned} x^{(i+1)} &= x^{(i)} - B^{-1}(Ax^{(i)} - b) \\ &= x^{(i)} - B^{-1}([(A-B)+B]\,x^{(i)} - b) \\ &= x^{(i)} - B^{-1}(A-B)x^{(i)} - B^{-1}Bx^{(i)} + B^{-1}b \\ &= B^{-1}(b - (A-B)x^{(i)}) \end{aligned}$$

gilt und mit $B = L + D$ und $A - B = R$ folgt

$$x^{(i+1)} = (L+D)^{-1}(b - Rx^{(i)})\,.$$

Ein Schritt eines Gauß-Seidel-Verfahrens ist also etwa so aufwendig wie eine Matrix-Vektor-Multiplikation. Ähnlich kann man auch bei dem Jacobi-Verfahren vorgehen, d.h.

$$x^{(i+1)} = D^{-1}(b - (L+R)x^{(i)})$$

und das Richardson-Verfahren vereinfacht man zu

$$x^{(i+1)} = x^{(i)} + \frac{1}{\gamma}(b - Ax^{(i)})\,.$$

4.1 Folgen von Iterationsmatrizen

Es sei x Lösung von $Ax = b$. Mit (4.1) erhalten wir

$$\begin{aligned} x - x^{(k)} &= x - B^{-1}b - (I - B^{-1}A)x^{(k-1)} \\ &= Ix - B^{-1}Ax - (I - B^{-1}A)x^{(k-1)} \\ &= (I - B^{-1}A)(x - x^{(k-1)}) = \ldots = (I - B^{-1}A)^k(x - x^{(0)})\,. \end{aligned}$$

Die Konvergenz des dargestellten Verfahrens hängt also nur von den Eigenschaften der Iterationsmatrix $I - B^{-1}A$ ab. Es sei C eine beliebige komplexwertige $(n \times n)$-Matrix, $\lambda_i := \lambda_i(C)$ $(i = 1, \ldots, n)$ seien die Eigenwerte von C. Dann bezeichnen wir mit

$$\varrho(C) := \max_{1 \le i \le n} \{|\lambda_i(C)|\}$$

den **Spektralradius** von C. Bevor wir ein Konvergenzkriterium angeben, bereiten wir noch den Begriff der Jordanschen Normalform vor.

Definition 4.1.1 (Jordan-, bzw. Elementarmatrix) *Eine Matrix $E_k(\lambda) \in \mathbb{C}^{k\times k}$ heißt Jordanmatrix (oder Elementarmatrix) zum Eigenwert λ, wenn*

$$E_k(\lambda) = \begin{pmatrix} \lambda & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda \end{pmatrix} . \tag{4.2}$$

Satz 4.1.2 (Jordansche Normalform (siehe z.B. [Fischer])) *Zu jeder Matrix $A \in \mathbb{C}^{n\times n}$ existiert eine reguläre Matrix $T \in \mathbb{C}^{n\times n}$, so dass*

$$A = T^{-1}JT\,,$$

wobei J, die durch die Paare $(\lambda_1, n_1), \ldots, (\lambda_k, n_k)$ mit $\lambda_i \in \mathbb{C}$, $n_i \geq 1$ (eindeutig bis auf die Reihenfolge) bestimmte Jordansche Normalform

$$J = \begin{pmatrix} E_{n_1}(\lambda_1) & 0 & \\ & \ddots & \\ 0 & & E_{n_k}(\lambda_k) \end{pmatrix}$$

von A ist.

Satz 4.1.3 (Konvergenzkriterium) *Es sei $C \in \mathbb{C}^{n\times n}$. Die Folge $(C^k)_{k\in\mathbb{N}}$ ist genau dann eine Nullfolge, wenn $\varrho(C) < 1$ gilt.*

Beweis. Sei zunächst $\varrho(C) \geq 1$. Dann gibt es einen Eigenwert λ mit $|\lambda| \geq 1$ und einen Vektor $x \neq 0$ mit $Cx = \lambda x$. Wegen $C^k x = \lambda^k x$ und $\lim_{k\to\infty} \lambda^k \neq 0$ kann folglich $(C^k)_k$ keine Nullfolge sein. Die Bedingung $\varrho(C) < 1$ ist somit notwendig.
Sei nun $\varrho(C) < 1$. Weil $(TCT^{-1})^k = TC^kT^{-1}$ für jede Ähnlichkeitstransformation T gilt, reicht es, $\lim_{k\to\infty}(TCT^{-1})^k = 0$ zu zeigen. Die Matrix C lässt sich durch Ähnlichkeitstransformation auf die Jordansche Normalform J transformieren. Wir zeigen, daß $\lim_{k\to\infty} J^k = 0$ gilt, wenn alle Eigenwerte $\lambda_1, \ldots, \lambda_n$ dem Betrag nach kleiner Eins sind. Dazu sei

$$E_\mu = E_{n_\mu}(\lambda_\mu) = \begin{pmatrix} \lambda_\mu & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda_\mu \end{pmatrix} \in \mathbb{C}^{n_\mu\times n_\mu}$$

eine Elementarmatrix zum Eigenwert λ_μ der Jordanschen Normalform J von C. Da offenbar

$$J^k = \begin{pmatrix} E_1^k & & & \\ & E_2^k & & \\ & & \ddots & \\ & & & E_\ell^k \end{pmatrix}$$

mit $1 \leq \ell \leq n$ gilt, genügt es, das Konvergenzverhalten einer Jordanmatrix E_μ zu untersuchen. Wir schreiben E_μ in der Form $E_\mu = \lambda_\mu I + S$ mit

$$S = \begin{pmatrix} 0 & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & 0 \end{pmatrix}$$

und bilden $E_\mu^k = (\lambda_\mu I + S)^k$. Nach Anwendung der binomischen Formel und unter Beachtung von $S^n = 0$ erhält man die Beziehung

$$E_\mu^k = \sum_{\nu=0}^{\min\{k,n-1\}} \binom{k}{\nu} \lambda_\mu^{k-\nu} S^\nu.$$

Für feste ν hat man mit

$$\binom{k}{\nu} = \frac{k!}{\nu!(k-\nu)!} = \frac{k(k-1)\cdot\ldots\cdot(k-\nu+1)}{1\cdot\ldots\cdot\nu} \leq k^\nu$$

die Abschätzung

$$\left| \binom{k}{\nu} \lambda_\mu^{k-\nu} \right| \leq |\lambda_\mu^{k-\nu} k^\nu| \,.$$

Da $|\lambda_\mu| < 1$, strebt $k \log|\lambda_\mu| + \nu \log(k) \to -\infty$ für $k \to \infty$ und mit

$$|\lambda_\mu^{k-\nu} k^\nu| \leq \exp((k-\nu)\log|\lambda_\mu| + \nu \log k)$$

folgt die Konvergenz $\lim_{k\to\infty} \left| \binom{k}{\nu} \lambda_\mu^{k-\nu} \right| = 0$. Damit ist gezeigt, dass $(E_\mu^k)_k$ eine Nullfolge ist und somit auch die Behauptung. □

Um die Konvergenz der Richardson-Iteration mathematisch zu beweisen, muss der Spektralradius der Iterationsmatrix bestimmt werden. Dies ist im Allgemeinen nicht analytisch möglich und numerisch sehr aufwendig. Es ist daher das Ziel der nächsten beiden Abschnitte, aus einfachen algebraischen Eigenschaften der Matrix A auf die Konvergenz der Jacobi- bzw. Gauß-Seidel-Iteration zu schließen. Anders als in Satz 4.1.3 sind die folgenden Ergebnisse hinreichende Konvergenzkriterien, im Allgemeinen aber nicht notwendig.

4.2 KONVERGENZ DES JACOBI-VERFAHRENS

Bei dem **Jacobi-Verfahren** (auch Gesamtschrittverfahren genannt) werden alle Komponenten des Lösungsvektors in einem Iterationsschritt gleichzeitig korrigiert. Die Iterationsvorschrift lautet

$$x^{(i+1)} = D^{-1}(b - (L+R)x^{(i)}) \,,$$

d.h. die Iterationsmatrix ist $C_J = I - D^{-1}A = -D^{-1}(L+R)$.

Satz 4.2.1 (Starkes Zeilen- und Spaltensummenkriterium) *Es sei $A \in \mathbb{C}^{n\times n}$. Das Jacobi-Verfahren konvergiert für jeden Startvektor $x^{(0)} \in \mathbb{C}^n$, wenn für die Matrix A das*

i) (starke Zeilensummenkriterium)

$$|a_{ii}| > \sum_{\substack{k=1 \\ k\neq i}}^{n} |a_{ik}|, \quad \textit{für } i = 1, 2, \ldots, n \,,$$

d.h. A ist strikt diagonaldominant, oder das

ii) (starke Spaltensummenkriterium)

$$|a_{kk}| > \sum_{\substack{i=1 \\ i\neq k}}^{n} |a_{ik}|, \quad \textit{für } k = 1, 2, \ldots, n\,,$$

d.h. A^T ist strikt diagonaldominant, erfüllt ist.

Für den Beweis von Satz 4.2.1 benötigen wir das folgende Lemma.

Lemma 4.2.2 *Es sei $A \in \mathbb{C}^{n\times n}$. Dann gilt für jede natürliche p-Matrixnorm $\varrho(A) \leq \|A\|_p$.*

Beweis. Jeder Eigenwert λ von A mit zugehörigem Eigenvektor x genügt für jede natürliche p-Matrixnorm $\|\cdot\|_p$ der Beziehung

$$\frac{\|Ax\|_p}{\|x\|_p} = |\lambda|$$

und damit der Abschätzung $\|A\|_p \geq |\lambda|$. □

Beweis von Satz 4.2.1. i): Die Iterationsmatrix des Jacobi-Verfahrens bezeichnen wir mit $C_J = -D^{-1}(L+R)$. Wenn das starke Zeilensummenkriterium erfüllt ist, gilt die Abschätzung

$$\|C_J\|_\infty = \left\| \begin{pmatrix} 0 & -\frac{a_{1,2}}{a_{1,1}} & \cdots & \cdots & -\frac{a_{1,n}}{a_{1,1}} \\ -\frac{a_{2,1}}{a_{2,2}} & 0 & -\frac{a_{2,3}}{a_{2,2}} & & -\frac{a_{2,n}}{a_{2,2}} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ -\frac{a_{n,1}}{a_{n,n}} & \cdots & \cdots & -\frac{a_{n,n-1}}{a_{nn}} & 0 \end{pmatrix} \right\|_\infty = \max_{1\leq i\leq n} \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} < 1\,.$$

Lemma 4.2.2 liefert dann die Behauptung i).
ii) Ist für A das starke Spaltensummenkriterium (ii) erfüllt, so gilt (i) für A^T. Also konvergiert das Jacobi-Verfahren für A^T und es ist daher wegen Lemma 4.2.2, $\varrho(C) < 1$ für $C = I - D^{-1}A^T$. Nun hat C die gleichen Eigenwerte wie C^T und $D^{-1}C^T D = I - D^{-1}A$. Also ist auch $\varrho(I - D^{-1}A) < 1$, d.h. das Jacobi-Verfahren ist auch für die Matrix A konvergent. □

Definition 4.2.3 *Eine Matrix $A \in \mathbb{R}^{n\times n}$, heißt zerlegbar, wenn es nichtleere Teilmengen N_1 und N_2 der Indexmenge $N := \{1, 2, \ldots, n\}$ gibt mit den Eigenschaften*

i) $N_1 \cap N_2 = \emptyset$

ii) $N_1 \cup N_2 = N$

iii) $a_{ij} = 0$ *für alle* $i \in N_1$ *und* $j \in N_2$

Eine Matrix, die nicht zerlegbar ist, heißt unzerlegbar (irreduzibel).

Beispiel 4.2.4 i)

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1k} & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{N1} & \cdots & a_{NN} & 0 & \cdots & 0 \\ a_{N+1,1} & \cdots & a_{N+1,N} & a_{N+1,N+1} & \cdots & a_{N+1,2N} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{2N,1} & \cdots & a_{2N,N} & a_{2N,N+1} & \cdots & a_{2N,2N} \end{pmatrix}$$

Die Teilmengen $N_1 = \{1, 2, \ldots, N\}\,, N_2 = \{N+1, \ldots, 2N\}$ haben alle in der Definition geforderten Eigenschaften. Somit ist A zerlegbar.

ii) Eine Tridiagonalmatrix mit nicht verschwindenden Nebendiagonal- und Diagonalelementen ist unzerlegbar.

$$A = \begin{pmatrix} 2 & -1 & 0 \\ 1 & 4 & 0 \\ 0 & -1 & 3 \end{pmatrix}, \quad G(A): \quad \circlearrowright K_1 \quad \circlearrowright K_2 \quad \circlearrowright K_3$$

Abb. 4.2: Beispiel einer zerlegbaren Matrix A und ihres Graphen $G(A)$.

Bemerkung 4.2.5 *Dass eine Matrix A unzerlegbar (irreduzibel) ist, kann man häufig leicht mit Hilfe des der Matrix A zugeordneten (gerichteten) Graphen $G(A)$ zeigen. Wenn A eine $n \times n$-Matrix ist, so besteht $G(A)$ aus n Knoten $K_1, \ldots, K_n$ und es gibt eine gerichtete Kante $K_i \to K_j$ in $G(A)$ genau dann, wenn $a_{ij} \neq 0$.Man zeigt leicht, dass A genau dann unzerlegbar ist, falls der Graph $G(A)$ in dem Sinne zusammenhängend ist, dass es für jedes Knotenpaar (K_i, K_j) in $G(A)$ einen gerichteten Weg K_i nach K_j gibt.*

Definition 4.2.6 (Schwaches Zeilensummenkriterium) *Eine Matrix $A \in \mathbb{R}^{n\times n}$ erfüllt das* **schwache Zeilensummenkriterium**, *falls*

$$\sum_{\substack{\nu=1 \\ \nu\neq\mu}}^{n} |a_{\mu\nu}| \leq |a_{\mu\mu}|$$

für alle Zeilen $j = 1, \ldots, n$ gilt, d.h. A ist diagonaldominant, und

$$\sum_{\substack{\nu=1 \\ \nu\neq\mu_0}}^{n} |a_{\mu_0\nu}| < |a_{\mu_0\mu_0}|$$

für mindestens einen Index $\mu_0 \in \{1, \ldots, n\}$ erfüllt ist.
Eine Matrix $A \in \mathbb{R}^{n\times n}$ erfüllt das **schwache Spaltensummenkriterium**, *wenn A^T das schwache Zeilensummenkriterium erfüllt.*

Satz 4.2.7 (Schwaches Zeilensummenkriterium) *Es sei $A \in \mathbb{R}^{n\times n}$ eine unzerlegbare Matrix, die das schwache Zeilensummenkriterium erfüllt. Dann ist das Jacobi-Verfahren konvergent.*

Zum Beweis von Satz 4.2.7 werden wir direkt den Spektralradius der Iterationsmatrix abschätzen. Die wesentliche Beobachtung dabei ist, dass jede irreduzible Matrix, die das schwache Zeilensummenkriterium erfüllt, bereits regulär ist.

Lemma 4.2.8 *Jede irreduzible Matrix $M \in \mathbb{R}^{n\times n}$, die das schwache Zeilensummenkriterium erfüllt, ist regulär und für die Diagonalelemente gilt $m_{jj} \neq 0$ ($j = 1, \ldots, n$).*

Beweis. Wir nehmen an, M sei nicht regulär, d.h. es existiert ein $x \in \mathbb{K}^n \setminus \{0\}$ mit $Mx = 0$. Insbesondere folgt aus der Dreiecksungleichung

$$|m_{jj}|\,|x_j| \leq \sum_{\substack{\ell=1 \\ \ell\neq j}}^{n} |m_{j\ell}|\,|x_\ell| \quad \text{für alle } j = 1, \ldots, n. \tag{4.3}$$

Wir definieren die Indexmengen $J := \{j : |x_j| = \|x\|_\infty\}$ und $K := \{k : |x_k| < \|x\|_\infty\}$. Offensichtlich gilt $J \cap K = \emptyset$, $J \cup K = \{1, \ldots, n\}$ und $J \neq \emptyset$. Wäre $K = \emptyset$, so könnte man in (4.3) die x_j- und x_ℓ-Terme herauskürzen und erhielte einen Widerspruch zum schwachen

Zeilensummenkriterium. Also gilt auch $K \neq \emptyset$, und aufgrund der Irreduzibilität von M existieren Indizes $j \in J$ und $k \in K$ mit $m_{jk} \neq 0$. Mit diesem ergibt sich

$$|m_{jj}| \leq \sum_{\substack{\ell=1 \\ \ell \neq j}}^{n} |m_{j\ell}| \frac{|x_\ell|}{|x_j|} < \sum_{\substack{\ell=1 \\ \ell \neq j}}^{n} |m_{j\ell}|,$$

denn der Quotient ist stets ≤ 1 wegen $|x_j| = \|x\|_\infty$ und er ist < 1 für $\ell = k \in K$. Also erhalten wir einen Widerspruch zum schwachen Zeilensummenkriterium von M, d.h. M ist regulär. Gäbe es schließlich ein triviales Diagonalelement $m_{jj} = 0$, so folgte aus dem schwachen Zeilensummenkriterium, dass bereits die j-te Zeile die Nullzeile wäre. Da M regulär ist, folgt insbesondere $m_{jj} \neq 0$ für alle $j = 1, \ldots, n$. □

Beweis von Satz 4.2.7. Wegen $a_{jj} \neq 0$ für alle $j = 1, \ldots, n$ ist $C_J = -D^{-1}(A - D)$ wohldefiniert. Um $\varrho(C_J) < 1$ zu zeigen, beweisen wir, dass $M := C_J - \lambda I$ für $\lambda \in \mathbb{C}$ mit $|\lambda| \geq 1$ regulär ist. Da A irreduzibel ist, ist auch $A - D$ irreduzibel, denn es wird lediglich die Diagonale verändert. C_J entsteht durch zeilenweise Multiplikation von $A - D$ mit Werten $\neq 0$. Deshalb ist auch C_J irreduzibel. Da M und C_J sich nur auf der Diagonale unterscheiden, ist M irreduzibel. Aufgrund des schwachen Zeilensummenkriteriums von A gilt

$$\sum_{\substack{k=1 \\ k \neq j}}^{n} |m_{jk}| = \sum_{\substack{k=1 \\ k \neq j}}^{n} |c_{jk}^{(J)}| = \sum_{\substack{k=1 \\ k \neq j}}^{n} \frac{|a_{jk}|}{|a_{jj}|} \leq 1 \leq |\lambda| = |m_{jj}| \text{ für alle } j = 1, \ldots, n.$$

und für mindestens einen Index j gilt diese Ungleichung strikt. Also erfüllt M auch das schwache Zeilensummenkriterium und ist nach Lemma 4.2.8 insgesamt regulär. □

Aufgabe 4.2.9 Man zeige, dass das Jacobi-Verfahren auch konvergent ist unter der Annahme, dass $A \in \mathbb{R}^{n \times n}$ irreduzibel ist und das schwache Spaltensummenkriterium erfüllt.

4.3 Konvergenz des Gauss-Seidel-Verfahrens

Die Iterationsvorschrift des **Gauß-Seidel-Verfahrens** (auch Einzelschrittverfahren genannt) lautet

$$x^{(i+1)} = (L + D)^{-1}(b - Rx^{(i)}),$$

d.h. die Iterationsmatrix ist $C_{GS} = -(L + D)^{-1}R$.

Satz 4.3.1 (Konvergenzsatz) *Es sei $A \in \mathbb{R}^{n \times n}$ eine reguläre Matrix, die das starke Zeilensummenkriterium erfüllt. Dann ist das Gauß-Seidel-Verfahren zur Lösung des linearen Gleichungssystems $Ax = b$ konvergent. Genauer gilt in diesem Fall die Abschätzung*

$$\|C_{GS}\|_\infty \leq \|C_J\|_\infty < 1 .$$

Beweis. Sei das starke Zeilensummenkriterium erfüllt. Die Iterationsmatrizen des Gauß-Seidel-Verfahrens bzw. des Jacobi-Verfahrens werden mit $C_{GS} := -(L+D)^{-1}R$ bzw. $C_J := -D^{-1}(L+R)$ bezeichnet. Wenn das starke Zeilensummenkriterium erfüllt ist, gilt die Abschätzung

$$\|C_J\|_\infty = \max_{1 \leq i \leq n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} < 1 .$$

Es sei jetzt $y \in \mathbb{R}^n$ beliebig und $z = C_{GS}\, y$. Durch vollständige Induktion beweisen wir, dass alle Komponenten z_i des Vektors z der Abschätzung

$$|z_i| \le \sum_{\substack{j=1\\ j\neq i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} \|y\|_\infty$$

genügen. Dazu schreiben wir die Gleichung $z = C_{GS}\, y$ um in $-(L+D)z = Ry$ und schätzen ab:

$$|z_1| \le \sum_{j=2}^{n} \frac{|a_{1j}|}{|a_{11}|} |y_j| \le \sum_{j=2}^{n} \frac{|a_{1j}|}{|a_{11}|} \|y\|_\infty .$$

Schreiben wir das Gauß-Seidel-Verfahren mit $B = L + D$ in der Form

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

so folgt daraus dann mit der Induktionsvoraussetzung

$$|z_i| \le \frac{1}{|a_{ii}|} \left(\sum_{j=1}^{i-1} |a_{ij}|\, |z_j| + \sum_{j=i+1}^{n} |a_{ij}|\, |y_j| \right)$$
$$\le \frac{1}{|a_{ii}|} \left(\sum_{j=1}^{i-1} |a_{ij}|\, \|C_J\|_\infty + \sum_{j=i+1}^{n} |a_{ij}| \right) \|y\|_\infty \le \sum_{\substack{j=1\\ j\neq i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} \|y\|_\infty .$$

Hiermit erhält man die Abschätzung

$$\|C_{GS}\, y\|_\infty = \|z\|_\infty \le \|C_J\|_\infty \|y\|_\infty \qquad \forall\, y \in \mathbb{R}^n$$

und somit

$$\|C_{GS}\|_\infty \le \|C_J\|_\infty < 1 \,. \tag{4.4}$$

Da $\varrho(C_{GS}) \le \|C_{GS}\|_\infty$, folgt daraus die Konvergenz des Verfahrens. □

Bemerkung 4.3.2 *Häufig verleitet* (4.4) *zu der falschen Schlußfolgerung, dass das Gauß-Seidel-Verfahren schneller als das Jacobi-Verfahren konvergiert , wenn die Matrix diagonaldominant ist.*

Beispiel 4.3.3 Dass bei strikter Diagonaldominanz einer Matrix $A \in \mathbb{R}^{n\times n}$ aus $\|C_{GS}\|_\infty \le \|C_J\|_\infty < 1$ nicht $\varrho(C_{GS}) \le \varrho(C_J)$ folgen muss, sieht man, wenn man die Matrix A folgendermaßen wählt:

$$A = \begin{pmatrix} 50 & -10 & -20 \\ -20 & 49 & -20 \\ 20 & -10 & 49 \end{pmatrix} .$$

Dann besitzen die zugehörigen Iterationsmatrizen

$$C_{GS} = -(L+D)^{-1}R = \frac{1}{12005} \begin{pmatrix} 0 & 2401 & 4802 \\ 0 & 980 & 6860 \\ 0 & -780 & -560 \end{pmatrix} \qquad C_J = \frac{1}{245} \begin{pmatrix} 0 & 49 & 98 \\ 100 & 0 & 100 \\ -100 & 50 & 0 \end{pmatrix}$$

nämlich die Spektralradien $\varrho(C_{GS}) = (4\sqrt{5})/49 \approx 0.1825$ und $\varrho(C_J) = 2/49 \approx 0.04082$ sowie die Maximumnormen $\|C_{GS}\|_\infty = 32/49 \approx 0.6531$ und $\|C_J\|_\infty = 40/49 \approx 0.8163$.

Bemerkung 4.3.4 *Mit Bezug auf das letzte Beispiel halten wir fest: Ist eine Matrix A strikt diagonaldominant, gilt für die Iterationsmatrizen $\|C_{GS}\|_\infty \leq \|C_J\|_\infty < 1$, es folgt im Allgemeinen aber nicht $\varrho(C_{GS}) \leq \varrho(C_J)$.*

Satz 4.3.5 (Schwaches Zeilensummenkriterium) *Ist $A \in \mathbb{R}^{n\times n}$ irreduzibel und erfüllt das schwache Zeilensummenkriterium, so ist das Gauß-Seidel-Verfahren konvergent.*

Beweis. Die Wohldefiniertheit von $C_{GS} = -(L+D)^{-1}R$ ist wieder klar. Wir betrachten $M := C_{GS} - \lambda I$ für $\lambda \in \mathbb{C}$ mit $|\lambda| \geq 1$. Durch Multiplikation mit $-(L+D)$ sieht man, dass M genau dann regulär ist, wenn $\widetilde{M} := R + \lambda L + \lambda D$ regulär ist. Offensichtlich erbt $\widetilde{M}$ die Irreduzibilität von $A = D + L + R$. Ferner erfüllt $\widetilde{M}$ das schwache Zeilensummenkriterium, denn es gilt

$$\sum_{\substack{k=1\\k\neq j}}^{n} |\widetilde{m}_{jk}| = |\lambda| \sum_{k=1}^{j-1} |a_{jk}| + \sum_{k=j+1}^{n} |a_{jk}| \leq |\lambda| \sum_{\substack{k=1\\k\neq j}}^{n} |a_{jk}| \leq |\lambda|\, |a_{jj}| = |\widetilde{m}_{jj}| \quad \text{für } j = 1, \ldots, n$$

mit strikter Ungleichung für mindestens einen Index j. Nach Lemma 4.2.8 ist $\widetilde{M}$ regulär. Insgesamt erhalten wir wie zuvor $\varrho(C_{GS)}) < 1$. □

Beispiel 4.3.6 Es sei $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ mit $a, b, c, d \in \mathbb{C}$. Die zugehörige Iterationsmatrix zum Jacobi-Verfahren lautet somit

$$C_J := -D^{-1}(L+R) = -\begin{pmatrix} a^{-1} & 0 \\ 0 & d^{-1} \end{pmatrix} \begin{pmatrix} 0 & b \\ c & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{b}{a} \\ -\frac{c}{d} & 0 \end{pmatrix}.$$

Das charakteristische Polynom hierzu lautet $p(\lambda) = \lambda^2 - \frac{bc}{ad}$ und hat Nullstellen $\lambda_{1,2} = \pm\sqrt{\frac{bc}{ad}}$. Entsprechend erhält man für das Gauß-Seidel-Verfahren

$$C_{GS} := -(L+D)^{-1}R = -\frac{1}{ad}\begin{pmatrix} d & 0 \\ -c & a \end{pmatrix} \begin{pmatrix} 0 & b \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{bd}{ad} \\ 0 & \frac{bc}{ad} \end{pmatrix}.$$

Hier lautet das charakteristiscshe Polynom $p(\lambda) = \lambda(\lambda - bc/ad)$ und hat Nullstellen

$$\lambda_1 = 0, \quad \lambda_2 = \frac{bc}{ad}.$$

womit

$$\varrho\,(C_J) = \sqrt{\frac{|bc|}{|ad|}} \quad \text{und} \quad \varrho\,(C_{GS}) = \frac{|bc|}{|ad|}$$

gilt. Man beachte

$$\|C_J\|_1 = \|C_J\|_\infty = \max\{|b/a|, |c/d|\}, \quad \|C_{GS}\|_1 = \frac{|b|(|c|+|d|)}{|ad|}, \quad \|C_{GS}\|_\infty = \frac{|b|\max\{|c|,|d|\}}{|ad|}.$$

Wir können somit für A aus $\mathbb{R}^{2\times 2}$ festhalten: Konvergiert das Jacobi- oder Gauß-Seidel-Verfahren, so konvergiert auch das jeweilige andere. Und im Falle der Konvergenz, konvergiert das Gauß-Seidel doppelt so schnell wie das Jacobi-Verfahren. Gilt dies immer, bzw. kann dies ggf. einfach charakterisiert werden?

Aufgabe 4.3.7 Man nutze das letzte Beispiel, um zu zeigen, dass aus $\|C_J\|_1 < 1$ im Allgemeinen

$$\|C_{GS}\|_1 \leq \|C_J\|_1 < 1$$

nicht folgt. (Z.B. $a = d = 1,\ b = c = 2/3$ in der Matrix von Beispiel 4.3.6 liefert ein Gegenbeispiel).

Definition 4.3.8 *Eine Matrix $A \in \mathbb{R}^{m\times n}$ heißt nichtnegativ, wenn alle Koeffizienten a_{ij} von A nichtnegativ sind.*

Satz 4.3.9 (von Stein und Rosenberg [Hämmerlin/Hoffmann]) *Die Iterationsmatrix $C_J \in \mathbb{R}^{n\times n}$ des Jacobi-Verfahrens sei nichtnegativ. Dann gilt genau eine der folgenden Aussagen:*

i) $\varrho(C_J) = \varrho(C_{GS}) = 0$

ii) $\varrho(C_J) = \varrho(C_{GS}) = 1$

iii) $0 < \varrho(C_{GS}) < \varrho(C_J) < 1$

iv) $1 < \varrho(C_J) < \varrho(C_{GS})$

Beispiel 4.3.10

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \Rightarrow C_J = -D^{-1}(L+R) = \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

Die Iterationsmatrix ist nichtnegativ und somit folgt, dass das Gauß-Seidel-Verfahren schneller ist als das Jacobi-Verfahren!

Bemerkung 4.3.11 (Jacobi- immer schlechter als Gauß-Seidel-Verfahren?) *Dass das Gauß-Seidel-Verfahren nicht immer besser sein muss als das Jacobi-Verfahren oder aus der Divergenz des Jacobi-Verfahrens nicht auch die Divergenz des Gauß-Seidel-Verfahrens folgen muss, zeigen die folgenden beiden Beispiele.*

i.) Die Iterationsmatrizen C_J bzw. C_{GS} zur Matrix

$$A = \begin{pmatrix} 1 & 2 & -2 & 2 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 2 \\ -1 & -2 & 1 & 1 \end{pmatrix}$$

lauten

$$C_J = \begin{pmatrix} 0 & -2 & 2 & -2 \\ -1 & 0 & -1 & 0 \\ -2 & -2 & 0 & -2 \\ 1 & 2 & -1 & 0 \end{pmatrix} \quad \text{bzw.} \quad C_{GS} = -(L+D)^{-1}R = \begin{pmatrix} 0 & -2 & 2 & -2 \\ 0 & 2 & -3 & 2 \\ 0 & 0 & 2 & -2 \\ 0 & 2 & -6 & 4 \end{pmatrix}$$

mit den Spektralradien $\varrho(C_J) = 0$ und $\varrho(C_{GS}) \approx 7.3850$.

ii.) Die Iterationsmatrizen C_J bzw. C_{GS} zur Matrix

$$A = \frac{1}{3}\begin{pmatrix} 3 & -2 & -1 & 1 \\ 1 & 2 & -2 & 1 \\ 1 & -2 & 2 & 2 \\ 1 & -2 & 1 & 1 \end{pmatrix}$$

lauten

$$C_J = \frac{1}{6}\begin{pmatrix} 0 & 4 & 2 & -2 \\ -3 & 0 & 6 & -3 \\ -3 & 6 & 0 & -6 \\ -6 & 12 & -6 & 0 \end{pmatrix} \quad \text{bzw.} \quad C_{GS} = -(L+D)^{-1}R = \frac{1}{6}\begin{pmatrix} 0 & 4 & 2 & -2 \\ 0 & -2 & 5 & -2 \\ 0 & -4 & 4 & -7 \\ 0 & -4 & 4 & 5 \end{pmatrix}$$

mit den Spektralradien $\varrho(C_J) \approx 1.4527$ und $\varrho(C_{GS}) \approx 0.9287 < 1$.

Bemerkungen 4.3.12 *i) Die obigen Iterationsverfahren ließen sich in der Form* $x^{(k+1)} = B^{-1}(B-A)x^{(k)} + B^{-1}b = Cx^{(k)} + d$ *schreiben. Da die Iterationsmatrix* C *für alle* k *konstant ist, spricht man von stationären Iterationsverfahren.*

ii) Das quantitative Verhalten solch stationärer Verfahren lässt sich durch die Einführung eines (Relaxations-) Parameters ω *verbessern:*

$$x^{(k+1)} = \omega(Cx^{(k)} + d) + (1-\omega)x^{(k)}\,.$$

Für $0 < \omega < 1$ *spricht man von einem* **Unterrelaxationsverfahren** *und für* $\omega > 1$ *von einem* **Überrelaxationsverfahren**. *Man kann zeigen, dass der optimale Parameter für eine positiv definite Matrix* A *beim gedämpften Jacobiverfahren*

$$\omega_{opt} = \frac{2}{\lambda_{\min}(D^{-1}A) + \lambda_{\max}(D^{-1}A)}$$

lautet und für das überrelaxierte Gauß-Seidel-Verfahren (SOR = **s**uccessive **o**ver**r**elaxation *method) angewandt auf eine positiv definite Matrix* $A = L+D+L^T$ *ergibt sich der optimale Parameter zu*

$$\omega_{opt} = \frac{2}{1+\sqrt{\lambda_{\min}(D^{-1}A) + \lambda_{\max}((D+2L)D^{-1}(D+2L^T)A^{-1})}}\,.$$

Ergebnisse für allgemeinere Fälle findet man z.B. bei [Niethammer].

iii) Die Bedeutung der oben genannten Iterationsverfahren liegt heute weniger in deren unmittelbarem Einsatz zur Lösung von $Ax = b$*, sondern auf Grund deren „***Glättungseigenschaften***“ als Beschleuniger anderer moderner Verfahren (vorkonditioniertes konjugiertes Gradienten–Verfahren, Mehrgitter).*

4.4 ABBRUCH-KRITERIEN

Da ein Iterationsverfahren aufeinanderfolgende Näherungen der Lösung liefert, ist ein praktischer Test notwendig, um das Verfahren zu stoppen, wenn die gewonnene Approximation genau genug ist. Da es nicht möglich ist, den Fehler $x - x^{(k)}$ zu bestimmen, d.h. den Abstand zur eigentlichen (gesuchten) Lösung zu bestimmen, müssen andere Quantitäten gewonnen werden, die meist auf dem Residuum $r = b - Ax$ basieren.
Die hier vorgestellten Verfahren liefern eine Folge $(x^{(i)})$ von Vektoren die gegen den Vektor x streben, welcher Lösung des linearen Gleichungssystems $Ax = b$ ist. Um effizient zu sein, muss die Methode wissen, wann sie abbrechen soll. Eine gute Methode sollte

i) feststellen, ob der Fehler $e^{(i)} := x - x^{(i)}$ klein genug ist,

ii) abbrechen, falls der Fehler nicht weiter kleiner wird oder nur noch sehr langsam, und

iii) den maximalen Aufwand, der zur Iteration verwendet wird, beschränken.

Das folgende Abbruchkriterium ist eine einfache aber häufig genügende Variante. Man muss hierzu die Quantitäten *maxit*, $\|b\|$, *tol* und wenn möglich auch $\|A\|$ (und $\|A^{-1}\|$) zur Verfügung stellen. Dabei ist

- die natürliche Zahl *maxit* die maximale Anzahl an möglichen Iterationen des Verfahrens,
- die reelle Zahl $\|A\|$ eine Norm von A, (Jede einigermaßen vernünftige Approximation des betragsgrößten Eintrags in A genügt schon.)

- die reelle Zahl $\|b\|$ eine Norm von b, (Auch hier genügt eine einigermaßen vernünftige Approximation des betragsgrößten Eintrags in b.)
- die reelle Zahl *tol* eine Schranke für die Größe des Residuums bzw. des Fehlers.

MATLAB-Beispiel: Beispiel eines vernünftigen Abbruchkriteriums

```
k = 0;
while 1
  k = k + 1;
  % Berechne die Approximation x^(k)
  % Berechne das Residuum r^(k) = b - A x^(k)
  % Berechne norm_ak = || A * x^(k) ||, norm_rk = || r^(k) ||
  % und norm_b = || b ||
  if (k >= maxit) | ( norm_rk <= tol * max( norm_ak, norm_b ) )
    break
  end
end
```

Da sich nach einigen Iterationen der Term $\|Ax^{(k)}\|$ nicht mehr groß ändert, braucht man diesen nicht immer neu zu bestimmen. Zu bestimmen ist eigentlich $\|e^{(k)}\| = \|A^{-1}r^{(k)}\| \leq \|A^{-1}\|\,\|r^{(k)}\|$. Man beachte, dass man $\|A^{-1}B\|$ bei den bisherigen Verfahren mit Lemma 3.2.6 zur Neumannschen-Reihe abschätzen kann. Es gilt $x^{(k+1)} = B^{-1}(B-A)x^{(k)} + B^{-1}b = Cx^{(k)} + d$ und

$$B^{-1}A = I - B^{-1}(B-A) = I - C \quad \text{sowie} \quad \|A^{-1}B\| = \|(I-C)^{-1}\| \leq \frac{1}{1-\|C\|}\,.$$

4.5 GRADIENTEN–VERFAHREN

Im Folgenden nehmen wir stets an, dass

$$A \in \mathbb{R}^{n\times n} \quad \text{positiv definit (s.p.d.) ist.} \tag{4.5}$$

Wir ordnen nun dem Gleichungssystem $Ax = b$ die Funktion $f(x) := \frac{1}{2}x^T Ax - b^T x,\ f : \mathbb{R}^n \to \mathbb{R}$ zu. Der Gradient von f ist $f'(x) = \frac{1}{2}(A + A^T)x - b$. Da $A = A^T$ nach Voraussetzung (4.5), lautet die Ableitung

$$f'(x) = \nabla f(x) = Ax - b\,.$$

Notwendig für ein Minimum von f ist das Verschwinden des Gradienten, d.h. $Ax = b$. Da die Hesse-Matrix $f''(x) = A$ positiv definit ist, liegt für die Lösung von $Ax = b$ tatsächlich ein Minimum vor.
Mit $\arg\min\limits_{y\in\mathbb{R}^n} f(y)$ bezeichnen wir denjenigen Wert aus $\mathbb{R}^n$, der den Term f minimiert, d.h. $f(x) = \min\limits_{y\in\mathbb{R}^n} f(y)$, wenn $x = \arg\min\limits_{y\in\mathbb{R}^n} f(y)$ ist.
Idee:

$$Ax = b \iff x = \arg\min_{y\in\mathbb{R}^n} f(y)$$

mit $f(y) := \frac{1}{2}y^T Ay - b^T y,\ f : \mathbb{R}^n \to \mathbb{R}$.
Bewiesen haben wir soeben das folgende Lemma.

Lemma 4.5.1 *Es sei $A \in \mathbb{R}^{n\times n}$ positiv definit, dann gilt*

$$Ax = b \iff x = \arg\min_{y\in\mathbb{R}^n} f(y)\,.$$

Alternativer Beweis zu Lemma 4.5.1 . Ein zweiter Beweis folgt aus der Darstellung

$$f(x) = f(x^*) + \frac{1}{2}(x - x^*)^T A(x - x^*) \quad \text{mit } x^* := A^{-1}b. \tag{4.6}$$

Hieraus folgt $f(x) > f(x^*)$ für $x \neq x^*$, d.h. $x^* := A^{-1}b$ ist das eindeutige Minimum von f. Man beachte dabei, dass (4.6) ein Sonderfall der folgenden Entwicklung von f um einen beliebigen Wert $\tilde{x} \in \mathbb{R}^n$ ist, welche sich durch ausmultiplizieren zeigen lässt:

$$f(x) = f(\tilde{x}) + \langle A\tilde{x} - b, x - \tilde{x}\rangle + \frac{1}{2}\langle A(x - \tilde{x}), x - \tilde{x}\rangle$$

□

Folgerung: Man kann also Verfahren zur numerischen Optimierung/Minimierung verwenden.
Der **Gradient** ist die Richtung des **steilsten Anstiegs**, also kann man $-\nabla f$ als Abstiegsrichtung wählen und entlang dieser Geraden ein Minimum suchen.

Gradientenverfahren (allgemein):
Es sei $\Omega \subseteq \mathbb{R}^n$, $f : \Omega \to \mathbb{R}$, $x_0 \in \Omega$ Startwert, für $k = 1, 2, 3, \ldots$

1) *Bestimmung der Abstiegsrichtung* $d_k = -\nabla f(x_k)$

2) *Liniensuche:* suche auf der Geraden $\{x_k + td_k : t \geq 0\} \cap \Omega$ ein Minimum, d.h. bestimme $\alpha_k \geq 0 : f(x_k + \alpha_k d_k) \leq f(x_k)$

$$x_{k+1} = x_k + \alpha_k d_k\,.$$

Daraus folgt $f(x_0) \geq f(x_1) \geq f(x_2) \geq \ldots$

Für die quadratische Funktionen $f(x) = \frac{1}{2}x^T Ax - b^T x$ und $\Omega \equiv \mathbb{R}^n$ kann man 1) und 2) leicht berechnen: Da $\nabla f(x) = Ax - b$, ergibt sich $d_k = -\nabla f(x_k) = b - Ax_k$. Sei $p \in \mathbb{R}^n \setminus \{0\}$ und $F(\lambda) := f(x + \lambda p)$, dann gilt für die Liniensuche

$$\begin{aligned} F(\lambda) &= f(x+\lambda p) \\ &= \frac{1}{2}\langle x+\lambda p, A(x+\lambda p)\rangle - \langle b, x+\lambda p\rangle \\ &= \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle + \lambda\langle p, Ax-b\rangle + \frac{1}{2}\lambda^2\langle p, Ap\rangle \qquad (4.7) \\ &= f(x) + \lambda\langle p, Ax-b\rangle + \frac{1}{2}\lambda^2\langle p, Ap\rangle \end{aligned}$$

Da $p \neq 0$ nach Voraussetzung, folgt $\langle p, Ap\rangle > 0$. F ist also eine quadratische Funktion mit positivem führenden Koeffizienten. Somit folgt aus

$$0 \stackrel{!}{=} F'(\lambda) = \langle p, Ax-b\rangle + \lambda\langle p, Ap\rangle\,, \qquad (4.8)$$

dass der Parameter

$$\lambda_{opt}(x,p) = \frac{\langle p, b-Ax\rangle}{\langle p, Ap\rangle}\,.$$

zu gegebenem Vektor $p \in \mathbb{R}^n \setminus \{0\}$ das Funktional $F(\lambda) := f(x+\lambda p)$ minimiert.

Für allgemeine Funktionen wird die Liniensuche angenähert, z.B. mit der **Schrittweitenregel von Armijo**

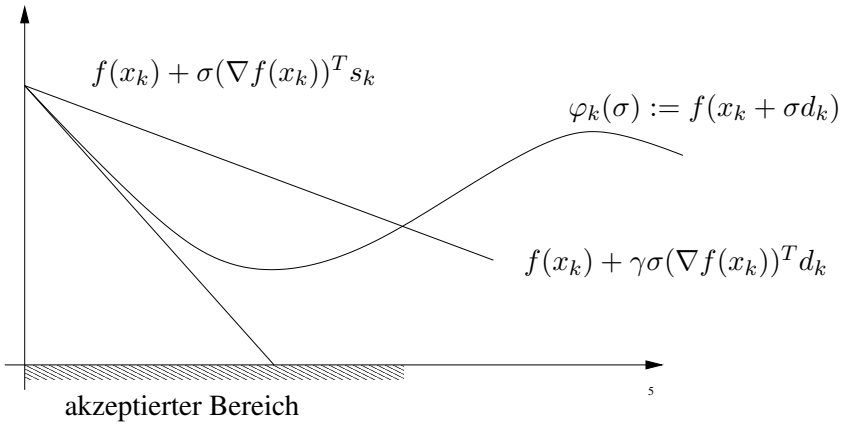


Abb. 4.3: *Optimale Wahl des Dämpfungsparameters* .

Schrittweitenregel von Armijo:
Wähle $\beta \in (0,1)$ (z.B. $\beta = \frac{1}{2}$) und $\gamma \in (0,1)$ (z.B. $\gamma \in [10^{-3}, 10^{-2}]$)
Bestimme die größte Schrittweite $\sigma_k \in \{1, \beta, \beta^2, \beta^3, \ldots\}$ mit

$$f(x_k) - f(x_k - \sigma_k d_k) \geq -\gamma\sigma_k \nabla f(x_k)^T d_k\,,$$

d.h.

$$f(x_k - \sigma_k d_k) \leq f(x_k) + \gamma\sigma_k \nabla f(x_k)^T d_k$$

.

Wir untersuchen nun die Konvergenz des Verfahrens für quadratische Funktionen. Hierzu bietet sich die sogenannte *Energienorm* an

$$\|x\| := \sqrt{x^T Ax}\,,\ (A \in \mathbb{R}^{n\times n})$$

(beachte: alle Normen auf dem $\mathbb{R}^{n\times n}$ sind äquivalent).

Algorithmus 4.5.1: Gradientenverfahren: $Ax = b$

Input: `Initial guess` x^0

$r^0 := b - Ax^0$

Iteration: $k = 0, 1, \ldots$

$a^k := Ar^k$

$\lambda_{opt} := \langle r^k, r^k\rangle \,/\, \langle r^k, a^k\rangle$

$x^{k+1} := x^k + \lambda_{opt}\, r^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

`Man beachte` $r^{k+1} = b - Ax^{k+1} = b - A(x^k + \lambda_{opt}\, r^k) = r^k - \lambda_{opt}\, Ar^k$

Lemma 4.5.2 *Es sei $A \in \mathbb{R}^{n\times n}$ positiv definit und $x^* \in \mathbb{R}^n$ erfülle $Ax^* = b$. Dann gilt*

$$\|x_{k+1} - x^*\|_A^2 \le \|x_k - x^*\|_A^2 \left(1 - \frac{\langle r_k, r_k\rangle^2}{\langle r_k, Ar_k\rangle \langle r_k, A^{-1}r_k\rangle}\right) .$$

Beweis. Es gilt

$$\begin{aligned} f(x_{k+1}) &= f(x_k + \lambda_{opt}\, r_k) = \frac{1}{2}\lambda_{opt}^2\langle r_k, Ar_k\rangle - \lambda_{opt}\langle r_k, r_k\rangle + f(x_k) \\ &= f(x_k) + \frac{1}{2}\frac{\langle r_k, r_k\rangle^2}{\langle r_k, Ar_k\rangle} - \frac{\langle r_k, r_k\rangle^2}{\langle r_k, Ar_k\rangle} = f(x_k) - \frac{1}{2}\frac{\langle r_k, r_k\rangle^2}{\langle r_k, Ar_k\rangle} \end{aligned} \tag{4.9}$$

Für die exakte Lösung x^* von $Ax = b$ gilt für $x \in \mathbb{R}^n$

$$\begin{aligned} f(x^*) &+ \frac{1}{2}\|x - x^*\|_A^2 = \frac{1}{2}\langle x^*, Ax^*\rangle - \langle b, x^*\rangle + \frac{1}{2}\langle (x - x^*), A(x - x^*)\rangle \\ &= \frac{1}{2}\langle x^*, Ax^*\rangle - \langle x^*, b\rangle + \frac{1}{2}\langle x, Ax\rangle - \langle x, Ax^*\rangle + \frac{1}{2}\langle x^*, Ax^*\rangle \\ &= \langle x^*, Ax^* - b\rangle + \frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle \\ &= f(x) , \qquad (\text{d.h. } \|x - x^*\|_A^2 = 2(f(x) - f(x^*)) \end{aligned}$$

also mit (4.9)

$$\begin{aligned} \|x_{k+1} - x^*\|_A^2 &= 2f(x_{k+1}) - 2f(x^*) = 2f(x_{k+1}) - 2f(x_k) + \|x_k - x^*\|_A^2 \\ &= \|x_k - x^*\|_A^2 - \frac{\langle r_k, r_k\rangle^2}{\langle r_k, Ar_k\rangle} \end{aligned}$$

Mit $r_k = b - Ax_k = A(x^* - x_k)$ folgt wegen

$$\begin{aligned} \|x_k - x^*\|_A^2 &= \|x^* - x_k\|_A^2 = \langle (x^* - x_k), A(x^* - x_k)\rangle \\ &= \langle A^{-1}A(x^* - x_k), r_k\rangle = \langle r_k, A^{-1}r_k\rangle \end{aligned}$$

die Behauptung. □

Frage: Was sagt Lemma 4.5.2 bzgl. der Konvergenz und Kondition aus?

Lemma 4.5.3 ***(Kantorowitsch–Ungleichung)***
Es sei $A \in \mathbb{R}^{n\times n}$ positiv definit und $\kappa := \kappa_2(A) := \|A\|_2 \, \|A^{-1}\|_2$. Dann gilt für alle $x \in \mathbb{R} \setminus \{0\}$

$$\frac{\langle x, Ax\rangle \, \langle x, A^{-1}x\rangle}{\langle x, x\rangle^2} \leq \frac{1}{4}\left(\sqrt{\kappa} + \sqrt{\kappa^{-1}}\right)^2$$

Beweis:

Beweis. Die Eigenwerte von A seien geordnet gemäß

$$0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \, , \; \kappa = \frac{\lambda_n}{\lambda_1}$$

Da A symmetrisch ist, existiert eine orthonormale Matrix Q mit $Q^T A Q = \Lambda = \text{ diag } (\lambda_i)$. Für $y = Q^T x$ gilt dann

$$x^T A x = x^T Q \Lambda Q^T x = y^T \Lambda y = \sum_{i=1}^n \lambda_i y_i^2, \quad x^T A^{-1} x = x^T Q \Lambda^{-1} Q^T x = \sum_{i=1}^n \lambda_i^{-1} y_i^2$$

sowie $x^T x = x^T Q Q^T x = y^T y$, also

$$\frac{\langle x, Ax\rangle \, \langle x, A^{-1}x\rangle}{\langle x, x\rangle^2} \;=\; \frac{\left(\sum\limits_i \lambda_i y_i^2\right)\left(\sum\limits_i \lambda_i^{-1} y_i^2\right)}{\|y\|_2^4} = \left(\sum_{i=1}^n \lambda_i z_i\right)\left(\sum_{i=1}^n \lambda_i^{-1} z_i\right) \quad (4.10)$$

mit $z_i := \frac{y_i^2}{\|y\|_2^2}$. Man beachte $\sum\limits_{i=1}^n z_i = 1$. Für $\lambda_1 \leq x \leq \lambda_n$ folgt

$$0 \geq (x - \lambda_1)(x - \lambda_n) = x^2 - x(\lambda_1 + \lambda_n) + \lambda_1 \lambda_n$$

und somit

$$\lambda_1 \lambda_n + \lambda_k^2 \leq \lambda_k(\lambda_1 + \lambda_n) \quad \Rightarrow \quad \frac{\lambda_1 \lambda_n}{\lambda_k} + \lambda_k \leq \lambda_1 + \lambda_n \quad (k = 1, \ldots, n) \, .$$

Es gilt nun

$$\lambda_1\lambda_n \sum_{i=1}^n \lambda_i^{-1} z_i + \sum_{i=1}^n \lambda_i z_i = \left(\frac{\lambda_1\lambda_n}{\lambda_1} + \lambda_1\right) z_1 + \left(\frac{\lambda_1\lambda_n}{\lambda_2} + \lambda_2\right) z_2 + \ldots + \left(\frac{\lambda_1\lambda_n}{\lambda_n} + \lambda_n\right) z_n \leq \lambda_1 + \lambda_n,$$

d.h.

$$\sum_{i=1}^n \lambda_i^{-1} z_i \leq \frac{\lambda_1 + \lambda_n - \lambda}{\lambda_1 \lambda_n}$$

mit $\lambda := \sum\limits_{i=1}^n \lambda_i z_i$. Somit lässt sich (4.10) abschätzen durch

$$\frac{\langle x, Ax\rangle \, \langle x, A^{-1}x\rangle}{\langle x, x\rangle^2} \leq \underbrace{\lambda \cdot \frac{\lambda_1 + \lambda_n - \lambda}{\lambda_1 \lambda_n}}_{=:h(\lambda)}$$

Für welches λ wird nun das Polynom h maximal?

$$\begin{aligned} h'(\lambda) &= \frac{\lambda_1 + \lambda_n - \lambda}{\lambda_1\lambda_n} - \frac{\lambda}{\lambda_1\lambda_n} = \frac{\lambda_1 + \lambda_n}{\lambda_1\lambda_n} - \lambda\frac{2}{\lambda_1\lambda_n} \stackrel{!}{=} 0 \quad \Rightarrow \lambda^* = \frac{\lambda_1 + \lambda_n}{2} \\ h''(\lambda) &= -\frac{2}{\lambda_1\lambda_n} < 0 \quad \Rightarrow \lambda^* \text{ maximiert h} \end{aligned}$$

d.h.

$$\max_{\lambda\in[\lambda_1,\lambda_n]} h(\lambda) = h(\lambda^*) = \frac{(\lambda_1+\lambda_n)^2}{4\,\lambda_1\lambda_n} = \frac{1}{4}\left(\sqrt{\frac{\lambda_1}{\lambda_n}} + \sqrt{\frac{\lambda_n}{\lambda_1}}\right)^2 = \frac{1}{4}\left(\sqrt{\kappa} + \sqrt{\kappa^{-1}}\right)^2 .$$

□

Satz 4.5.4 *Es sei $A \in \mathbb{R}^{n\times n}$ positiv definit. Dann gilt für das Gradientenverfahren*

$$\|x_k - x^*\|_A \le \left(\frac{\kappa-1}{\kappa+1}\right)^k \|x_0 - x^*\|_A$$

Beweis. Lemma 4.5.2 liefert die Abschätzung

$$\|x_{k+1} - x^*\|_A^2 \le \|x_k - x^*\|_A^2 \left(1 - \frac{\langle r_k, r_k\rangle^2}{\langle r_k, Ar_k\rangle \langle r_k, A^{-1}r_k\rangle}\right) .$$

und mit Lemma 4.5.3 ergibt sich

$$1 - \frac{\langle r_k, r_k\rangle^2}{\langle r_k, Ar_k\rangle \langle r_k, A^{-1}r_k\rangle} \le 1 - 4(\sqrt{\kappa} + \sqrt{\kappa^{-1}})^{-2} = \frac{(\sqrt{\kappa} + \sqrt{\kappa^{-1}})^2 - 4}{(\sqrt{\kappa} + \sqrt{\kappa^{-1}})^2}$$
$$= \frac{\kappa + 2 + \kappa^{-1} - 4}{\kappa + 2 + \kappa^{-1}} = \frac{\kappa^2 - 2\kappa + 1}{\kappa^2 + 2\kappa + 1} = \left(\frac{\kappa-1}{\kappa+1}\right)^2 .$$

□

Bemerkungen:

(a) Für große κ gilt

$$\frac{\kappa-1}{\kappa+1} = \underbrace{\frac{\kappa}{\kappa+1}}_{\approx 1} - \frac{1}{\kappa+1} \approx 1 - \frac{1}{\kappa+1} ,$$

also sehr nahe bei 1, d.h. geringe Konvergenzgeschwindigkeit!

(b) Dies tritt auch schon bei einfachen Beispielen auf:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & a \end{pmatrix} ,\ a \gg 1 ,\ b = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \text{ und } x_0 = \begin{pmatrix} a \\ 1 \end{pmatrix}$$

daraus folgt (*Übung*).

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \rho \begin{pmatrix} x_k \\ -y_k \end{pmatrix} \text{ mit } \rho = \frac{a-1}{a+1}$$

wegen $a = \kappa_2(A)$ ist das genau die Rate aus Satz 4.5.4!

(c) Anschaulich sieht man ein „Zick–Zack–Verhalten“ der Iteration.

4.6 Verfahren der konjugierten Gradienten

- entstanden 1952 von Hestenes und Stiefel
- enormer Gewinn an Bedeutung 1971 durch Vorkonditionierung
- gehören zu den schnellsten Verfahren, werden heute sehr oft verwandt

- für die 2D–Standard–Matrix ab Systemgröße 2000–4000 besser als Gauß bei zusätzlich erheblich geringerem Speicherbedarf.

Idee: Vermeide Zick–Zack–Verhalten durch Verwendung von Orthogonalität bzgl.

$$(x, y)_A = x^T A y \ ,$$

daraus folgt „konjugierte Gradienten“: *cg-Verfahren* (conjugate gradient)

Bemerkungen 4.6.1

(a) Zwei Vektoren $x, y \in \mathbb{R}^n$ heißen **konjugiert** *oder A-***orthogonal***, falls $(x, y)_A = 0$.*

(b) $\{x_1, \ldots, x_k\}$ paarweise konjugiert, d.h. $(x_i, x_j)_A = \delta_{ij}\|x_i\|_A^2$, $x_i \neq 0$ $(i, j \in \{1, \ldots, k\})$ daraus folgt $\{x_1, \ldots, x_k\}$ linear unabhängig.

(c) Jeder Vektor $x \in \mathbb{R}^n$ besitzt eine eindeutige Entwicklung

$$x = \sum_{k=1}^{n} \alpha_k d_k \tag{4.11}$$

bezüglich konjugierter Richtungen $\{d_1, \ldots, d_n\}$. Aus (4.11) folgt

$$(x, d_i)_A = \sum_{k=1}^{n} \alpha_k \underbrace{(d_k, d_i)_A}_{=\delta_{ik}\|d_i\|_A^2} \quad = \quad \alpha_i \|d_i\|_A^2 \ ,$$

also

$$\alpha_k = \frac{d_k^T A x}{d_k^T A d_k} \quad (k = 1, \ldots, n) \, . \tag{4.12}$$

(d) Für die Lösung x^* von $Ax = b$ gilt offenbar

$$\alpha_i = \frac{d_i^T b}{d_i^T A d_i}$$

Lemma 4.6.2 *Seien $\{d_1, \ldots, d_n\}$ konjugierte Richtungen. Für jedes $x_0 \in \mathbb{R}^n$ und*

$$x_k = x_{k-1} + \alpha_k d_k \ , \ \alpha_k = \frac{r_{k-1}^T d_k}{d_k^T A d_k} \ , \ r_k := b - A x_k \quad (k \geq 1) \tag{4.13}$$

gilt nach (höchstens) n Schritten $x_n = A^{-1}b$.

Beweis. Aus (4.12), (4.13) folgt für $x^* = A^{-1}b$,

$$x^* - x_0 = \sum_{k=1}^{n} \tilde{\alpha}_k d_k \ \text{ mit } \ \tilde{\alpha}_k = \frac{d_k^T A(x^* - x_0)}{d_k^T A d_k} = \frac{d_k^T (b - A x_0)}{d_k^T A d_k} \, .$$

Da d_k zu d_i, $i \neq k$ konjugiert ist, gilt

$$d_k^T A(x_{k-1} - x_0) = d_k^T A \left(\sum_{i=1}^{k-1} \hat{\alpha}_i d_i \right) = \sum_{i=1}^{k-1} \hat{\alpha}_i \underbrace{d_k^T A d_i}_{=0, \text{ da } i \neq k} = 0 \, ,$$

also

$$\begin{aligned} d_k^T A(x^* - x_0) &= d_k^T A(x^* - x_{k-1}) + \underbrace{d_k^T A(x_{k-1} - x_0)}_{=0} \\ &= d_k^T (b - A x_{k-1}) = d_k^T r_{k-1} \Rightarrow \tilde{\alpha}_k = \alpha_k \, . \end{aligned}$$

□

Bemerkungen 4.6.3

(a) $r_k = b - Ax_k$ *wird als* **Residuum** *von* $Ax = b$ *bzgl* x_k *bezeichnet.*

(b) Lemma 4.6.2 besagt, dass das Verfahren ein direktes Verfahren ist, welches nach n *Iterationen konvergiert.*

$$A \text{ sparse } \Rightarrow O(n^2)$$

(dies ist nicht *optimal).*

(c) Wie in (4.8) gilt

$$\begin{aligned} &\tfrac{d}{d\lambda} f(x_k + \lambda\, d_k) = \lambda \langle d_k, Ad_k\rangle + \langle d_k, (Ax_k - b)\rangle \\ &f(x_k) = f(x_{k-1} + \alpha_k d_k) = \min_{\lambda\in\mathbb{R}} : f(x_{k-1} + \lambda\, d_k)\,, \end{aligned}$$

dann

$$\lambda_{opt} = -\frac{\langle d_k, (Ax_{k-1} - b)\rangle}{\langle d_k, Ad_k\rangle} = \frac{\langle r_{k-1}, d_k\rangle}{\langle d_k, Ad_k\rangle} = \alpha_k$$

Satz 4.6.4 *Seien* $\{d_1, \ldots, d_n\}$ *konjugierte Richtungen und* r_k *(*$k = 0, \ldots, n-1$*), die durch* (4.13) *definierten Residuen. Dann gilt*

$$r_k^T d_j = 0 \text{ bzw. } r_k \perp U_k := \text{span}\{d_1, \ldots, d_k\} \quad (1 \le k \le n,\ 1 \le j \le k)\,. \tag{4.14}$$

Beweis. Nach (4.13) gilt für $k \in \{1, \ldots, n\}$

$$r_k = b - Ax_k = r_{k-1} - \alpha_k Ad_k = r_{k-2} - \alpha_{k-1} Ad_{k-1} - \alpha_k Ad_k = \cdots = r_0 - \sum_{j=1}^{k} \alpha_j Ad_j\,.$$

Daraus folgt nun für $1 \le j \le k$

$$\begin{aligned} r_k^T d_j &= r_0^T d_j - \sum_{\ell=1}^{k} \alpha_\ell\, d_\ell^T A d_j \\ &= r_0^T d_j - \alpha_j\, d_j^T A d_j = r_0^T d_j - \frac{r_{j-1}^T d_j}{d_j^T A d_j}\, d_j^T A d_j \\ &= (r_0^T - r_{j-1}^T) d_j = \sum_{\ell=1}^{j-1} \alpha_\ell\, d_\ell^T A d_j = 0 \end{aligned}$$

womit der Satz bewiesen ist. □

Frage: Wie sind nun die d_k und damit erzeugten Teilräume span$\{d_1, \ldots, d_k\}$ zu wählen?

Falls $r_0 \neq 0$ gilt (sonst ist x_0 schon die gesuchte Lösung) setzt man $d_1 = r_0$.
Für $k = 1, 2, 3, \ldots$ verfahren wir wie folgt. Falls $r_k \neq 0$ (sonst ist x_k schon die gesuchte Lösung), gewinnt man formal d_{k+1} mittels des Gram-Schmidtschen-Orthogonalisierungsverfahren aus r_k und den schon bestimmten konjugierten Richtungen $d_1, \ldots, d_k$, d.h.

$$d_{k+1} = r_k - \sum_{j=1}^{k} \frac{\langle r_k, Ad_j\rangle}{\langle d_j, Ad_j\rangle} d_j\,. \tag{4.15}$$

Aufgabe 4.6.5 Man zeige induktiv, dass die so erzeugten A-orthogonalen Richtungen

$$d_{k+1} \notin \text{span}\{r_0, \ldots, r_{k-1}\} \quad \text{und} \quad d_{k+1} \in \text{span}\{r_0, \ldots, r_k\}$$

erfüllen für $0 \le k \le n-1$ mit $r_k \neq 0$.

Damit das ganze Verfahren effizient wird, benötigen wir noch folgende Eigenschaft

$$Ad_k \in U_{k+1} := \text{span}\{d_1, \ldots, d_{k+1}\} = \text{span}\{r_0, \ldots, r_k\}\,,$$

wenn $r_k \neq 0$ gilt. Denn daraus ergibt sich $\langle r_k, Ad_j\rangle = 0$ für $1 \leq j \leq k-1$ und (4.15) verkürzt sich zu

$$d_{k+1} = r_k - \underbrace{\frac{\langle r_k, Ad_k\rangle}{\langle d_k, Ad_k\rangle}}_{=:\beta_{k+1}} d_k\,. \tag{4.16}$$

(Beweis: Aus der Definition von $r_k = r_{k-1} - \alpha_k Ad_k$ folgt sofort $Ad_k \in \{r_0, \ldots, r_k\}$, da $r_{k-1} \in U_{k-1}$ und $\alpha_k \neq 0$ gilt. Die Gleichheit von U_k und $\{r_0, \ldots, r_k\}$ ergibt sich aus Aufgabe 4.6.5.)
Für den Algorithmus schreiben wir nur noch α_k, β_k um: $\alpha_k = \frac{r_{k-1}^T d_k}{d_k^T Ad_k}$ und

$$\begin{aligned} r_{k-1}^T d_k &= r_{k-1}^T r_{k-1} - \beta_k \underbrace{r_{k-1}^T d_{k-1}}_{=0} \text{ also} \\ \alpha_k &= \frac{r_{k-1}^T r_{k-1}}{d_k^T Ad_k} \end{aligned} \tag{4.17}$$

und wegen $\alpha_k r_k^T Ad_k = (r_{k-1} - r_k)^T r_k = -r_k^T r_k$

$$\beta_{k+1} = \frac{r_k^T Ad_k}{d_k^T Ad_k} = -\frac{r_k^T r_k}{\alpha_k d_k^T Ad_k} = -\frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}\,. \tag{4.18}$$

Bemerkung 4.6.6 *Die Ausdrücke (4.17), (4.18) haben sich als numerisch stabiler und Speicherplatz-optimal herausgestellt.*

Algorithmus 4.6.1: Konjugiertes Gradientenverfahren (cg-Verfahren): $Ax = b$

Input: `Initial guess` x_0

$r_0 := b - Ax_0$

$\rho_0 := \langle r_0, r_0 \rangle$

$d_1 := r_0$

Iteration: $k = 1, 2, \ldots$ `as long as` $k \leq n$ `and` $r_k \neq 0$

$a_k := Ad_k$

$\alpha_k := \rho_{k-1} \,/\, \langle d_k, a_k \rangle$

$x_k := x_{k-1} + \alpha_k \, d_k$

$r_k := r_{k-1} - \alpha_k \, a_k$

$\rho_k := \langle r_k, r_k \rangle$

$d_{k+1} := r_k + \frac{\rho_k}{\rho_{k-1}} \, d_k$

Satz 4.6.7 *Es sei A positiv definit. Für das cg-Verfahren gilt folgende Abschätzung*

$$\|x^* - x_k\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|x^* - x_0\|_A \, .$$

Beweis. Der Beweis gliedert sich in 4 Schritte. Es sei $e_k := x^* - x_k$.

(1) Zuerst zeige man induktiv, dass Polynome $p_k \in \mathbb{P}_k$, $k = 0, \ldots, n-1$, existieren mit

$$e_k = p_k(A) e_0 \, , \quad p_k(0) = 1. \tag{4.19}$$

$k = 0$: klar!
$k - 1 \Rightarrow k$: Aus der Definition (4.13) folgt

$\alpha_k d_k = x_k - x_{k-1} = x^* - x_{k-1} - (x^* - x_k) = e_{k-1} - e_k$, bzw. $e_k = e_{k-1} - \alpha_k d_k$.

Mit (4.16) ergibt sich nun

$$\begin{aligned} d_k &= r_{k-1} - \beta_k d_{k-1} = b - Ax_{k-1} - \beta_k \frac{1}{\alpha_{k-1}} (e_{k-2} - e_{k-1}) \\ &= A(x^* - x_{k-1}) + \frac{\beta_k}{\alpha_{k-1}} (e_{k-1} - e_{k-2}) \\ &= \left(\frac{\beta_k}{\alpha_{k-1}} I + A \right) e_{k-1} - \frac{\beta_k}{\alpha_{k-1}} e_{k-2} \\ &= \underbrace{\left\{ \left(\frac{\beta_k}{\alpha_{k-1}} I + A \right) p_{k-1}(A) - \frac{\beta_k}{\alpha_{k-1}} p_{k-2}(A) \right\}}_{=:\tilde{q}_k(A) \, , \, \tilde{q}_k \in \mathbb{P}_k} e_0 \, , \end{aligned}$$

also

$$e_k = \underbrace{\left[\left(\frac{\alpha_k \beta_k}{\alpha_{k-1}} I + A \right) p_{k-1}(A) - \frac{\alpha_k \beta_k}{\alpha_{k-1}} p_{k-2}(A) + p_{k-1}(A) \right]}_{= (\alpha_k \tilde{q}_k(A) + p_{k-1}(A)) =: p_k(A)} e_0$$

(2) Nun zeige man, dass für alle $q_k \in \mathbb{P}_k$ mit $q_k(0) = 1$ die Ungleichung $\|e_k\|_A \leq \|q_k(A)e_0\|_A$ gilt.

Zuerst halten wir $r_k = b - Ax_k = A(x^* - x_k) = Ae_k$ fest. Des weiteren gilt

$$\begin{aligned}
\|e_k\|_A^2 &:= e_k^T A e_k = r_k^T e_k \\
&= r_k^T \left(e_k - + \sum_{j=0}^{k-1} \sigma_j r_j \right) \quad \text{für beliebige } \sigma_0, \ldots, \sigma_{k-1} \in \mathbb{R} \ (r_j^T r_k = \delta_{jk}) \\
&= r_k^T \left(e_k + \sum_{j=0}^{k-1} \sigma_j A e_j \right) \\
&= (Ae_k)^T \left(p_k(A) + \sum_{j=0}^{k-1} \sigma_j A p_j(A) \right) e_0 \,.
\end{aligned}$$

Sei $q_k \in \mathbb{P}_k$ mit $q_k(0) = 1$ beliebig. Da die $\{p_0, \ldots, p_{k-1}\}$ linear unabhängig sind, folgt aus der letzten Umformung, dass es eindeutig bestimmte $\tilde{\sigma}_0, \ldots, \tilde{\sigma}_{k-1}$ existieren mit

$$q_k(t) = p_k(t) + \sum_{j=0}^{k-1} \tilde{\sigma}_j t p_j(t) \,.$$

Mit $\sigma_i = \tilde{\sigma}_i$ folgt also

$$e_k^T A e_k \;=\; e_k^T A^{1/2} A^{1/2} q_k(A) e_0 \overset{CSU}{=} e_k^T A e_k (q_k(A)e_0)^T A q_k(A) e_0$$

Wir haben somit gezeigt, dass

$$\|e_k\|_A \leq \|q_k(A)e_0\|_A \ \text{ für alle } \ q_k \in \mathbb{P}_k, q_k(0) = 1 \tag{4.20}$$

gilt.

(3) Die Eigenwerte von A seien $0 < \lambda_{\min} = \lambda_1 \leq \ldots \leq \lambda_n = \lambda_{\max}$. Es gelte $A = Q\Lambda Q^T$ mit $QQ^T = Q^TQ = I$ und $\Lambda = \text{diag}(\lambda_1, \ldots \lambda_n)$. Beachtet man $q_k(A) = q_k(Q\Lambda Q^T) = Qq_k(\Lambda)Q^T$, so schätzt man wie folgt ab

$$\begin{aligned}
\|q_k(A)e_0\|_A^2 &= e_0^T Q\, q(\Lambda)\, Q^T Q \Lambda Q^T Q\, q(\Lambda)\, Q^T e_0 = e_0^T Q\, q(\Lambda\, \Lambda\, q(\Lambda)\, Q^T e_0 \qquad (4.21) \\
&\leq \max_{\lambda \in \{\lambda_1, \ldots, \lambda_n\}} \{q(\lambda)^2\}\, e_0^T Q \Lambda Q^T e_0 \leq \left(\max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |q(\lambda)| \right)^2 e_0^T A e_0 \,,
\end{aligned}$$

d.h. es gilt

$$\|q_k(A)e_0\|_A \leq \|e_0\|_A \cdot \max_{\lambda \in [\lambda_1, \lambda_n]} |q_k(\lambda)| \,.$$

(4) Man sucht nun Polynome p_k, die die rechte Seite minimieren. Man kann zeigen, dass die *Tschebyscheff–Polynome* $T_k : \mathbb{C} \to \mathbb{C}$ $(k = 0, 1, \ldots)$ definiert als

$$T_k(x) := \frac{1}{2} \left[(x + \sqrt{x^2 - 1})^k + (x + \sqrt{x^2 - 1})^{-k} \right]$$

auf $[-1, 1]$ folgende Eigenschaften haben

$$T_k(1) = 1 \,, \ |T_k(x)| \leq 1 \quad \forall -1 \leq x \leq 1$$

und minimal bzgl. $\|\cdot\|_\infty$ unter allen Polynomen $p \in \mathbb{P}_k$ mit $p(1) = 1$ sind (siehe Aufgabe 4.6.9). Gesucht ist nun eine Transformation, die $[\lambda_{\min}, \lambda_{\max}]$ auf $[-1, 1]$ abbildet. Somit

ist $\widetilde{T}_k(z) := T_k\left(\frac{\lambda_{\max}+\lambda_{\min}-2z}{\lambda_{\max}-\lambda_{\min}}\right)$ minimal bzgl. $\|\cdot\|_\infty$ auf $[\lambda_{\min}, \lambda_{\max}]$ unter allen Polynomen aus $p \in \mathbb{P}_k$ mit $p(\lambda_{\max}) = 1$. Man wählt also

$$q_k(z) := T_k\left(\frac{\lambda_{\max}+\lambda_{\min}-2z}{\lambda_{\min}-\lambda_{\max}}\right) \Big/ T_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\min}-\lambda_{\max}}\right)$$

auf $[0, \lambda_{\max}]$, $q_k(0) = 1$
Daraus folgt

$$\min_{q_k(0)=1} \max_{\lambda\in[\lambda_1,\lambda_n]} |q_k(\lambda)| \le \frac{1}{T_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\max}-\lambda_{\min}}\right)}$$

und

$$T_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\min}-\lambda_{\max}}\right) = T_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\max}-\lambda_{\min}}\right) = T_k\left(\frac{\kappa+1}{\kappa-1}\right) \ge \frac{1}{2}(z+\sqrt{z^2-1})^k$$

mit $z = \frac{\kappa+1}{\kappa-1}$, also

$$\begin{aligned} z+\sqrt{z^2-1} &= \frac{\kappa+1}{\kappa-1} + \sqrt{\frac{\kappa^2+2\kappa+1-\kappa^2+2\kappa-1}{(\kappa-1)^2}} \\ &= \frac{1}{\kappa-1}(\kappa+1+2\sqrt{\kappa}) = \frac{(\sqrt{\kappa}+1)^2}{(\sqrt{\kappa}+1)(\sqrt{\kappa}-1)} = \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\,. \end{aligned}$$

□

Bemerkung 4.6.8 *Im letzten Beweis wurde unter (3) (siehe* (4.21)*) gezeigt, dass*

$$\|q_k(A)e_0\|_A \le \max_{\lambda\in\{\lambda_1,\ldots,\lambda_n\}} |q(\lambda)|\,\|e_0\|_A$$

gilt. Daraus folgt sofort, dass das cg-Verfahren nach m-Schritten terminiert, wenn das Spektrum von A nur m verschiedene Eigenwerte besitzt.

Aufgabe 4.6.9 (Tschebyscheff-Polynome) Die *Tschebyscheff*-Polynome T_n sind orthogonal bezüglich des Skalarprodukts

$$(f,g)_\omega := \int_{-1}^{1} \frac{f(x)g(x)}{\sqrt{1-x^2}}dx$$

und werden standardisiert durch $T_n(1) = 1$.
Man zeige, dass die *Tschebyscheff*-Polynome folgende Eigenschaften besitzen:

(i) Sie haben stets ganzzahlige Koeffizienten

(ii) Der höchste Koeffizient von T_n ist $a_n = 2^{n-1}$

(iii) T_n ist stets eine gerade Funktion, falls n gerade und eine ungerade, falls n ungerade ist

(iv) $T_n(1) = 1, T_n(-1) = (-1)^n$ (vgl. hierzu auch Beispiel 3)

(v) $|T_n(x)| \le 1$ für $x \in [-1, 1]$

(vi) Die Nullstellen von $T_n(x)$ sind

$$x_k := \cos\left(\frac{2k-1}{2n}\pi\right), \quad (k = 1, \ldots, n)$$

(vii)

$$T_k(x) = \begin{cases} \cos(k \cdot \arccos(x)), & -1 \le x \le 1; \\ \cosh(k \cdot Arccosh(x)), & x \ge 1; \\ (-1)^k \cosh(k \cdot Arccosh(-x)), & x \le -1. \end{cases}$$

$(viii)$ Die *Tschebyscheff*-Polynome besitzen die globale Darstellung

$$T_k(x) = \frac{1}{2}\left((x + \sqrt{x^2-1})^k + (x - \sqrt{x^2-1})^k\right), \quad \text{wobei } x \in \mathbb{R}$$

(ix) $|T_n(x)|$ nimmt seinen maximalen Wert im Intervall $[-1, 1]$ an den sogenannten *Tschebyscheff*-Abszissen $\overline{x}_k = \cos(\frac{k\pi}{n})$ für $k = 0, \ldots, n$ an, d.h.

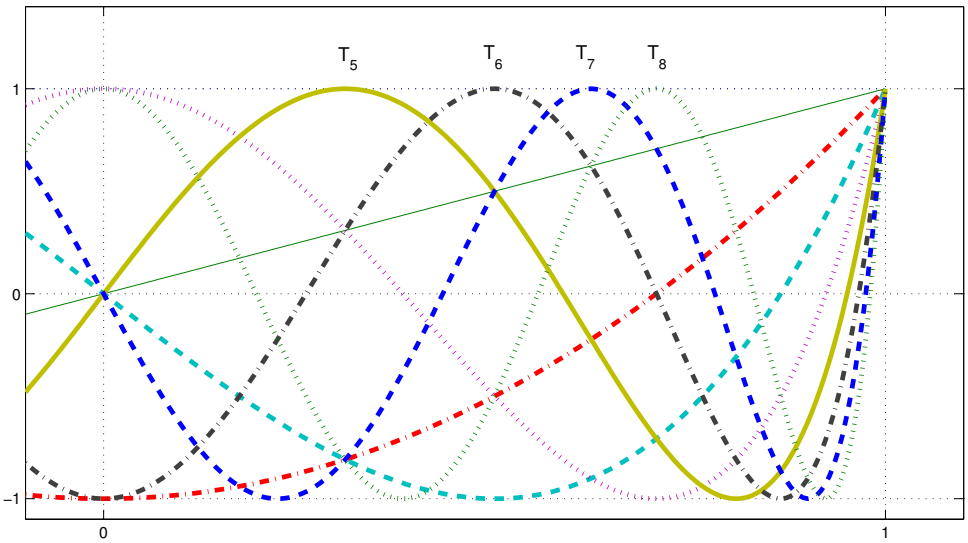
$$|T_n(x)| = 1 \Leftrightarrow x = \overline{x}_k = \cos\left(\frac{k\pi}{n}\right) \quad mit\ k = 0, \ldots, n.$$

(x) Für $x \in \mathbb{R}$ erfüllen die *Tschebyscheff*-Polynome die folgende Drei-Term-Rekursion

$$\boxed{T_0(x) = 1,\ T_1(x) = x,\ T_k(x) = 2x\,T_{k-1}(x) - T_{k-2}(x),\ \ k \ge 2}$$

Ergänzend seien für $n = 0, \ldots, 5$ die T_n explizit angeben und diese grafisch dargestellt:

$$\begin{aligned} T_0 &= 1 & T_3 &= 4x^3 - 3x \\ T_1 &= x & T_4 &= 8x^4 - 8x^2 + 1 \\ T_2 &= 2x^2 - 1 & T_5 &= 16x^5 - 20x^3 + 5x \end{aligned}$$



4.7 PRÄKONDITIONIERUNG, DAS PCG-VERFAHREN

Die Abschätzung der Konvergenzgeschwindigkeit des cg-Verfahrens hängt gemäß Satz 4.6.7 monoton von der Kondition κ der Matrix A ab. Ziel dieses Abschnittes ist es daher, das Problem

$Ax = b$ so zu transformieren, dass die enstehende Matrix möglichst „gut konditioniert“ (d.h. κ möglichst klein) ist.
Idee: Betrachte anstatt

$$Ax = b \text{ mit } A \text{ s.p.d.} \tag{4.22}$$
$$\bar{A}\bar{x} = b \text{ mit } \bar{x} = B^{-1}x \text{ und } \bar{A} = AB \tag{4.23}$$

mit einer invertierbaren Matrix $B \in \mathbb{R}^{n\times n}$ und wende hierauf ein iteratives Lösungsverfahren an. Um die Symmetrie des Problems zu erhalten, betrachtet man im Falle, dass B ebenfalls s.p.d. ist, anstatt des Euklidischen Skalarproduktes $\langle\cdot,\cdot\rangle$ das von der Matrix B iduzierte Skalarprodukt $\langle\cdot,\cdot\rangle_B := \langle\cdot, B\cdot\rangle$. Aufgrund der folgenden Gleichung

$$\langle x, \bar{A}y\rangle_B = \langle x, ABy\rangle_B = \langle x, BABy\rangle = \langle ABx, By\rangle = \langle ABx, y\rangle_B = \langle \bar{A}x, y\rangle_B$$

ist $\bar{A} = AB$ bezüglich $\langle\cdot,\cdot\rangle_B$ selbstadjungiert. Man kann daher zur Lösung von $\bar{A}\bar{x} = b$ das cg-Verfahren anwenden, vorausgesetzt man berücksichtigt, dass hierbei an die Stelle des Euklidischen Skalarproduktes die Form $\langle\cdot,\cdot\rangle_B$ rückt. Da man jedoch nicht an dem Vektor $\bar{x}$, sondern an der eigentlichen Lösung $x = B\bar{x}$ von $Ax = b$ interessiert ist, ersetzt man die so bestimmten Iterierten $\bar{x}_k$ durch $x_k = B\bar{x}_k$. Hieraus ergibt sich unmittelbar der Algorithmus 4.7.1.

Algorithmus 4.7.1: Präkonditioniertes Konjugiertes Gradientenverfahren (pcg-Verfahren)

Input: `Initial guess` x_0

$r_0 := b - Ax_0$

$\rho_0 := \langle r_0, Br_0\rangle$

$d_1 := Br_0$

Iteration: $k = 1, 2, \ldots$ `as long as` $k \le n$ `and` $r_k \neq 0$

$a_k := Ad_k$

$\alpha_k := \rho_{k-1} \,/\, \langle d_k, a_k\rangle$

$x_k := x_{k-1} + \alpha_k\, d_k$

$r_k := r_{k-1} - \alpha_k\, a_k$

$\rho_k := \langle r_k, Br_k\rangle$

$d_{k+1} := Br_k + \frac{\rho_k}{\rho_{k-1}}\, d_k$

Pro Iterationsschritt benötigt dieser Algorithmus gegenüber dem cg-Verfahren nur eine Multiplikation mit der Matrix B mehr. Doch dieser zusätzliche Aufwand rentiert sich, sofern sich die Kondition $\kappa(\bar{A})$ der Matrix $\bar{A} = AB$ gegenüber der Konditionszahl $\kappa(A)$ des nicht präkonditionierten Systems „verbessert“, d.h. sie nimmt ab.

Beispiel 4.7.1 Eine sehr einfache, aber häufig schon wirkungsvolle Vorkonditionierung einer s.p.d. Matrix A mit nichtverschwindenden Diagonalelementen liefert die Wahl $B := D^{-1}$, also der Inversen der Diagonale von A. Man spricht in diesem Fall von *diagonaler Präkonditionierung*.

Bemerkung 4.7.2 *Es ist klar, dass die Wahl des Präkonditionierers B elementar von den Eigenschaften der Matrix A abhängt. Die Wahl eines geeigneten Präkonditionierers ist daher oft nicht leicht. Hat man aber einen gefunden, so erhöht sich die Konvergenzgeschwindigkeit beträchtlich, was besonders für große lineare Gleichungssysteme von Bedeutung ist.*

5 AUSGLEICHSPROBLEME

Motivation Gesucht ist eine Gerade $g(x) = mx + b$, deren y-Werte den kleinsten Quadratsummenabstand zu den vorgegebenen Daten $\big(x_i, f(x_i)\big)$ haben, die sogenannte *Ausgleichsgerade*. Die geometrische Veranschaulichung dazu ist in der nachfolgenden Abbildung dargestellt.

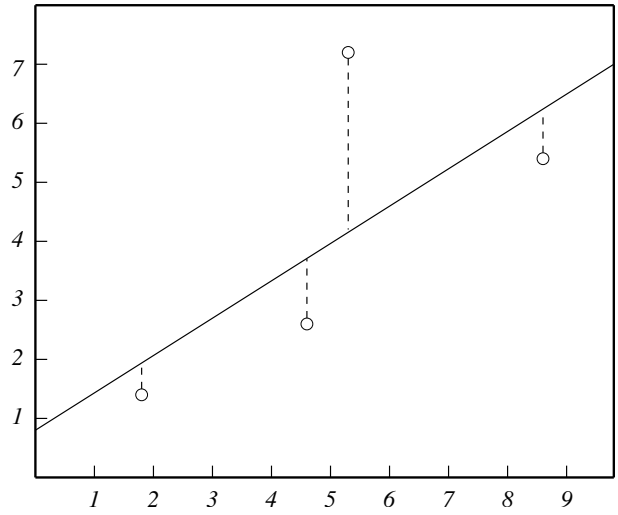


Abb. 5.1: Die Ausgleichsgerade, die die Quadratsumme der Abstände minimiert

5.1 DIE METHODE DER KLEINSTEN QUADRATE

Gemäß obiger Motivation gilt es zu gegebenen Punktepaaren $(x_1, f_1), (x_2, f_2), \ldots, (x_n, f_n)$ die in der Geradengleichung $g(x) = mx + b$ freien Parameter m und b so zu bestimmen, dass gilt

$$F(m, b) := \sum_{i=1}^{n} \big(g(x_i) - f_i\big)^2 \to \min,$$

oder in alternativer Formulierung

$$\min_{(b,m)\in\mathbb{R}^2} \|A\begin{pmatrix} b \\ m \end{pmatrix} - d\|_2^2,$$

wobei wir nachfolgende Definitionen verwenden

$$A = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad d = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}.$$

Hierzu bilden wir alle partiellen Ableitungen bzgl. m und b, woraus sich dann die kritischen Punkte der Funktion $F(m, b)$ wie folgt berechnen

$$\nabla F(m, b) = \Big(\frac{\partial}{\partial b} F, \frac{\partial}{\partial m} F\Big) = \overrightarrow{0}.$$

Im vorliegenden Fall erhalten wir damit

$$\begin{aligned}\frac{\partial}{\partial b}F(m,b) &= 2\sum_{i=1}^{n}(mx_i+b-f_i)\cdot 1 \stackrel{!}{=} 0\\ \frac{\partial}{\partial m}F(m,b) &= 2\sum_{i=1}^{n}(mx_i+b-f_i)\cdot x_i \stackrel{!}{=} 0.\end{aligned}$$

Diese beiden Bestimmungsgleichungen können auch wie folgt geschrieben werden

$$\begin{aligned}1^T\Big(A\begin{pmatrix}b\\m\end{pmatrix}-d\Big) &= 0\\ x^T\Big(A\begin{pmatrix}b\\m\end{pmatrix}-d\Big) &= 0,\end{aligned}$$

wobei $1=(1,\ldots,1)^T\in\mathbb{R}^n$ und $x=(x_1,\ldots,x_n)\in\mathbb{R}^n$ seien.
Es gilt also

$$\begin{aligned}\nabla F(m,b)=0 \quad\Rightarrow\quad & 1^T\Big(A\begin{pmatrix}b\\m\end{pmatrix}-d\Big)=0\\ & x^T\Big(A\begin{pmatrix}b\\m\end{pmatrix}-d\Big)=0.\end{aligned}$$

Zusammengefasst folgt für potentielle Minimalstellen

$$\begin{pmatrix}1 & x_1\\ \vdots & \vdots\\ 1 & x_n\end{pmatrix}^T\Big(A\begin{pmatrix}b\\m\end{pmatrix}-d\Big)=\begin{pmatrix}0\\0\end{pmatrix}\quad\Leftrightarrow\quad \boxed{A^TA\tbinom{b}{m}=A^Td}$$

Bemerkung 5.1.1 *Falls in einem linearen Gleichungssystem die Anzahl der Gleichungen die der Unbekannten übersteigen sollte (wie es im vorliegenden Fall bei $Ax=d$ ist), so nennen wir das System* **überbestimmt** *und können es in der Regel nicht exakt lösen. Deshalb betrachten wir sinnvollerweise das Ausgleichsproblem der* **Gaußschen Normalengleichung**

$$A^TAx=A^Td \tag{5.1}$$

und erhalten dafür genau dann eine eindeutige Lösung, falls A vollen Rang besitzt.

Bemerkung 5.1.2 *Als einführendes Beispiel haben wir uns auf die Betrachtung von linear unabhängigen Basisvektoren des $\mathbb{P}_1$ in der Darstellung von g beschränkt, d.h. 1, x. Als Nächstes werden wir zu allgemeineren Funktionen in der Funktionsvorschrift von g übergehen, wie etwa*

$$g(x)=a\exp(x)+b\exp(-x)+c\log(x).$$

Damit erhalten wir folgendes Minimierungsproblem bei bekannten Daten (x_i,f_i) $(i=1,\ldots,n)$

$$\sum_{i=1}^{n}\big(g(x_i)-f_i\big)^2\to\min,$$

welches wir völlig analog zu obigem Verfahren als Normalengleichung in der Form

$$A^TA\begin{pmatrix}a\\b\\c\end{pmatrix}=A^Td,\ \text{mit } A=\begin{pmatrix}e^{x_1} & e^{-x_1} & \log x_1\\ \vdots & \vdots & \vdots\\ e^{x_n} & e^{-x_n} & \log x_n\end{pmatrix}\text{ und } d=\begin{pmatrix}f_1\\ \vdots\\ f_n\end{pmatrix}$$

interpretieren können. Je komplexer also die Funktionsvorschrift von g, desto aufwendiger wird auch das Lösen der Gaußschen Normalengleichung (5.1). Im Folgenden werden wir ein Verfahren bereitstellen, das eine numerisch stabile und effiziente Lösung des Ausgleichsproblems garantiert.

5.2 DIE *QR*-ZERLEGUNG

Definition 5.2.1 *Eine quadratische Matrix $A \in \mathbb{R}^{n\times n}$ heißt* orthogonal, *falls gilt:*

$$AA^T = A^TA = I.$$

Weiterhin bezeichne $\|x\|$ die vom Euklidschen *Skalarprodukt im $\mathbb{R}^n$ induzierte Norm in der Form*

$$\|x\| := \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x^Tx}.$$

Darüber hinaus halten wir noch fest, dass orthogonale Abbildungen stets längen- und winkeltreu sind, dass also gilt

$$\|Qx\|^2 = (Qx)^TQx = x^TQ^TQx = x^Tx = \|x\|^2, \text{ d.h. } \|Qx\| = \|x\|.$$

Motivation Zur Lösung des Minimierungsproblems $\min_{x\in\mathbb{R}^n} \|Ax-b\|^2$ setzen wir voraus, dass zu $A \in \mathbb{R}^{m\times n}$ (rg $A = n < m$) stets eine Zerlegung $A = QR$ existiert, wobei $Q \in \mathbb{R}^{m\times m}$ orthogonal ist und R folgende Gestalt besitzt

$$R = \begin{pmatrix} * & * & * \\ & * & * \\ & & * \\ & & \\ 0 & & \end{pmatrix} = \begin{pmatrix} \widehat{R} \\ \hline \\ 0 \end{pmatrix},$$

$\widehat{R}$ stellt dabei eine reguläre obere Dreiecksmatrix dar. Damit können wir unser bisheriges Minimierungsproblem wie folgt modifizieren (beachte die Faktorisierung $A = QR$):

$$\|Ax-b\|^2 = \min \;\Leftrightarrow\; \|Q^T(Ax-b)\|^2 = \min \;\Leftrightarrow\; \|Rx - Q^Tb\|^2 = \min,$$

hierbei weisen wir vor allen Dingen auf die Gestalt von R hin, als

$$Rx = \left(\begin{array}{ccc} * & * & * \\ & * & * \\ & & * \\ \hline 0 & & \end{array}\right) \cdot x = \left(\begin{array}{c} b_1 \\ \hline b_2 \end{array}\right) = Q^Tb$$

Dieser Faktorisierungsansatz führt uns nun zu folgenden Fragen:

- Zu welchen Matrizen $A \in \mathbb{R}^{m\times n}$ existiert eine derartige *QR*-Zerlegung?
- Wie bestimmen wir eine solche Zerlegung möglichst effizient?

Satz 5.2.2 *Zu jeder Matrix $A \in \mathbb{R}^{m\times n}$ mit Maximalrang $n < m$ existiert eine orthogonale Matrix $Q \in \mathbb{R}^{m\times m}$ derart, dass*

$$A = QR \text{ mit } R = \begin{pmatrix} \widehat{R} \\ 0 \end{pmatrix},\ \widehat{R} \in \mathbb{R}^{n\times n}$$

gilt, wobei $\widehat{R}$ eine reguläre obere Dreiecksmatrix darstellt.

Beweis. Der Kern des Beweises liegt in der Verwendung von orthogonalen Drehmatrizen der Form

$$\mathcal{U}(p,q;\varphi) := \begin{pmatrix} 1 & & & & & & & & 0 \\ & 1 & & & & & & & \\ & & \ddots & & & & & & \\ & & & \cos\varphi & & \sin\varphi & & & \\ & & & & \ddots & & & & \\ & & & -\sin\varphi & & \cos\varphi & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \\ 0 & & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \xleftarrow{p-te\ Zeile} \\ \\ \xleftarrow{q-te\ Zeile} \\ \\ \\ \end{matrix} \tag{5.2}$$

$p-te\ Spalte$ $q-te\ Spalte$

Die Orthogonalität von $\mathcal{U}(p,q;\varphi)$ ergibt sich unmittelbar, denn es gilt $\mathcal{U}^T(p,q;\varphi)\cdot\mathcal{U}(p,q;\varphi) = I$. Das Produkt $\mathcal{U}^T(p,q;\varphi)\cdot A$ liefert dagegen für $j = 1,\ldots,n$

$p-te\ Spalte$ $q-te\ Spalte$

$$\begin{pmatrix} 1 & & & & & & & & 0 \\ & \ddots & & & & & & & \\ & & \cos\varphi & & & -\sin\varphi & & & \\ & & & \ddots & & & & & \\ & & \sin\varphi & & & \cos\varphi & & & \\ & & & & & & \ddots & & \\ 0 & & & & & & & & 1 \end{pmatrix} \cdot A \quad = \quad \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{p-1,1} & \cdots & a_{p-1,n} \\ a_{p1}\cos\varphi - a_{q1}\sin\varphi & \cdots & a_{pn}\cos\varphi - a_{qn}\sin\varphi \\ a_{p+1,1} & \cdots & a_{p+1,n} \\ \vdots & & \vdots \\ a_{q-1,1} & \cdots & a_{q-1,n} \\ a_{p1}\sin\varphi + a_{q1}\cos\varphi & \cdots & a_{pn}\sin\varphi + a_{qn}\cos\varphi \\ a_{q+1,1} & \cdots & a_{q+1,n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

Demzufolge werden die Zeilen p und q von A durch Linearkombinationen der cos- und sin-Terme ersetzt.
Die sukzessive Multiplikation der Matrix A von links mit Rotationsmatrizen $\mathcal{U}^T(p,q;\varphi)$ (φ dabei so gewählt, dass $\widetilde{a}_{qp} = 0$) mit den Rotationsindexpaaren

$$(1,2),(1,3),\ldots,(1,m),(2,3),(2,4),\ldots,(2,m),(3,4),\ldots,(n,m)$$

eliminiert dann die Matrixelemente $a_{21}, a_{31}, \ldots, a_{mn}$. Sukzessive werden dabei unterhalb des Diagonalelements Nullen erzeugt. Nach $k := \frac{1}{2}n(2m-n-1)$ Transformationsschritten gilt dann $A = QR$ mit

$$\mathcal{U}_k^T \cdot \ldots \cdot \mathcal{U}_2^T\mathcal{U}_1^T A = Q^T A = R \quad \text{oder} \quad A = QR.$$

Da die orthogonale Matrix Q regulär ist, ist der Rang von A und der Rang von R gleich n und folglich ist die Rechtecksdreiecksmatrix $\widehat{R}$ regulär, da A nach Voraussetzung Maximalrang hat. □

Der soeben erbrachte Existenzbeweis ist rein konstruktiv und liefert damit eine erste Möglichkeit eine Zerlegung QR zu bestimmen. Wir unterscheiden die folgenden beiden Vorgehensweisen:

- *Givens*[1]-Rotation
- *Householder*[2]-Spiegelung

5.3 *Givens*-ROTATIONEN

Es bezeichne im Folgenden stets $c = \cos\varphi$ und $s = \sin\varphi$. Die Grundidee der *Givens*-Rotation besteht darin durch die Multiplikation einer geeigneten orthogonalen (Dreh-)Matrix einen Vektor $x \in \mathbb{R}^n$ so zu drehen, dass möglichst viele seiner Komponenten verschwinden.
Im Falle $(a, b) \in \mathbb{R}^2$ werden wir also $c, s \in \mathbb{R}$ so bestimmen, dass gilt

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}.$$

Setzen wir zuerst $a,\ b \neq 0$ voraus. Wir erhalten damit folgende Gleichung

$$as = -bc \tag{5.3}$$

und mit $s^2 + c^2 = 1$ gilt

$$r^2 = (ac - bs)^2 = a^2c^2 - abcs - abcs + b^2s^2 \overset{(5.3)}{=} a^2c^2 + a^2s^2 + b^2c^2 + b^2s^2 = a^2 + b^2\ .$$

Weiter ergibt sich für $a \neq 0$

$$\begin{aligned} ac - bs = r \Rightarrow c = \frac{1}{a}(r + bs) \quad &\Rightarrow \quad as + \frac{b}{a}(r + bs) = 0 \\ &\Leftrightarrow \quad a^2s + br + b^2s = 0 \\ &\Leftrightarrow \quad br = -(a^2 + b^2)s \\ &\Leftrightarrow \quad s = \frac{-br}{a^2 + b^2} \overset{(r^2 = a^2 + b^2)}{=} \frac{-b}{r}. \end{aligned}$$

Und damit erhalten wir dann auch für $b \neq 0$

$$-bc = as \ \Rightarrow\ c = -\frac{a}{b}s = \frac{a}{r}.$$

Offen ist noch die Wahl des Vorzeichens des Vorzeichens $r = \pm\sqrt{a^2 + b^2}$. Setzen wir $r = \text{sign}(a)\sqrt{a^2 + b^2}$, so folgt $c > 0$, falls $a \neq 0$, und für $a = 0$ setzen wir $c = 0$ und $s = 1$. Somit sind auch s und c für die anderen Spezialfälle $b = 0$ und $a = b = 0$ wohldefiniert. Damit gilt (obwohl der Winkel bisher gar nicht explizit genannt wurde) $\varphi \in (-\frac{\pi}{2}, \frac{\pi}{2}]$ und somit lässt sich $\cos\varphi$ eindeutig aus $\sin\varphi$ mit $\cos\varphi = \sqrt{1 - \sin^2\varphi}$ bestimmen.
Seien p und q zwei Zeilenindizes von A. Zur Eliminierung des Matrixelements a_{qj} gehen wir daher wie folgt vor:

$$\begin{aligned} \text{Falls} \quad a_{pj} \neq 0: \quad \cos\varphi &= \frac{|a_{pj}|}{\sqrt{a_{pj}^2 + a_{qj}^2}}, \\ \sin\varphi &= \frac{-\text{sign}(a_{pj})\ a_{qj}}{\sqrt{a_{pj}^2 + a_{qj}^2}}, \\ a_{pj} = 0: \quad \cos\varphi &= 0,\ \ \sin\varphi = 1. \end{aligned}$$

[1] Givens, W.
[2] Householder, A.

Da sich $\cos\varphi$ aus $\sin\varphi$ bestimmen lässt, ermöglicht das Verfahren eine effiziente Speicherung, denn es genügt $\sin\varphi$ an den entsprechenden freiwerdenden Stellen in A abzulegen:

$$A = \begin{pmatrix} a_{11}^{(0)} & \cdots & a_{1n}^{(0)} \\ \vdots & & \vdots \\ a_{m1}^{(0)} & \cdots & a_{mn}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{11}^{(1)} & \cdots & \cdots & a_{1n}^{(1)} \\ \sin\varphi_{21} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ a_{31}^{(0)} & a_{32}^{(0)} & \cdots & a_{3n}^{(0)} \\ a_{41}^{(0)} & a_{42}^{(0)} & \cdots & a_{4n}^{(0)} \\ \vdots & & & \vdots \\ a_{m1}^{(0)} & \cdots & \cdots & a_{mn}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{11}^{(2)} & \cdots & \cdots & a_{1n}^{(2)} \\ \sin\varphi_{21} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \sin\varphi_{31} & a_{32}^{(2)} & \cdots & a_{3n}^{(2)} \\ a_{41}^{(0)} & a_{42}^{(0)} & \cdots & a_{4n}^{(0)} \\ \vdots & & & \vdots \\ a_{m1}^{(0)} & \cdots & \cdots & a_{mn}^{(0)} \end{pmatrix}$$

$$\rightsquigarrow \begin{pmatrix} a_{11}^{(3)} & \cdots & \cdots & a_{1n}^{(3)} \\ \sin\varphi_{21} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \sin\varphi_{31} & a_{32}^{(2)} & \cdots & a_{3n}^{(2)} \\ \sin\varphi_{41} & a_{42}^{(3)} & \cdots & a_{4n}^{(3)} \\ \vdots & & & \vdots \\ a_{m1}^{(0)} & \cdots & \cdots & a_{mn}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{11}^{(m-1)} & \cdots & \cdots & a_{1n}^{(m-1)} \\ \sin\varphi_{21} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \sin\varphi_{31} & a_{32}^{(2)} & \cdots & a_{3n}^{(2)} \\ \sin\varphi_{41} & a_{42}^{(3)} & \cdots & a_{4n}^{(3)} \\ \vdots & & & \vdots \\ \sin\varphi_{m1} & a_{m2}^{(m-1)} & \cdots & a_{mn}^{(m-1)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} \ddots & & \widehat{R} \\ & \ddots & \\ \text{„Q“} & & \ddots \end{pmatrix}.$$

Die *QR*-Zerlegung liefert also QR in kompakter Form, gespeichert in A.
Der Aufwand dieser Vorgehensweise liegt bei ungefähr $\mathcal{O}(n^3)$ Operationen.

Bemerkung 5.3.1 *Die Multiplikation eines gegebenen Vektors $x \in \mathbb{R}^n$ mit der Drehmatrix $\mathcal{U}(p,q;\varphi)$ ist äquivalent zur Drehung von x um einen Winkel φ entgegen dem Uhrzeigersinn in der Koordinatenebene. Vergleiche hierzu die nachfolgende Abbildung, die diesen geometrischen Sachverhalt verdeutlicht:*

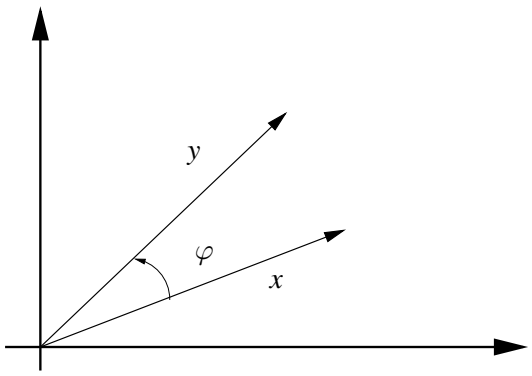


Abb. 5.2: Drehung um einen Winkel φ in der Ebene

MATLAB-Funktion: GivensRotMat.m

```
function rot = GivensRotMat(ap,aq)
if ap == 0
  s = 1; c = 0;
else
  dist2 = sqrt(ap^2+aq^2);
  c = abs(ap)/dist2;
  s = -sign(ap)*aq/dist2;
end
rot = [c,s;-s,c];
```

MATLAB-Funktion: Givens.m

```
function [Q,A] = Givens(A)
Q = eye(size(A,1));
for j=1:size(A,2)
  for i=j+1:size(A,1)
    rot = GivensRotMat(A(j,j),A(i,j));
    A([j,i],j:end) = rot' * A([j,i],j:end);
    Q(:,[j,i]) = Q(:,[j,i]) * rot;
  end
  A(j+1:end,j) = 0;
end
```

MATLAB-Beispiel:

Alternativ zu der Matlab-Funktion `qr` können wir auch die Routine `Givens` verwenden, die eine QR-Zerlegung, wie oben diskutiert, berechnet.

```
>> A = [1,2;3,4;5,6];
> [Q,R] = Givens(A);
>> Q*R
ans =
    1.0000    2.0000
    3.0000    4.0000
    5.0000    6.0000
```

5.4 UPDATE EINER QR-ZERLEGUNG

Bei verschiedenen numerischen Verfahren müssen wiederholt QR-Zerlegungen bestimmt werden, wobei sich die Matrizen nur wenig voneinander unterscheiden. Dies ist z.B. der Fall beim Broyden-Verfahren (siehe Numerik II, nichtlineare Gleichungen). Sei die Zerlegung $A_k = Q_k R_k$ gegeben und die QR-Zerlegung von $A_{k+1} = A_k + st^T$ zu bestimmen. Wir werden zeigen, dass

anstatt jeweils $\mathcal{O}(n^3)$ Operationen für eine vollständige QR-Zerlegung zu verwenden, für jeden einzelnen Updateschritt $\mathcal{O}(n^2)$ Operationen genügen.
Wir benötigen noch einige Hilfsmittel, um diese Aussage zu zeigen.

Definition 5.4.1 (Hessenberg-Matrix) *Eine Matrix $A \in \mathbb{R}^{n\times n}$ der Form*

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

d.h. $a_{ij} = 0$ für $i - j \geq 2$, wird obere **Hessenberg-Matrizen** *genannt. Gilt $a_{ij} = 0$ für $j - i \geq 2$, so hat die Matrix A untere Hessenberg-Gestalt.*

Lemma 5.4.2 *Es seien $u, v \in \mathbb{R}^n$ und $R \in \mathbb{R}^{n\times n}$ eine obere Dreiecksmatrix. Dann benötigt man $n - 2$ Givens-Rotationen, um $R + uv^T$ auf obere Hessenberg-Gestalt zu transformieren.*

Beweis. Multipliziert man R sukzessive von links mit Drehmatrizen $\mathcal{U}^T(p, q)$ (siehe (5.2)) zu Indexpaaren $(n, n-1)$, $\ldots$, $(4, 3)$,$(3, 2)$ so hat QR obere Hessenberg-Gestalt, wobei $Q := \mathcal{U}_{32} \cdots \mathcal{U}_{n,n-1}$ ist. Dabei lassen sich die Drehmatrizen so wählen, dass $Qu = (*, *, 0, \ldots, 0)^T$ gilt. Beachtet man nun, dass $(Qu)v^T$ und somit auch $Q(R+uv^T)$ obere Hessenberg-Form hat, so ist die Aussage bewiesen. □

Bemerkung 5.4.3 *Beachtet man, dass sich jede Givens-Rotation angewandt auf $A \in \mathbb{R}^{n\times n}$ durch $\mathcal{O}(n)$ Operationen realisieren lässt, so genügen $\mathcal{O}(n^2)$ Operationen um eine orthogonale Matrix $\widetilde{Q}$ und eine Hessenberg-Matrix $\widetilde{H}$ mit $\widetilde{Q}\widetilde{H} = R + uv^T$ zu bestimmen.*

Bemerkung 5.4.4 *Es genügen $n - 1$ Givens-Rotation um eine Hessenberg-Matrix $A \in \mathbb{R}^{n\times n}$ auf Dreiecksgestalt zu transformieren.*

Kommen wir nun zu unserer ursprünglichen Fragestellung zurück, d.h. der effizienten Update-Berechnung. Der Ansatz

$$A_{k+1} = Q_{k+1}R_{k+1} = Q_k(R_k + uv^T) = A_k + Q_k uv^T = A_k + st^T$$

liefert $v = t$ und $s = Q_k u$ bzw. $u = Q_k^T s$. Das Update besteht nun aus zwei Schritten.

1. Mittels $n - 2$ Givens-Rotationen wird $R_k + uv^T$ auf eine obere Hessenbergform gebracht, d.h. wir bestimmen eine orthogonale Matrix Q'_{k+1} und eine Hessenberg-Matrix H_{k+1} mit $Q'_{k+1}H_{k+1} = R_k + uv^T = R_k + (Q_k^T s)t^T$.

2. Mittels $n - 1$ Givens-Rotationen wird diese obere Hessenbergform zur neuen Matrix R_{k+1} transformiert, d.h. wir bestimmen eine orthogonale Matrix Q''_{k+1} und eine obere Dreiecksmatrix R_{k+1} mit $Q''_{k+1}R_{k+1} = H_{k+1}$. Die Matrix Q_{k+1} ergibt sich dann als Produkt $Q_k \cdot Q'_{k+1} \cdot Q''_{k+1}$.

MATLAB-Funktion: QR2Hessenberg.m

```
function [Q,R] = QR2Hessenberg(Q,R,s,t)
% computes QH-decomposition of Q(R+s*t), H upper Hessenberg form
for j=size(R,1):-1:3
  rot = GivensRotMat(s(j),s(j-1));
  s([j,j-1]) = rot * s([j,j-1]);
  Q(:,[j,j-1]) = Q(:,[j,j-1]) * rot';
  R([j,j-1],:) = rot * R([j,j-1],:);
end
R(1:2,:) = R(1:2,:)+ s(1:2) * t';
```

MATLAB-Funktion: GivensHessenberg.m

```
function [Q,H] = GivensHessenberg(Q,H)
% computes QR-decomposition of QH, H Hessenberg matrix
for j=1:min(size(H,1)-1,size(H,2))
  rot = GivensRotMat(H(j+1,1),H(j,1));
  H([j+1,j],j:end) = rot * H([j+1,j],j:end);
  Q(:,[j+1,j]) = Q(:,[j+1,j]) * rot';
  H(j+1,j) = 0;
end
```

MATLAB-Beispiel:

Ein einfaches Beispiel zeigt, dass man das $\mathcal{O}(n^2)$-Verhalten des Updates auch im Vergleich zur optimierten Matlab-Routine `qr` sieht und nutzen kann. Verdoppelt man z.B. n auf $n = 4000$ im nebenstehenden Listing, so benötigt das Update nur $2^2 = 4$-mal länger, die interne Funktion jedoch $2^3 = 8$-mal. Testen Sie es einmal!

```
>> n = 2000;
>> A=rand(n); s=rand(n,1); t=rand(n,1);
>> An = A + s * t';
>> [Q,R] = qr(A);
>> tic
>> [Qtmp,H] = QR2Hessenberg(Q,R,Q'*s,t);
>> [Qn,Rn] = GivensHessenberg(Qtmp,H);
>> toc
Elapsed time is 1.078000 seconds.
>> tic, [Qn,Rn] = qr(An); toc
Elapsed time is 5.766000 seconds.
```

5.5 *Householder*-SPIEGELUNGEN

Definition 5.5.1 *Sei $\omega \in \mathbb{R}^n$. Die $n \times n$- Matrix*

$$P = I - 2\frac{\omega\omega^T}{\omega^T\omega} \tag{5.4}$$

wird als Householder-Transformation *und ω als* Householder-Vektor *bezeichnet.*

Satz 5.5.2 *Eine* Householder-*Transformation $P = I - 2(\omega\omega^T)/(\omega^T\omega)$ ist symmetrisch und orthogonal, d.h. $P^{-1} = P^T$.*

Beweis. Da $\omega\omega^T$ symmetrisch ist $\left((\omega\omega^T)^T = \omega\omega^T\right)$, folgt

$$P^T = \left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right)^T = I - 2\frac{\omega\omega^T}{\omega^T\omega} = P\,.$$

Des Weiteren ergibt sich

$$P\,P^T = \left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right)\left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right) = I - 2\frac{\omega\omega^T}{\omega^T\omega} - 2\frac{\omega\omega^T}{\omega^T\omega} + 4\frac{\omega\omega^T\omega\omega^T}{(\omega^T\omega)^2}$$

und mit $\omega\omega^T\omega\omega^T = (\omega^T\omega)\,(\omega\omega^T)$ somit die Behauptung. □

Betrachten wir nun folgende Frage: Sei $x \in \mathbb{R}^n$. Wie müsste ein $\omega \in \mathbb{R}^n$ aussehen, so dass $Px = \alpha e_1$ gelte ($\alpha \in \mathbb{R}$, e_1 erster kanonischer Einheitsvektor)?
Mit $Px = x - 2\frac{\omega\omega^T}{\omega^T\omega}x = x - \lambda\omega \stackrel{!}{=} \alpha e_1$ folgt $\omega \in span\{x - \alpha e_1\}$.
Wir setzen nun $\omega = x - \alpha e_1$ in $Px = \alpha e_1$ ein und erhalten

$$Px = x - 2\frac{\omega^T x}{\omega^T\omega}\,\omega = \left(1 - \frac{2(x-\alpha e_1)^T x}{\|x - \alpha e_1\|^2}\right)x + \alpha\frac{2(x-\alpha e_1)^T x}{\|x-\alpha e_1\|^2}e_1 = \alpha e_1\,.$$

Damit in der letzten Gleichung der Faktor vor x verschwindet, muss

$$1 = \frac{2(x-\alpha e_1)^T x}{\|x-\alpha e_1\|^2} \Leftrightarrow (x-\alpha e_1)^T(x-\alpha e_1) = 2x^Tx - 2\alpha x_1 \Leftrightarrow \alpha = \pm\sqrt{x^Tx}$$

gelten.
Wie ist nun das Vorzeichen zu wählen, $\alpha = \pm\sqrt{x^Tx}$? Die Wahl $\alpha = \|x\|_2$ hat die schöne Eigenschaft, dass Px ein positives Vielfaches von e_1 ist. Aber das Rezept ist gefährlich, wenn x annähernd ein positives Vielfaches von e_1 ist, da Auslöschungen auftreten können. Berechnet man ω_1 mittels $\omega_1 = x_1 - \|x\|_2$ treten für $x_1 \leq 0$ keine Auslöschungen auf und mit der Umformung

$$\omega_1 = x_1 - \|x\|_2 = \frac{x_1^2 - \|x\|_2^2}{x_1 + \|x\|_2} = \frac{-(x_2^2 + \ldots + x_n^2)}{x_1 + \|x\|_2}$$

treten auch für $x_1 > 0$ keine Auslöschungen auf. Man beachte außerdem $(\omega_2, \ldots \omega_n) = (x_2, \ldots, x_n)$.
Normiert man den Vektor ω so, dass $\omega_1 = 1$ gilt, d.h.

$$\omega := \frac{x - \alpha e_1}{x_1 - \alpha} \text{ mit } \alpha^2 = \|x\|^2 \quad (\omega^T e_1 = 1),$$

dann folgt

$$\omega^T\omega = \frac{(x-\alpha e_1)^T(x-\alpha e_1)}{(x_1-\alpha)^2} = \frac{\|x\|^2 - 2\alpha x^T e_1 + \alpha^2}{(x_1-\alpha)^2} = \frac{2\alpha(\alpha - x_1)}{(x_1-\alpha)^2} = \frac{2\alpha}{\alpha - x_1}\,.$$

Des Weiteren bleibt festzuhalten, dass bei der Speicherung der Matrix $P = I - 2(\omega\omega^T)/(\omega^T\omega) \in \mathbb{R}^{n\times n}$ auch nur der Vektor $(\omega_2, \ldots, \omega_n)^T$ zu berücksichtigen ist, der sich auf den $n-1$ freigewordenen Einträgen speichern lässt.

Algorithmus 5.5.1: Berechnung Householder-Vektor:

Zu gegebenem $x \in \mathbb{R}^n$ berechnet die Funktion ein $\omega \in \mathbb{R}^n$ mit $\omega(1) = 1$ und $\beta \in \mathbb{R}$, so dass $P = I_n - \beta\omega\omega^T$ orthogonal ist und $Px = \|x\|_2 e_1$.

function: $[\omega, \beta] = housevector(x)$
$n = length(x)$
$\sigma = x(2:n)^T x(2:n)$
$\omega = \begin{bmatrix} 1 \\ x(2:n) \end{bmatrix}$
if $\sigma = 0$
  $\beta = 0$
else
  $\mu = \sqrt{x(1)^2 + \sigma}$
  if $x(1) \leq 0$
    $\omega(1) = x(1) - \mu$
  else
    $\omega(1) = -\sigma/(x(1) + \mu)$
  end
  $\beta = 2\,\omega(1)^2/(\sigma + \omega(1)^2)$
  $\omega = \omega/\omega(1)$
end

MATLAB-Funktion: HouseholderVektor.m

```
1  function [v,beta] = HouseholderVector(x)
2  n = length(x);
3  if n>1
4    sigma = x(2:end)'*x(2:end);
5    if sigma==0
6      beta = 0;
7    else
8      mu = sqrt(x(1)^2+sigma);
9      if x(1)<=0
10       tmp = x(1) - mu;
11     else
12       tmp = -sigma / (x(1) + mu);
13     end
14     beta = 2*tmp^2/(sigma + tmp^2);
15     x(2:end) = x(2:end)/tmp;
16   end
17   v = [1;x(2:end)];
18 else
19   beta = 0;
20   v = 1;
21 end
```

Wir haben gerade konstruktiv gezeigt, dass sich mit Hilfe der *Householder*-Transformation eine Matrix $A \in \mathbb{R}^{m\times n}$ in eine Matrix der folgenden Form transformieren können:

$$H_1 A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}$$

Gehen wir nun davon aus, dass wir nach einigen Schritten die Ausgangsmatrix A auf folgende Gestalt gebracht haben:

$$H_2 H_1 A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \end{pmatrix}$$

Im nächsten Schritt soll nun die nächste Subspalte unterhalb der Diagonalen eliminiert werden. Es ist also eine Householder-Matrix zu bestimmen, so dass

$$\widetilde{H}_3 \begin{pmatrix} \boxplus \\ \boxplus \\ \boxplus \\ \boxplus \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

gilt. Definiert man H_3 als Blockdiagonalmatrix

$$H_3 := \begin{pmatrix} I & 0 \\ 0 & \widetilde{H}_3 \end{pmatrix}$$

so erhalten wir

$$H_3 H_2 H_1 A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

Mit dieser Vorgehensweise erhalten wir folgende Faktorisierung:

$$R = H_{n-1} H_{n-2} \cdot \ldots \cdot H_1 A \;\Leftrightarrow\; A = (H_1 \cdot \ldots \cdot H_{n-1}) R \rightsquigarrow Q = H_1 \cdot \ldots \cdot H_{n-1}$$

Stellen wir nun dar, wie A überschrieben wird. Es sei

$$\omega^{(j)} = (\underbrace{0, \ldots, 0}_{\text{j-1}}, 1, \omega_{j+1}^{(j)}, \ldots, \omega_m^{(j)})^T$$

der j-te Householder-Vektor, dann erhält man nach Durchführung der QR-Zerlegung

$$A = \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} \\ \omega_2^{(1)} & r_{22} & r_{23} & r_{24} & r_{25} \\ \omega_3^{(1)} & \omega_3^{(2)} & r_{33} & r_{34} & r_{35} \\ \omega_4^{(1)} & \omega_4^{(2)} & \omega_4^{(3)} & r_{44} & r_{45} \\ \omega_5^{(1)} & \omega_5^{(2)} & \omega_5^{(3)} & \omega_5^{(4)} & r_{55} \\ \omega_6^{(1)} & \omega_6^{(2)} & \omega_6^{(3)} & \omega_6^{(4)} & \omega_6^{(5)} \end{pmatrix}$$

Algorithmus 5.5.2: Berechnung Householder-QR:

Gegeben sei $A \in \mathbb{R}^{m\times n}$ mit $m \geq n$. Der folgende Algorithmus bestimmt die Householder-Matrizen $H_1, \ldots, H_n$, so dass mit $Q = H_1 \cdots H_n$ die Matrix $R = Q^T A$ eine obere Dreiecksmatrix ist.
Der obere Dreiecksteil von A wird mit R überschrieben und unterhalb der Diagonalen werden die nichttrivialen Komponenten der Householder-Vektoren gespeichert.

function: $A = housematrix(A)$
for $j = 1 : n$
  $[\omega, \beta] = \texttt{housevector}(A(j : m, j))$
  $A(j : m, j : n) = (I_{m-j+1} - \beta\omega\omega^T)A(j : m, j : n)$
  if $j < m$
    $A(j + 1 : m, j) = \omega(2 : m - j + 1)$
  end
end

Bemerkung 5.5.3 *Analog zur* Givens-*Rotation möchten wir auch an dieser Stelle kurz die geometrische Anschauung der* Householder-*Spiegelung anführen. Für einen gegebenen Vektor $x \in \mathbb{R}^n$ ist der Vektor $y = Px$ die Spiegelung von x an der Ebene $span\{\omega\}^\perp$. Hierzu nachfolgende Abbildung:*

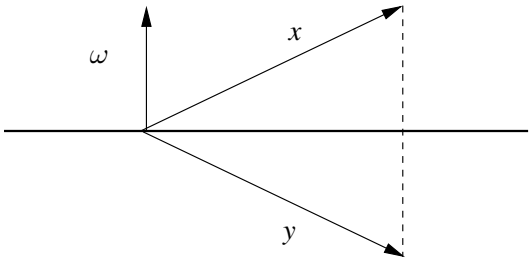


Abb. 5.3: Spiegelung an der zu ω orthogonalen Ebene

Aufgabe 5.5.4 Man zeige, dass die QR-Zerlegung mittels Givens-Rotation $\approx 50\%$ mehr Rechnenoperationen benötigt als mittels Householder-Transformation.

MATLAB-Funktion: Householder.m

```
function [A,R] = Householder(A)
for j = 1:size(A,2)
  [v,beta(j)] = HouseholderVector(A(j:end,j));
  A(j:end,j:end) = A(j:end,j:end)- v * (beta(j) * v'*A(j:end,j:end)
      );
  if j<size(A,1)
    A(j+1:end,j) = v(2:end);
  end
end
if nargout == 2
  R = A;
  A = eye(size(A,1));
  for j = size(R,2):-1:1
    v = [1;R(j+1:end,j)];
    A(j:end,j:end) = A(j:end,j:end) - v*(beta(j)*v'*A(j:end,j:end))
        ;
  end
  R = triu(R);
end
```

MATLAB-Funktion: HouseholderMult.m

```
function y = prod_rx(A,x)
% extract R from A and multiply with x
y = triu(A) * x;

function y = prod_qx(A,x)
y = x;
for j = size(A,2)-1:-1:1
    v = [1;A(j+1:end,j)];
    beta = 2/(v'*v);
    y(j:end) = y(j:end) - v*(beta*v'*y(j:end));
end

function y = prod_qtx(A,x)
y = x;
for j = 1:size(A,2)-1
    v = [1;A(j+1:end,j)];
    beta = 2/(v'*v);
    y(j:end) = y(j:end) - v*(beta*v'*y(j:end));
end
```

Mit der Funktion `Householder` wird zum einen die Matrix A kompakt überschrieben, d.h. der Rückgabewert benötigt nur den Speicherplatz der ursprünglichen Matrix; zum anderen kann man sich auch Q und R mit $A = QR$ ausgeben lassen. Die Berechnung von Rx, Qx und $Q^T x$, wenn die kompakte Form vorliegt, ist in den Routinen `prod_qx`, etc. realisiert.

MATLAB-Beispiel:

```
>> A = rand(3);
>> x = rand(3,1);
>> [Q,R] = Householder(A);
>> A-Q*R
ans =
  1.0e-015 *
   -0.1110         0   -0.1110
   -0.0035   -0.1110         0
   -0.1110         0    0.2220
>> C = Householder(A);
>> Q'*x - prod_qtx(C,x)
ans =
  1.0e-015 *
    0.2220
    0.0555
    0.5551
```

A NORMEN

Normen erfüllen den gleichen Zweck auf Vektorräumen, den Beträge auf der reellen Achse erfüllen. Genauer gesagt, $\mathbb{R}^n$ mit einer Norm auf $\mathbb{R}^n$ liefert einen metrischen Raum. Daraus ergeben sich bekannte Begriffe wie Umgebung, offene Menge, Konvergenz und Stetigkeit für Vektoren und vektorwertige Funktionen.

Definition A.0.1 (Vektornorm) *Eine Vektornorm auf $\mathbb{R}^n$ ist eine Funktion $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ mit den Eigenschaften*

$$\begin{array}{ll} ||x|| \geq 0 & x \in \mathbb{R}^n \qquad (||x|| = 0, \text{ g.d.w. } x = 0), \\ ||x+y|| \leq ||x|| + ||y|| & x, y \in \mathbb{R}^n, \\ ||\alpha x|| = |\alpha|\,||x|| & \alpha \in \mathbb{R}, x \in \mathbb{R}^n. \end{array} \tag{A.1}$$

Eine nützliche Klasse von Vektornormen sind die p-Normen, definiert durch

$$||x||_p = (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{\frac{1}{p}} \qquad p \geq 1. \tag{A.2}$$

Von diesen sind besonders die 1, 2 und ∞ interessant:

$$\begin{aligned} ||x||_1 &= |x_1| + \ldots + |x_n| \\ ||x||_2 &= (|x_1|^2 + \ldots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}} \\ ||x||_\infty &= \max_{1 \leq j \leq n} |x_j| \end{aligned} \tag{A.3}$$

Ein **Einheitsvektor** bzgl. der Norm $||\cdot||$ ist ein Vektor x mit $||x|| = 1$.

A.1 VEKTORNORM-EIGENSCHAFTEN

Ein klassisches Ergebnis bzgl. p-Normen ist die **Hölder-Ungleichung**

$$|x^T y| \leq ||x||_p ||y||_q \qquad \text{mit } \frac{1}{p} + \frac{1}{q} = 1. \tag{A.4}$$

Spezialfall der Hölder-Ungleichung ist die **Cauchy-Schwarz-Ungleichung**

$$|x^T y| \leq ||x||_2 ||y||_2. \tag{A.5}$$

Allen Normen auf $\mathbb{R}^n$ sind äquivalent, d.h. es seien $||\cdot||_\alpha$ und $||\cdot||_\beta$ Normen auf $\mathbb{R}^n$, dann existieren positive Konstanten c_1, c_2, so dass

$$c_1 ||x||_\alpha \leq ||x||_\beta \leq c_2 ||x||_\alpha \qquad (x \in \mathbb{R}^n). \tag{A.6}$$

Zum Beispiel, es sei $x \in \mathbb{R}^n$, dann gilt

$$\begin{array}{ll} ||x||_2 \leq ||x||_1 \leq \sqrt{n}\,||x||_2 \\ ||x||_\infty \leq ||x||_2 \leq \sqrt{n}\,||x||_\infty \\ ||x||_\infty \leq ||x||_1 \leq n\,||x||_\infty. \end{array} \tag{A.7}$$

Bemerkung A.1.1 *Der Beweis der Ungleichungen* (A.4)–(A.7) *sei hier zum Selbststudium überlassen.*

A.2 MATRIXNORMEN

Da $\mathbb{R}^{m\times n}$ isomorph (d.h. es existiert eine bijektive Abb. mit $\varphi(\lambda A+\mu B)=\lambda\varphi(A)+\mu\varphi(B)$) zu $\mathbb{R}^{m\cdot n}$ ist, sollte die Definition einer Matrixnorm äquivalent sein zur Definition einer Vektornorm.

Definition A.2.1 (Matrixnorm) *Es sei $||\cdot||:\mathbb{R}^{m\times n}\to\mathbb{R}$. Dann ist $||\cdot||$ eine Matrixnorm, wenn folgende Eigenschaften gelten:*

$$\begin{array}{lll} ||A||\geq 0 & A\in\mathbb{R}^{m\times n} & (||A||=0,\ \textit{g.d.w. } A=0)\\ ||A+B||\leq||A||+||B|| & A,B\in\mathbb{R}^{m\times n} & \\ ||\alpha A||=|\alpha|\,||A|| & \alpha\in\mathbb{R}, A\in\mathbb{R}^{m\times n}. & \end{array} \tag{A.8}$$

Bemerkung A.2.2 *Die mit am häufigsten verwendeten Normen in der Numerischen Linearen Algebra sind die Frobenius-Norm*

$$||A||_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \tag{A.9}$$

und die p-Normen

$$||A||_p := \sup_{x\neq 0}\frac{||Ax||_p}{||x||_p}\ . \tag{A.10}$$

Man beachte folgende äquivalente Definition

$$||A||_p = \sup_{x\neq 0}\left|\left|A\left(\frac{x}{||x||}\right)\right|\right|_p = \sup_{||x||_p=1}||Ax||_p\ . \tag{A.11}$$

Bemerkung A.2.3 (Submultiplikativität) *Die Frobenius-Norm und die p-Normen erfüllen zusätzlich auch noch die Eigenschaft der* **Submultiplikativität**, *d.h.*

$$||A\cdot B||\leq||A||\ \cdot\ ||B||. \tag{A.12}$$

Bemerkung A.2.4 *Es sei y ein Vektor mit $||y||_p=1$, dann gilt*

$$\begin{aligned} ||A\cdot B||_p &= \max_{x\neq 0}\frac{||ABx||_p}{||x||_p} = \max_{x\neq 0}\frac{||ABx||_p||Bx||_p}{||Bx||_p||x||_p}\\ &\leq \max_{x\neq 0}\frac{||ABx||_p}{||Bx||_p}\cdot\max_{x\neq 0}\frac{||Bx||_p}{||x||_p}\\ &= \max_{y\neq 0}\frac{||Ay||_p}{||y||_p}\cdot\max_{x\neq 0}\frac{||Bx||_p}{||x||_p} = ||A||_p\ ||B||_p\ . \end{aligned}$$

Bemerkung A.2.5

- *Nicht alle Matrix-Normen erfüllen diese Eigenschaft, z.B. gilt für $||A||_A := \max|a_{ij}|$ und $A=B=\begin{pmatrix}1&1\\1&1\end{pmatrix}$ die Ungleichung $||A\cdot B||_A = 2 > ||A||_A\cdot||B||_A$.*
- *Für die p-Normen haben wir die wichtige Eigenschaft, dass für alle $A\in\mathbb{R}^{m\times n}$ und $x\in\mathbb{R}^n$*

$$||Ax||_p\leq||A||_p\cdot||x||_p. \tag{A.13}$$

A.2.1 Einige Eigenschaften der Matrixnorm

Die Frobenius und p-Normen (speziell $p = 1, 2, \infty$) erfüllen gewisse Ungleichungen, welche in der Analysis von Matrixberechungen verwendet werden. Es sei $A \in \mathbb{R}^{m\times n}$, dann gilt

$$||A||_2 \leq ||A||_F \leq \sqrt{n}||A||_2 \tag{A.14}$$

$$\max_{i,j} |a_{ij}| \leq ||A||_2 \leq \sqrt{m \cdot n} \max_{i,j} |a_{ij}| \tag{A.15}$$

$$\frac{1}{\sqrt{n}}||A||_\infty \leq ||A||_2 \leq \sqrt{m}||A||_\infty \tag{A.16}$$

$$\frac{1}{\sqrt{m}}||A||_1 \leq ||A||_2 \leq \sqrt{n}||A||_1 \tag{A.17}$$

Des Weiteren gilt für $A \in \mathbb{R}^{m\times n}$

$$\begin{aligned} ||A||_\infty &= \max_{||x||_\infty=1} ||Ax||_\infty = \max_{||x||_\infty=1} \left\{ \max_{1\leq j\leq m} \left| \sum_{k=1}^{n} a_{jk}x_k \right| \right\} \\ &= \max_{1\leq j\leq m} \left\{ \max_{||x||_\infty=1} \left| \sum_{k=1}^{n} a_{jk}x_k \right| \right\} = \max_{1\leq j\leq m} \sum_{k=1}^{n} |a_{jk}|. \end{aligned}$$

Aufgrund dieser Gleichung bezeichnet man $||\cdot||_\infty$ auch als **Zeilensummennorm**. Diese ist mittels der letzten Gleichung auch leicht zu bestimmen, d.h.

$$||A||_\infty = \max_{1\leq j\leq m} \sum_{k=1}^{n} |a_{jk}| \tag{A.18}$$

Analoges erhält man für die 1-Norm:

$$\begin{aligned} ||A||_1 &= \max_{||x||_1=1} ||Ax||_1 = \max_{||x||_1=1} \sum_{j=1}^{m} \left| \sum_{k=1}^{n} a_{jk}x_k \right| = \max_{||x||_1=1} \sum_{j=1}^{m} \sum_{k=1}^{n} |a_{jk}|\,|x_k| \text{sign}(a_{jk}x_k) \\ &= \max_{||x||_1=1} \sum_{k=1}^{n} |x_k| \sum_{j=1}^{m} |a_{jk}| \text{sign}(a_{jk}x_k) = \max_{||x||_1=1} \sum_{k=1}^{n} |x_k| \sum_{j=1}^{m} |a_{jk}| = \max_{1\leq k\leq m} \sum_{j=1}^{n} |a_{jk}|. \end{aligned}$$

Also gilt:

$$||A||_1 = \max_{1\leq k\leq n} \sum_{j=1}^{m} |a_{jk}| \tag{A.19}$$

Diese Norm bezeichnet man auch als **Spaltensummennorm**.

Bemerkung A.2.6 *Einfache Eselbrücke:* $\boxed{1}$*-Spaltensummen,* $\boxed{\infty}$*-Zeilensummen.*

Auch für die 2-Norm läßt sich eine äquivalente Formulierung finden.

Satz A.2.7 *Es sei $A \in \mathbb{R}^{m\times n}$. Dann existiert ein Vektor $z \in \mathbb{R}^n$ mit $||z||_2 = 1$, so dass $A^TAz = \mu^2 z$ mit $\mu = ||A||_2$.*

Bemerkung A.2.8 *Der Satz impliziert, daß $||A||_2^2$ eine Nullstelle des Polynoms $p(z) = det(A^TA - \lambda I)$ ist. Genauer betrachtet, ist die 2-Norm von A die Wurzel des größten Eigenwerts von A^TA.*

Beweis. Es sei $z \in \mathbb{R}^n$ mit $||z||_2 = 1$ und $||Az||_2 = ||A||_2$. Da z die Funktion

$$g(x) = \frac{1}{2}\frac{||Ax||_2^2}{||x||_2^2} = \frac{1}{2}\frac{x^T A^T A x}{x^T x} \tag{A.20}$$

maximiert, folgt daraus, daß sie $\nabla g(x) = 0$ erfüllt, wobei ∇g der Gradient von g ist $\left(\nabla := \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n}\right)\right)^T$. Eine einfache Ableitung zeigt für $i = 1, \ldots, n$

$$\frac{\partial g(z)}{\partial z_i} = \left[z^T z \sum_{j=1}^{n} (A^T A)_{ij} z_j - (z^T A^T A z) z_i \right] / (z^T z)^2. \tag{A.21}$$

In Vektornotation bedeutet dies $A^T A z = (z^T A^T A z) z$. Setzt man nun $\mu = ||A||_2$ so ergibt sich daraus die Behauptung. □

Lemma A.2.9 *Es sei $A \in \mathbb{R}^{m \times n}$. Dann gilt*

$$||A||_2 \leq \sqrt{||A||_1 \; ||A||_\infty}. \tag{A.22}$$

Beweis. Es sei $z \in \mathbb{R}^n$ mit $A^T A z = \mu^2 z$ und $\mu = ||A||_2$, d.h. es sei $z \neq 0$ ein Vektor, für den das Maximum von $\frac{||Ax||_2}{||x||_2}$ angenommen wird. Dann gilt

$$\mu^2 ||z||_1 = ||A^T A z||_1 \leq ||A^T||_1 ||Az||_1 \leq ||A||_\infty ||A||_1 ||z||_1. \tag{A.23}$$

Kondensation von $||z||_1$ und Wurzelziehen liefert das gewünschte Ergebnis. □

Definition A.2.10 (verträgliche Vektornorm) *Eine Matrixnorm $||A||$ heißt kompatibel oder verträglich mit der Vektornorm $||x||$, falls die Ungleichung gilt*

$$||Ax|| \leq ||A|| \; ||x|| \qquad (x \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}) \tag{A.24}$$

Kombinationen von verträglichen Normen sind etwa

$||A||_G, ||A||_\infty$ sind verträglich mit $||x||_\infty$;

$||A||_G, ||A||_1$ sind verträglich mit $||x||_1$;

$||A||_G, ||A||_F, ||A||_2$ sind verträglich mit $||x||_2$, mit

$$||A||_G := n \max_{i,k} |a_{ik}| \qquad (A \in \mathbb{R}^{m \times n})$$

Bemerkung A.2.11 *Man verifiziere selbständig an einigen Beispielen die Verträglichkeit von o.g. Normenpaaren.*

Abschließend erhalten wir am Ende des Kapitels.

Satz A.2.12 *Die Matrix-p-Normen sind unter allen mit der Vektornorm $||x||_p$ verträglichen Matrixnormen die kleinsten.*

B EINFÜHRUNG IN MATLAB

B.1 GRUNDLEGENDE MATLAB-BEFEHLE

Ruft man das Programm MATLAB mit dem Befehl `matlab` auf, so erscheinen auf dem Monitor einige Fenster. Auf den Linux-Rechnern des KIZ müssen Sie vorher die notwendigen Pfade ergänzen. Geben Sie dazu in einem Konsole-Fenster den Befehl option matlab ein. Von diesen ist das Befehl-Fenster der primäre Ort um Befehle einzugeben und Fehler oder Ergebnisse abzulesen! Das Prompt-Zeichen $\gg$ ist im Befehl-Fenster dargestellt und dort findet man üblicherweise einen blinkenden Cursor. Der blinkende Cursor und der MATLAB-Prompt zeigen einem, daß MATLAB eine Eingabe erwartet.

B.1.1 Einfache mathematische Operationen

Genauso wie mit einem simplen Taschenrechner kann man auch mit MATLAB einfache mathematische Operationen ausführen, z.B. ergibt die Eingabe

```
≫ 3 + 4
```

die Ausgabe

```
ans =
     7
```

Man beachte, daß MATLAB im allgemeinen keine Zwischenräume benötigt, um Befehle eindeutig zu verstehen. Alternativ zu dem obigen Beispiel können in MATLAB auch Variablen verwendet werden.

```
≫ a = 3
a =
     3
≫ b = 4
b =
     4
≫ c = a + b
c =
     7
```

MATLAB besitzt folgende einfache arithmetische Operationen

Operation	**Symbol**	**Beispiel**
Addition, $a + b$	+	5 + 3
Subtraktion, $a - b$	−	23−12
Multiplikation, $a \cdot b$	*	13.3 * 63.13
Division, $a \div b$	/ or \	$17/4 = 4\backslash 17$
Potenz, a^b	^	3^4

Die Reihenfolge, in der eine Folge von Operationen abgearbeitet wird, läßt sich wie folgt beschreiben. Ausdrücke werden von links nach rechts ausgeführt, wobei die Potenzierung die höchste Priorität besitzt gefolgt von Punktoperation, sprich Multiplikation und Division. Die geringste Priorität haben Addition und Subtraktion. Mit Hilfe von Klammern kann diese Vorgehensweise geändert werden, wobei innere Klammern vor äußeren Klammern berechnet werden.

B.1.2 Variablen

Wie in anderen Programmiersprachen hat auch MATLAB Regeln für Variablennamen. Eine Variable repräsentiert ein Datenelement, dessen Wert während der Programmausführung – gegebenenfalls mehrfach – geändert werden kann. Variablen werden anhand ihrer „Namen“ identifiziert. Namen bestehen aus ein bis neunzehn Buchstaben, Ziffern oder Unterstrichen, wobei das erste Zeichen ein Buchstabe sein muß. Man beachte, d aß MATLAB Groß- und Kleinschreibung unterscheidet.(Windows is im Gegensatz zu Linux nicht so restriktiv, da Sie jedoch Programme austauschen wollen, sollten Windows-Benutzer besondere Aufmerksamkeit walten lassen)

Einer Variablen ist Speicherplatz zugeordnet. Wenn man eine Variable verwendet, dann meint man damit entweder den zugeordneten Speicherplatz oder den Wert, der dort augenblicklich abgespeichert ist. Einen Überblick über alle Variablen erhält man mit dem Befehl `who` oder `whos`, wobei letzterer die Angabe des benutzten Speicherplatzes beinhaltet.
Zusätzlich zu selbstdefinierten Variablen gibt es in MATLAB verschiedene spezielle Variablen. Diese lauten

spezielle Variablen	**Wert**
ans	standard Variablenname benutzt für Ergebnisse
pi	$3.1415\ldots$
eps	Maschinengenauigkeit
flops	Zähler für die Anzahl der Fließkommaoperationen
inf	steht für Unendlich (eng. infinity). z.B. $1/0$
NaN	eng. Not a Number, z.B. $0/0$
i (und) j	$i = j = \sqrt{-1}$

In MATLAB kann der Speicherplatz, der durch Variablen belegt ist, durch den Befehl `clear` wieder freigegeben werden, z.B.

```
≫ clear a b c
```

B.1.3 Kommentare und Punktion

Der Text, der nach einem Prozentzeichen `%` folgt, wird in MATLAB als Kommentar verstanden

```
≫ dummy = 4 % Wert von dummy
dummy =
     4
```

Mehrere Befehle können in eine Zeile geschrieben werden, wenn sie durch Kommata oder Semikola getrennt werden. Kommata veranlassen MATLAB, die Ergebnisse anzuzeigen. Bei einem Semikolon wird die Ausgabe unterdrückt. Durch eine Sequenz von drei Punkten kann man einen Befehl in der folgenden Zeile fortsetzen.

B.1.4 Spezielle Funktionen

Eine unvollständige Liste von Funktionen, die MATLAB bereitstellt, ist im folgenden dargestellt. Die meisten Funktionen sind so definiert, wie man sie üblicherweise benutzt.

```
≫ y = cos(pi)
y =
     -1
```

MATLAB bezieht sich im Zusammenhang mit Winkelfunktionen auf das Bogenmaß.

Funktion	Bedeutung
abs(x)	Absolutbetrag
cos(x)	Kosinus
exp(x)	Exponentialfunktion: e^x
fix(x)	rundet auf die nächste, vom Betrag her kleinere ganze Zahl
floor(x)	rundet auf die nächste, kleinere ganze Zahl
gcd(x,y)	größter gemeinsamer Teiler von x und y
lcm(x,y)	kleinstes gemeinsames Vielfaches von x und y
log(x)	natürlicher Logarithmus
rem(x,y)	Modulo (eng. remainder of division), z.B. rem(5,2)=1
sign(x)	Signum Funktion, z.B. sign(2.3) = 1, sign(0) = 0, sign(-.3) = -1
sin(x)	Sinus
sqrt(x)	Quadratwurzel
tan(x)	Tangens

B.1.5 Skript-Dateien

Für einfache Probleme ist es schnell und effizient, die Befehle am MATLAB-Prompt einzugeben. Für größere und umfangreichere Aufgabenstellungen bietet MATLAB die Möglichkeit, sogenannte Skript-Dateien zu verwenden, in denen die Befehle in Form einer Textdatei aufgeschrieben sind und die man am Prompt übergibt. MATLAB öffnet dann diese Dateien und führt die Befehle so aus, als hätte man sie am Prompt eingegeben. Die Datei nennt man Skript-Datei oder M-Datei, wobei der Ausdruck M-Datei daher rührt, daß diese Dateien das Suffix `.m` haben, z.B. `newton.m`. Um eine M-Datei zu erstellen, ruft man einen Editor auf und speichert die Datei in dem Verzeichnis, von dem aus man MATLAB gestartet hat oder starten wird. Die Datei, z.B. `newton.m`, wird in MATLAB dann durch Eingabe von `newton` am Prompt aufgerufen.

Für die Benutzung von Skript-Dateien hat MATLAB unter anderem folgende hilfreichen Befehle

M-Datei-Funktionen	
disp(ans)	zeigt den Wert der Variablen `ans`, ohne ihren Namen auszugeben
input	erwartet vom Benutzer eine Eingabe
keyboard	übergibt zeitweise die Kontrolle an die Tastatur
pause	hält das Programm an, bis eine Taste betätigt wird

Die folgende Skript-Datei `beispiel1.m`

```
% beispiel1.m
% Beispiel fuer eine Skript-Datei
tmp = input(' Geben Sie bitte eine Zahl an >' );
3 * tmp;
```

führt zu der Ausgabe

```
≫ beispiel1
Geben Sie bitte eine Zahl an > 6
ans =
     18
```

B.1.6 Dateiverwaltung

MATLAB unterstützt eine Vielzahl von Dateiverwaltungsbefehlen, welche es einem ermöglichen Dateien zu listen, Skript-Dateien anzusehen oder zu löschen und Verzeichnisse zu wechseln.

Datei-Managment-Funktionen

cd path	wechselt in das Verzeichnis `path`
delete beispiel	löscht die Datei `beispiel.m`
ls	zeigt alle Dateien im aktuellen Verzeichnis an
pwd	zeigt den aktuellen Verzeichnispfad an
type beispiel	zeigt den Inhalt der Datei `beispiel.m` im Befehl-Fenster
what	zeigt alle M-Dateien und MAT-Dateien im aktuellen Verzeichnis an

B.1.7 Hilfe

Online-Hilfe: Da sich nicht jeder Benutzer alle MATLAB-Befehle merken kann oder auch von einigen auch nur die Syntax unklar ist, bietet MATLAB die Möglichkeit der Online-Hilfe. Dabei gibt es prinzipiell mehrere Möglichkeiten. Ist einem ein Befehl bekannt und man sucht Informationen über die Syntax, so gibt es den Befehl `help`.

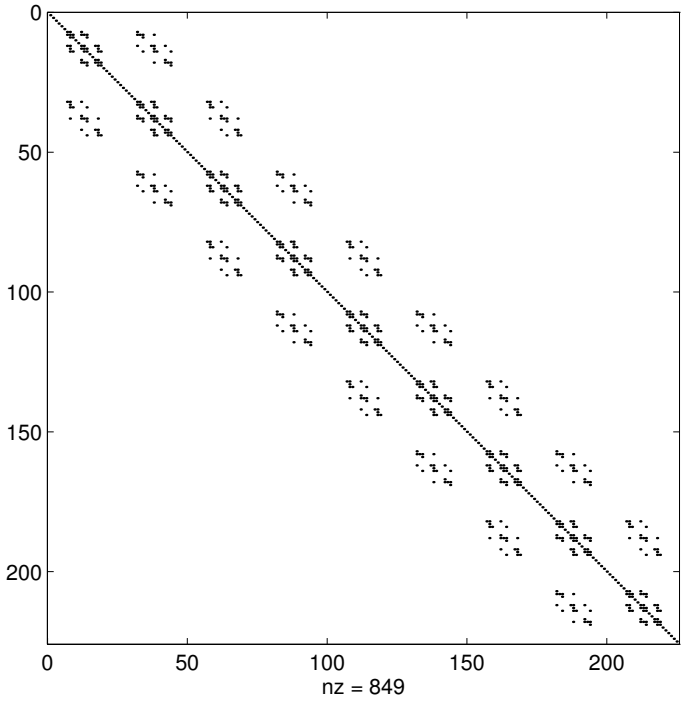
```
≫ help sqrt
  SQRT   Square root.
         SQRT(X) is the square root of the elements of X. Complex
         results are produced if X is not positive.

         See also SQRTM.
```

Als Beispiel haben wir uns hier die Hilfe zu dem Befehl `sqrt` ausgeben lassen.

Die andere Möglichkeit der von MATLAB gelieferten Hilfe ist durch den Befehl `lookfor` gegeben. Hier durchsucht das Programm alle ersten Zeilen der MATLAB Hilfe-Kennwörter und Skript-Dateien die im MATLAB Suchpfad zu finden sind. Das Bemerkenswerte dabei ist, daß dieser Begriff kein Befehl zu sein braucht.

```
≫ lookfor cholesky

 CHOL    Cholesky factorization

≫  CHOL   Cholesky factorization.

 CHOL(X) uses only the diagonal and upper triangle of X.
 The lower triangular is assumed to be the (complex conjugate)
 transpose of the upper.  If X is positive definite, then
 R = CHOL(X) produces an upper triangular R so that R'*R =X.
 If X is not positive definite, an error message is printed.

 With two output arguments, [R,p] = CHOL(X) never produces an
 error message.  If X is positive definite, then p is 0 and R
 is the same as above.   But if X is not positive definite, then
 p is a positive integer and R is an upper triangular matrix of
 order q = p-1 so that R'*R = X(1:q,1:q).

≫ lookfor  factorization

 CHOL    Cholesky factorization.
 QRDELETE Delete a column from the QR factorization.
 QRINSERT Insert a column in the QR factorization.
 SYMBFACT Symbolic factorization analysis.
```

Eine weitere Möglichkeit, sich Hilfe zu verschaffen, besteht darin, das Helpdesk aufzurufen. Wenn Sie helpdesk am Prompt eingeben, öffnet sich die folgende Hilfsumgebung



B.2 MATHEMATIK MIT MATRIZEN

B.2.1 Matrixkonstruktion und Adressierung

einfache Matrix Konstruktionen	
x=[1 4 2*pi 4]	erstelle einen Zeilenvektor x mit genannten Einträgen
x=$anfang$:$ende$	erstelle einen Zeilenvektor x beginnend mit $anfang$, Inkrement 1 und endend mit $ende$
x=$anfang$:$inkrement$:$ende$	ähnliches wie oben mit dem Inkrement $inkrement$
x=linspace($anfang$,$ende$,n)	erzeugt einen Zeilenvektor der Dimension n mit $x(i) = \frac{(n-i)\cdot anfang+(i-1)\cdot ende}{n-1}$

Im folgenden sind einige charakteristische Beispiele aufgeführt.

```
≫ B = [1 2 3 4; 5 6 7 8]
B =
     1 2 3 4
     5 6 7 8
```

Der Operator ′ liefert für reelle Matrizen die Transponierte.

```
≫ C = B'
C =
     1 5
     2 6
     3 7
     4 8
```

Der Doppelpunkt `:` in der zweiten Komponente spricht alle vorhandenen Spalten an, d.h. er ist ein zu `1:4` äquivalenter Ausdruck.

```
≫ C = B(1,:)
C =
     1 2 3 4
≫C = B(:,3)'
C =
     3 7
```

Es lassen sich auch einzelne Komponenten neu definieren.

```
≫A = [1 2 3; 4 5 6; 7 8 9]
A =
     1 2 3
     4 5 6
     7 8 9
≫ A(1,3) = 9
A =
     1 2 9
     4 5 6
     7 8 9
```

Ist ein Eintrag noch nicht definiert, so verwendet MATLAB die minimale Erweiterung dieser Matrix und setzt undefinierte Einträge zu Null.

```
≫ A(2,5) = 4
A =
     1 2 9 0 0
     4 5 6 0 4
     7 8 9 0 0
```

Im folgenden werden die Vektoren `(3,2,1)` und `(2,1,3,1,5,2,4)` dazu verwendet, die Matrix `C` zu indizieren, d.h. `C` hat die Struktur

```
A(3,2) A(3,1) A(3,3) A(3,1) A(3,5) A(3,2) A(3,4)
A(2,2) A(2,1) A(2,3) A(2,1) A(2,5) A(2,2) A(2,4)
A(1,2) A(1,1) A(1,3) A(1,1) A(1,5) A(1,2) A(1,4)
```

In MATLAB erhält man nun

```
≫ C=A(3:-1:1,[2 1 3 1 5 2 4])
C =
     8 7 9 7 0 8 0
     5 4 6 4 4 5 0
     2 1 9 1 0 2 0
```

Ein weiteres Beispiel für Indizierung ist

```
≫ C=C(1:2,2:3)
C =
     7 9
     4 6
```

Im nächsten Beispiel wird ein Spaltenvektor dadurch konstruiert, daß alle Elemente aus der Matrix `C` hintereinander gehängt werden. Dabei wird spaltenweise vorgegangen.

```
≫ b=C(:)'
b =
     7 4 9 6
```

Das Löschen einer ganzen Zeile oder Spalte kann durch das Umdefinieren in eine 0×0-Matrix geschehen, z.B.

```
≫ C(2,:)=[ ]
C =
     7 9
```

B.2.2 Skalar-Matrix-Operationen

In MATLAB sind Skalar-Matrix-Operationen in dem Sinne definiert, daß Addition, Subtraktion, Division und Multiplikation mit einem Skalar elementweise durchgeführt werden. Es folgen zwei erklärende Beispiele.

```
≫ B - 1
ans =
     0 1 2 3
     4 5 6 7
≫ 9 + 3 * B
ans =
     12 15 18 21
     24 27 30 33
```

B.2.3 Matrix-Matrix-Operationen

Die Operationen zwischen Matrizen sind nicht so kanonisch zu definieren wie die z wischen Skalar und Matrix, insbesondere sind Operationen zwischen Matrizen unterschiedlicher Dimension schwer zu definieren. Desweiteren sind die Operationen $*$ und $.*$, bzw. $/$ und $./$ sowie $\backslash$ und $.\backslash$ zu unterscheiden. In nachfolgender Tabelle sind die Matrixoperationen beschrieben.

	komponentenweise Matrixoperationen
Beispieldaten	$a = [a_1, a_2, \ldots, a_n]$, $b = [b_1, b_2, \ldots, b_n]$, c ein Skalar
komp. Addition	$a + c = [a_1 + c\ a_2 + c \ldots a_n = c]$
komp. Multiplikation	$a * c = [a_1 \cdot c\ a_2 \cdot c\ \ldots\ a_n \cdot c]$
Matrix–Addition	$a + b = [a_1 + b_1\ a_2 + b_2 \ldots a_n + b_n]$
komp. Matrix-Multiplikationen	$a. * b = [a_1 \cdot b_1\ a_2 \cdot b_2 \ldots a_n \cdot b_n]$
komp. Matrix-Div. von rechts	$a./b = [a_1/b_1\ a_2/b_2 \ldots a_n/b_n]$
komp. Matrix-Div. von links	$a.\backslash b = [b_1/a_1\ b_2/a_2 \ldots b_n/a_n]$
komp. Matrix-Potenz	$a.\hat{}c = [a_1^c\ a_2^c \ldots a_n^c]$
	$c.\hat{}a = [c^{a_1}\ c^{a_2} \ldots c^{a_n}]$
	$a.\hat{}b = [a_1^{b_1}\ a_2^{b_2} \ldots a_n^{b_n}]$

Es folgen nun einige Beispiele zu Matrixoperationen

```
≫ g=[1 2 3; 4 5 6]; % zwei neue Matrizen
≫ h=[2 2 2; 3 3 3];
≫ g+h   % addiere g und h komponentenweise
ans =
     3  4  5
     7  8  9
≫ ans-g   % subtrahiere g von der vorherigen Antwort
ans =
     2  2  2
     3  3  3
≫ h.*g   % multipliziere g mit h komponentenweise
ans =
      2   4   6
     12  15  18
≫ g*h'   % multipliziere g mit h'
ans =
     12  18
     30  45
```

B.2.4 Matrix-Operationen und -Funktionen

	Matrixfunktionen
reshape(A,m,n)	erzeugt aus den Eintägen der Matrix A eine $m \times n$-Matrix, wobei die Einträge spaltenweise aus A gelesen werden.
diag(A)	ergibt die Diagonale von A als Spaltenvektor
diag(v)	erzeugt eine Diagonalmatrix mit dem Vektor v in der Diagonalen
tril(A)	extrahiert den unteren Dreiecksanteil der Matrix A
triu(A)	extrahiert den oberen Dreiecksanteil der Matrix A

Es folgen einige Beispiele

```
≫ g=linspace(1,9,9)    % ein neuer Zeilenvektor
g =
      1    2    3    4    5    6    7    8    9
≫ B=reshape(g,3,3)    % macht aus g eine 3 x 3 Matrix
B =
      1    4    7
      2    5    8
      3    6    9
≫ tril(B)
ans =
      1    0    0
      2    5    0
      3    6    9
```

Funktion	Bedeutung
R=chol(A)	Choleskyzerlegung
cond(A)	Konditionszahl der Matrix A
d=eig(A)	Eigenwerte und -vektoren
[V,d]=eig(A)	
det(A)	Determinante
hess(A)	Hessenbergform
inv(A)	Inverse
[L,U]=lu(A)	Zerlegung gegeben durch Gauss-Algorithmus
norm(A)	euklidische-Norm
rank(A)	Rang der Matrix A

Bemerkung : D er \ Operator ist auch für Matrizen definiert und liefert in Kombination mit Vektoren für reguläre Matrizen ihre Inverse, d.h. `A^(-1)x=A\x` .

B.2.5 Spezielle Matrizen

spezielle Matrizen	
eye(n)	erzeugt eine Einheitsmatrix der Dimension n
ones(m,n)	erzeugt eine $m \times n$-Matrix mit den Einträgen 1
zeros(m,n)	erzeugt eine $m \times n$-Matrix mit den Einträgen 0

B.2.6 Spezielle Funktionen für schwachbesetzte Matrizen

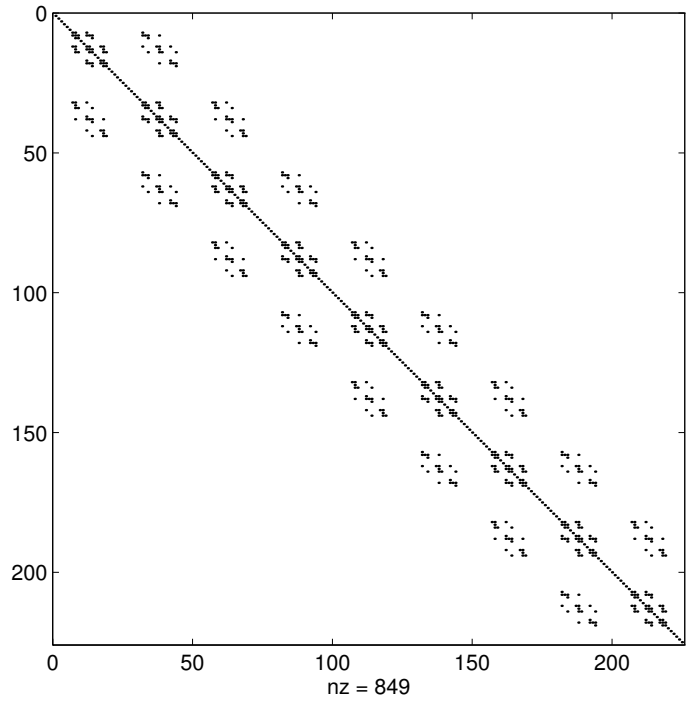
Bei vielen numerischen Anwendungen treten schwachbesetzte Matrizen auf. M ATLAB hat für solche Matrizen besondere Sparse-Funktionen, die dieser Eigenschaft Rechnung tragen.

Funktion	Bedeutung
find(A)	findet Indizes von Nichtnulleinträgen
nnz(A)	Anzahl an Nichtnulleinträgen
spdiags(v)	erzeugt eine Sparse-Diagonalmatrix mit dem Vektor v als Diagonale
speye(n)	erzeugt eine Sparse-Einheitsmatrix
spy(A)	visualisiert die Struktur der Matrix A

Kurze Illustration der Funktionsweise obiger Befehle anhand einiger Beispiele.

```
≫ E=eye(100);    % vollbesetzte 100 x 100 Einheitsmatrix
≫ Es=sparse(E);  % Sparse-Version von E
≫ whos
     Name      Size    Elements  Bytes  Density  Comple
        E  100 by 100    10000  80000     Full     No
       Es  100 by 100      100   1600   0.0100     No
 Grand total is 10100 elements using 81600 bytes
≫ A=spdiags([7*ones(4,1),ones(4,1),2*ones(4,1)],[-1,0,1],4,4);
≫ nnz(A)
ans =
     10
≫ full(A)
ans =
      1   2   0   0
      7   1   2   0
      0   7   1   2
      0   0   7   1
```

Als Beispiel für `spy` sei hier die Besetzungstruktur einer 3D-FEM Steifigskeitsmatrix gezeigt.



B.3 DATENVERWALTUNG

Für die meisten Anwendungen genügt es, Datenfelder in einem Format abzuspeichern und wieder laden zu können. Die Befehle `load` und `save` setzen voraus, daß die Daten in einem System unabhängigen, binären Format in einer Datei mit dem Suffix `.mat` gespeichert sind oder in einem einfachen ASCII-Format vorliegen.

B.3.1 Daten speichern

Im folgenden wird eine 3×5 -Matrix im binär-Format in der Datei `A.mat` gespeichert. Diese Daten sind sehr kompakt gespeichert.

```
≫ A=zeros(3,5);
≫ save A
```

Gibt man sich aber den Inhalt dieser Datei auf dem Bilschirm aus, so gibt er wenig Sinn. Möchte man sich also z.B. einen Lösungsvektor sichern um ihn später „per Hand zu analysieren“ so speichere man die Daten als ASCII-Datei, dabei kann man wählen zwischen einer 8-stelligen oder 16-stelligen Abspeicherung.

```
≫ save mat1.dat A -ascii          %  8-stellige Speicherung
≫ save mat2.dat A -ascii -double  % 16-stellige Speicherung
```

Hier wurde die Matrix mit 8-stelligem Format in der Datei `mat1.dat` gespeichert, bzw. 16-stellig in der Datei `mat2.dat`.

B.3.2 Daten laden

Mit dem Befehl `load A` versucht MATLAB, die in `A.mat` gespeicherten Daten in einem Datenfeld `A` zu speichern. Auch ASCII-Dateien kann MATLAB lesen. Da es hier jedoch keine Standardendung gibt, ist die Datei inklusive Endung anzugeben. Es ist darauf zu achten, daß ein rechteckiges Feld an Daten vorliegt, d.h. daß m Zeilen mit jeweils n numerischen Werten vorliegen. MATLAB erstellt dann eine $m \times n$-Matrix mit dem Namen der Datei ohne Suffix.

```
≫ load  mat1.dat
≫ whos
    Name         Size    Elements  Bytes  Density  Complex
    mat1        3 by 5          15    120     Full    No

 Grand total is 15 elements using 120 bytes
```

B.4 AUSGABE VON TEXT

Mit dem Befehl `fprintf` lassen sich Strings, d.h. Zeichenfolgen, auf dem Bildschirm ausgeben.

```
≫ fprintf('\n Hello world %12.3e\n',4);

 Hello world    4.000e+00
```

Man sieht, daß der auszugebende Text zusätzliche Zeichen enthält, die nicht mit ausgedruckt werden. Diese Zeichen nennt man Escape-Sequenzen. In obigem Beispiel ist die Sequenz `\n` eingebaut. `\n` steht für „newline“ und sorgt dafür, daß bei der Textausgabe an diesen Stellen eine neue Zeile begonnen wird. Der Ausdruck `%12.3e` dient als Platzhalter für einen reellen Wert, der durch Komma getrennt hinter der Zeichenkette folgt. Dabei sei die Zahl in Exponentialdarstellung auszugeben, wofür 12 Stellen mit 3 Nachkommastellen bereitgestellt. Im Beispiel ist dies der Wert 4. Anstatt eines expliziten Wertes können auch Variablen oder Ausdrücke, z.B. `3 * 4` stehen. Ein Platzhalter kann mehrfach in einem `printf`-Befehl vorkommen. In diesem Fall müssen hinter der Zeichenkette genau so viele Werte folgen, wie Platzhalter angegeben sind. Die Reihenfolge der Werte muß mit der Reihenfolge der Platzhalter übereinstimmen, da die Ausdrücke von links nach rechts bewertet werden. Die Escape-Sequenzen dürfen im Text an beliebiger Stelle stehen.

Ausgabeformate

Befehl	Ausgabe
fprintf('%.0e\n',1.234567)	1e00+
fprintf('%.2e\n',1.234567)	1.23e00+
fprintf('%.5e\n',1.234567)	1.23456e00+
fprintf('%10.0e\n',1.234567)	1e00+
fprintf('%10.2e\n',1.234567)	1.23e00+
fprintf('%10.5e\n',1.234567)	1.2346e00+
fprintf('%10.2f\n',1.234567)	1.23
fprintf('%10.5f\n',1.234567)	1.23457

MATLAB rundet numerische Werte bei der Ausgabe, wenn nötig!

B.5 KONTROLLBEFEHLE

B.5.1 For-Schleifen

1. Mit jeglicher gültigen Matrix-Darstellung läßt sich eine FOR-Schleife definieren, z.B.

```
≫ data = [1 7 3 2; 5 4 7 2]
data =
     1 7 3 2
     5 4 7 2
≫ for n=data
   x=n(1)-n(2)
  end
x =
     -4
x =
      3
x =
     -4
x =
     0
```

2. FOR-Schleifen können nach Belieben geschachtelt werden.

```
≫ for k=3:5
   for l=4:-1:2
    A(k,l)=k^2-l;
   end
  end
≫ A
A =
      0  0  0  0
      0  0  0  0
      0  7  6  5
      0 14 13 12
      0 23 22 21
```

3. FOR-Schleifen sollten vermieden werden, w ann immer sie durch eine äquivalente Matrix-Darstellung ersetzt werden können. Der folgende Ausdruck ist unten optimierter Version aufgeführt.

```
≫ for n=1:10
   x(n)=sin(n*pi/10);
  end
≫x
x =
     Columns 1 through 7
       0.3090  0.5878  0.8090  0.9511  1.0000  0.9511  0.8090
     Columns 8 through 10
       0.5878  0.3090  0.0000

≫ n=1:10;
≫ x=sin(n*pi/10);
≫ x
x =
     Columns 1 through 7
       0.3090  0.5878  0.8090  0.9511  1.0000  0.9511  0.8090
     Columns 8 through 10
       0.5878  0.3090  0.0000
```

4. Um die Ausführungsgeschwindigkeit zu maximieren, sollte b enötigter Speicherplatz vor Ausführung der FOR-Schleife alloziert werden.

```
≫ x=zeros(1,10);
≫ x(1)=0.5;
≫ for n=2:10
   x(n)=x(n-1)*sin(n*pi/10);
  end
≫ x
x =
     Columns 1 through 7
       0.5000  0.2939  0.2378  0.2261  0.2261  0.2151  0.1740
     Columns 8 through 10
       0.1023  0.0316  0.0000
```

B.5.2 WHILE-Schleifen

WHILE-Schleifen sind wie folgt aufgebaut.

```
while Aussage
     Anweisungen
end
```

Die Anweisungen zwischen `while` **und** `end` **werden so lange ausgeführt, wie** `Aussage` **wahr ist**, z.B.

```
≫ num=0; EPS=1;
≫ while (1+EPS) > 1
   EPS=EPS/2;
   um=num+1;
  end
≫ num
num =
     53
```

B.5.3 IF-ELSE-END Konstrukte

Eine IF-ELSE-END-Schleife enthält nach dem IF eine Aussage, die daraufhin überprüft wird, ob sie wahr oder falsch ist. Ist sie wahr, so werden die in den folgenden Zeilen stehenden Anweisungen ausgeführt und die Schleife beendet. Ist sie falsch, so erfolgen die Anweisungen, die dem ELSE folgen (ELSE ist optional). Solch ein Konstrukt kann erweitert werden um beliebig viele ELSIF Befehle, die dieselbe Funktion haben wie der am Anfang stehende IF Befehl, aber nur beachtet werden, falls alle vorher überprüften Aussagen falsch sind.

```
if Aussage1
     Anweisungen, wenn Aussage1 wahr
elseif Aussage2
     Anweisungen, wenn Aussage1 falsch und Aussage2 wahr
elseif
     Anweisungen, wenn Aussage1 und Aussage2 falsch
end
```

B.5.4 Relationen und logische Operatoren

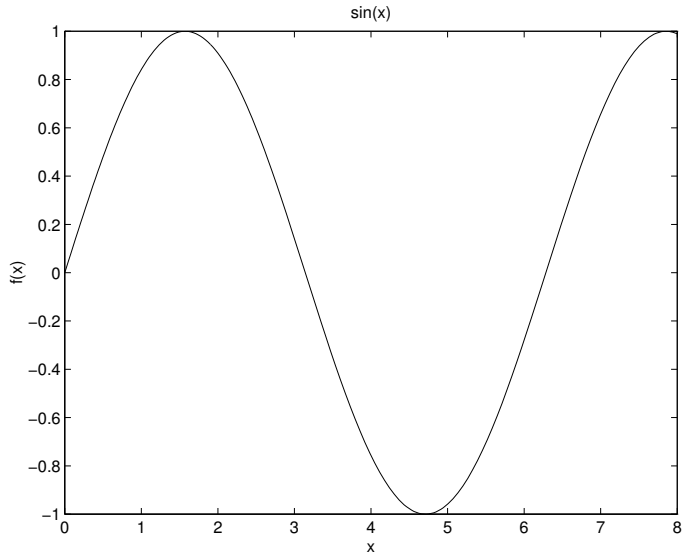
Relationen	
<	kleiner als
<=	kleiner als oder gleich
>	größer als
>=	größer als oder gleich
==	gleich
~=	ungleich

logische Operatoren	
&	UND
|	ODER
~	NICHT

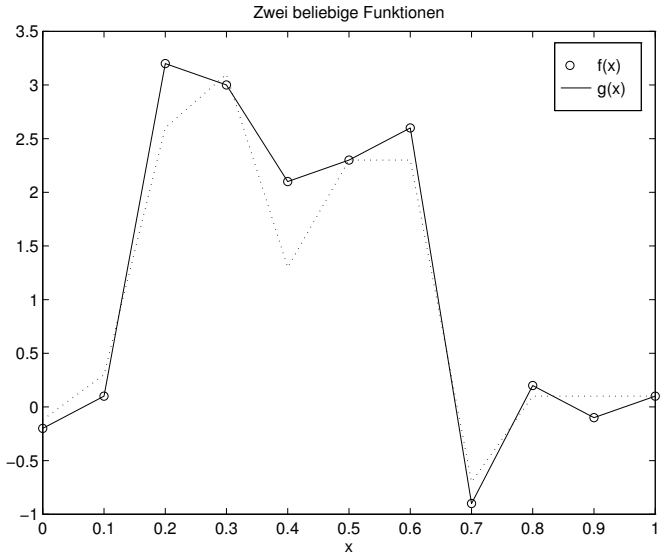
B.6 Graphische Darstellung

B.6.1 Zweidimensionale Graphiken

```
≫ f='sin(x)';
≫ fplot(f,[0 8]);
≫ title(f),xlabel('x'),ylabel('f(x)');
```




```
≫ x=0:0.1:1;
≫ y=[-0.2 0.1  3.2  3  2.1  2.3  2.6 -0.9 0.2 -.1 .1];
≫ z=[-0.12 0.3  2.6  3.1  1.3  2.3  2.3 -0.7 0.1 .1 .1];
≫ plot(x,y,'o',x,y,x,z,':');
≫ title('Zwei beliebige Funktionen'),xlabel('x');
≫ legend('f(x)','g(x)')
```



Linientypen und Farben

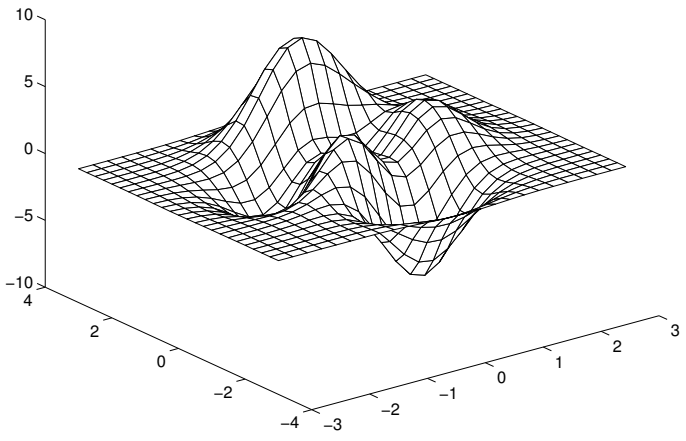
Symbol	Farbe	Symbol	Linientyp
y	gelb	$\cdot$	Punkt
m	magenta	$\circ$	Kreis
c	cyan	x	x-Markierung
r	rot	$+$	$+$-Markierung
g	grün	$*$	Sternchen
b	blau	$-$	durchgezogene Linie
w	weiß	:	gepunktete Linie
k	schwarz	$-.$	Strichpunkt-Linie
		$--$	gestrichelte Linie

2-D Graphikanweisung

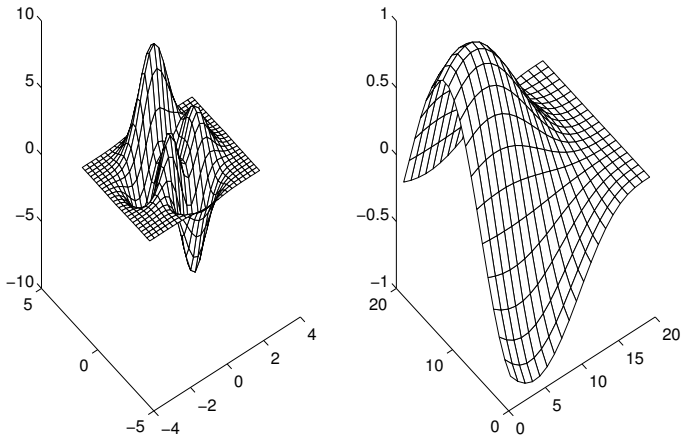
axis	modifiziert die Axen-Proportionen
clf	löscht die Graphik im Graphik-Fenster
close	schließt das Graphik-Fenster
grid	erzeugt ein achsenparalleles Gitter
hold	ermöglicht das Überlagern von Graphiken
subplot	erstellt mehrere Teilgraphiken in einem Fenster
text	gibt Text an vorgegebener Stelle aus
title	zeigt einen Titel an
xlabel	beschriftet die x-Achse
ylabel	beschriftet die y-Achse
colormap(white)	wechselt die Farbtabelle, für S/W-Monitore

B.6.2 Dreidimensionale Graphiken

```
≫ [X,Y,Z] = peaks(25)
≫ mesh(X,Y,Z)
```



```
≫ subplot(1,2,1);
≫ [X,Y,Z] = peaks(25);
≫ mesh(X,Y,Z);
≫ subplot(1,2,2);
≫ X=1:20;
≫ Y=1:20;
≫ Z(X,Y)=(-(cos(X/4)))'*(sin((20-Y)/10).^3);
≫ mesh(X,Y,Z);
```



B.6.3 Graphiken drucken

Graphik-Druckbefehl		
print [-dAusgabetyp] [-Optionen] [Dateiname]		
Ausgabetyp	-dps	Postscript für Schwarzweißdrucker
	-dpsc	Postscript für Farbdrucker
	-deps	Encapsulated Postcript
	-depsc	Encapsulated Color Postcript
Optionen	-P<Drucker>	Spezifiziert den zu benutzenden Drucker

Die Eingabe

```
≫ print fig4 -deps
```

erzeugt die Datei `fig4.eps`.

B.7 FORTGESCHRITTENES

B.7.1 MATLAB-Skripte

Wie schon unter B Skript-Dateien erwähnt, läßt sich eine Abfolge von MATLAB-Befehlen auch in einer Datei speichern. Diese läßt sich dann am Befehl-Fenster aufrufen und die gespeicherte Folge von Befehlen wird dann ausgeführt. Solche Dateien werden als MATLAB-Skripte oder M-Files bezeichnet. Alle o.g. Befehle lassen sich in einer Datei z.B. mit dem Namen `abc.m` zusammenstellen. Für die Wahl des Dateinamens gelten dabei die folgenden Regeln:

- das erste Zeichen ist ein Buchstabe und
- die Datei hat die Endung **.m** .

Um ein solches M-File zu erstellen, ruft man den Editor auf, der es erlaubt Text einzugeben und diesen als Datei zu speichern. Wenn man den Editor gestartet hat, gebe man die folgenden Befehle ein und speichere die Datei unter dem Namen `abc.m`

```
a = 1;
b = 3;
c = -5;
x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Um nun die Befehle aus der Datei `abc.m` auszuführen, gibt man im MATLAB-Befehlsfenster

```
≫ abc
```

ein. MATLAB sucht im aktuellen Pfad nach der Datei `abc.m` und führt die darin enthaltenen Befehle aus. Das aktuelle Verzeichnis wird angezeigt, wenn man

```
≫ pwd
```

eingibt (pwd, engl. print working directory).

B.7.2 Erstellen eigener Funktionen

Es wird sicherlich etwas komfortabler sein, eine eigene Funktion zu haben, der man *a,b* und *c* als Argument übergibt un die beiden Wurzeln als Eregebnis erhält, als jedesmal erneut ein eigenes M-File anzufertigen. Ein solches Programm könnte z.B. die folgende Datei `root.m` sein

```
%-----------------
modified "abc.m"
%-----------------

a = input('Enter a: ');
b = input('Enter b: ');
c = input('Enter c: ');

x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Die Prozentzeichen in den erstn Zeilen dienen der Dokumentation. Alles was einem solchen Zeichen folgt, wird von MATLAB nicht ausgewertet, sprich interpretiert. Die Argumente werden in deieser Funktion jedoch nur bedingt übergeben und ausgegeben. Eine Funktion die diese Aufgabe erfüllt ist

```
%-----------------
modified "abc.m"
%-----------------

function [x1, x2] = quadroot(a,b,c)

radical = sqrt(b^2 - 4*a*c);

x1 = (-b + radical)/(2*a)
x2 = (-b - radical)/(2*a)
```

Wenn eine Funktion keine Rückgabewerte hat, können die eckigen Klammern mit den Variablen und das Gleichheitszeichen fehlen. Fehlen die Eingabeparameter, so kann auch der Klammerausdruck nach `quadroot` fehlen. Auf die in der Funktion verwendeten Variablen, hier `radical`, kann man vom Befehlsfenster nicht zugreifen. Ist die Funktion in der Datei `quadroot.m` gespeichert, so erhält man die Wurzeln, indem man

```
[x1, x2] = quadroot(1,3,-5);
```

eingibt. Es kann vom Befehlsfenster oder einer anderen Datei immeer nur die erste Funktion in einer Datei aufgerufen werden. Dies bedeutet, in einer Datei können mehrere Funktionen stehen, aber nur die erste Funktion kann extern aufgerufen werden. Alle weiteren Funktionen können nur von Funktionen in der gleichen Datei aufgerufen werden. Für den Anfang ist es einfacher, wenn jede Datei nur eine Funktion enthält. Entscheidend für den Funktionsnamen ist der Name der Datei.

C SPEICHERFORMATE FÜR SCHWACH BESETZTE MATRIZEN

Im Folgenden werden wir einige Speicherformate für schwach besetzte Matrizen vorstellen, welche speziell im Zusammenhang mit Iterationsverfahren von Interesse sind. Den Speicherformaten für schwach besetzte Systeme kommen aus zweierlei Gründen eine besondere Bedeutung zu:

- Die Wahl der Speicherform hat direkte Auswirkungen auf die verwendeten Lösungsverfahren (spezielle Matrixform, z.B. Bandmatrix, symmetrische Matrix, etc.). Unter anderen wegen ihres Auffüllverhaltens (Erzeugung neuer Nichtnullelemente) der verschiedenen Algorithmen, der Zugriffshäufigkeit auf spezielle Einträge (z.B. der Diagonaleintäge) und somit ihrer Konsequenzen auf die Speicherstruktur.

- In den meisten höheren Programmiersprachen `C/C++`, `Java`, `Fortran`, etc. stehen für die meisten Standardprobleme der numerischen Linearen Algebra verschiedene Programmpakete (Lapack, Linpack, Eispack u.v.a.) mit fertig implementierten numerischen Algorithmen zur Verfügung. Diesen Bibliotheken müssen die Daten aber in bestimmten Datenstrukturen übergeben werden.

C.1 KOORDINATENFORMAT (COO-FORMAT)

Das einfachste Format zur Speicherung von schwach besetzten Matrizen für iterative Verfahren ist das Koordinatenformat (**Coo**rdinate Storage, COO-Format).
Es verwendet 3 Vektoren der Länge nnz$A$ (nnz $\simeq$ number of nonzero elements):

- ein Vektor „Wert “ vom Datentyp der Einträge a_{ij}, in dem die Nichtnullelemente von A in beliebiger Reihenfolge stehen,

- ein Index-Vektor „Zeilenindex“ und

- ein Index-Vektor „Spaltenindex“, in denen zu jedem Element von „Wert“ die entsprechenden Koordinaten, d.h. Zeilen- und Spaltenindices, aus der Matrix A eingetragen werden. Meist sind dies Vektoren vom `unsigned (long) integer`.

Dieses Format wird häufig wegen seiner Einfachheit verwendet (dient oft als eine von vielen möglichen Schnittstelle zu Bibliotheken). Der Umstand, daß die Nichtnull-Matrixelemente in beliebiger Reihenfolge im Feld „Wert“ stehen dürfen, ist sowohl ein Vorteil als auch ein Nachteil des COO-Formats. Einerseits können die bei der Lösung des Gleichungssystems neu auftretenden Nichtnullelemente einfach am Ende der Felder ohne irgendwelchen Umordnungsaufwand angehängt werden. Andererseits ist der Suchaufwand nach einem bestimmten Matrixelement enorm, da ja keinerlei Strukturinformation der Matrix A in das Speicherformat selbst eingeht (ist das gesuchte Element gleich Null, müssen die Felder komplett durchsucht werden, bis diese Information vorliegt). Ein weiterer Nachteil ist auch der immer noch recht hohe Speicherbedarf.

Ein Beipiel soll die Datenspeicherung illustrieren.

Beispiel C.1.1

$$A = \begin{pmatrix} 10 & 0 & 0 & 22 & 31 & 0 \\ -4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & -6 & 9 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 5 & 0 & 20 & 11 \\ 12 & 0 & 0 & 14 & 0 & 44 \end{pmatrix}$$

geht über zu

Wert a_{ij}	44	-6	20	12	14	31	5	-4	10	8	7	9	22	11
Zeilenindex i	6	3	5	6	6	1	5	2	1	2	4	3	1	5
Spaltenindex j	6	3	5	1	4	5	3	1	1	2	4	4	4	6

In Matlab kann man einen eigenen Datentypen `COO-Matrix` definieren und sowohl die Operatoren `+, *, -, /` überladen, als auch die Ein- und Ausgabe dem bekannten Verhalten anpassen. Im Folgenden beschränken wir uns auf die Wiedergabe einiger Methoden, die den Datentypen `coomatrix` definieren, Ein- und Ausgabe sowie Multiplikation ermöglichen, so dass man u.a. das CG-Verfahren testen könnte. Die Routinen bzw. Methoden zu einer Klasse stehen in Matlab üblicherweise in einem Unterverzeichnis gleichen Namens, d.h. die Methoden `coomatrix.m` etc. stehen im Verzeichnis `@coomatrix`. Für einen ersten Einstieg in Klassendefinitionen in Matlab, sei auf das Beispiel `polynom` in der Matlab-Dokumentation verwiesen.

MATLAB-Funktion: @coomatrix/coomatrix.m

```
1  function p = coomatrix(i,j,a)
2  %COOMATRIX Coordinate format class constructor.
3  %   A = COOMATRIX(i,j,a) creates a compressed matrix object from
4  %   the vectors i,j,a, containing the entries of a matrix A.
5  if nargin == 0
6     p.a = []; p.row = [];  p.col = [];
7     p.m = 0;
8     p.n = 0;
9     p = class(p,'coomatrix');
10 elseif nargin == 1 && isa(i,'coomatrix')
11    p = i;
12 elseif nargin == 1 && isnumeric(i)
13    [p.row,p.col,p.a] = find(i);
14    p.row=p.row';p.col=p.col';p.a = p.a';
15    p.m = max(p.row); p.n = max(p.col);
16    p = class(p,'coomatrix');
17 else
18    p.row = i(:).';
19    p.col = j(:).';
20    p.a   = a(:).';
21    p.m = max(p.row);
22    p.n = max(p.col);
23    p = class(p,'coomatrix');
24 end
```

MATLAB-Funktion: @coomatrix/display.m

```
function display(p)
% COOMATRIX/DISPLAY Command window display of coomatrix
formatedspace
disp([inputname(1),' = '])
formatedspace
short = ~(max(p.row)>9999 || max(p.col)>9999);
for k=1:length(p.row)
  i = num2str(p.row(k));  j = num2str(p.col(k));
  if short
    txt = '     ,     ';
    txt(5-length(i):5) = ['(',i];
    txt(7:7+length(j)) = [j,')'];
  else
    txt = '           ,           ';
    txt(11-length(i):11) = ['(',i];
    txt(13:13+length(j)) = [j,')'];
  end
  disp([txt,sprintf('%13.4f',p.a(k))])
end
formatedspace

function formatedspace
switch get(0,'FormatSpacing')
  case 'loose'
    disp(' ');
end
```

MATLAB-Funktion: @coomatrix/subsasgn.m

```
function p = subsasgn(p,s,b)
% SUBSASGN
switch s.type
  case '()'
    if size(s.subs,1)~=1 || size(s.subs,2)~=2
      error('??? Index exceeds matrix dimensions.')
    end
    ii = s.subs{1,1}; jj = s.subs{1,2};
    for i = 1:length(ii)
      for j = 1:length(jj)
        p.row = [p.row,ii(i)]; p.col = [p.col,jj(j)];
        p.a   = [p.a,b(i,j)];
      end
    end
    p.m = max(p.row); p.n = max(p.col);
otherwise
   error('Specify value for i and j as A(i,j)')
end
```

MATLAB-Beispiel:

Anlegen einer Matrix im COO-Format funktioniert mit der Anweisung `coomatrix`

Durch die Funktion `subsref` können einzelne Einträge zu der Matrix hinzugefügt. Indexpaare (i, j) können dabei doppelt auftreten.

```
>> A=coomatrix([1,2],[2,3],[1.2,1.4])
A =
   (1,2)          1.2000
   (2,3)          1.4000
>> A(1,4)=2;
>> A(1,4)=3
A =
   (1,2)          1.2000
   (2,3)          1.4000
   (1,4)          2.0000
   (1,4)          3.0000
```

MATLAB-Funktion: @coomatrix/subsref.m

```
function b = subsref(a,s)
% SUBSREF
switch s.type
case '()'
    if size(s.subs,1)~=1 || size(s.subs,2)~=2
      error('??? Index exceeds matrix dimensions.')
    end
    ii = s.subs{1,1};
    jj = s.subs{1,2};
    b = zeros(length(ii),length(jj));
    for i = 1:length(ii)
      if ii(i) > a.m, error('??? Index exceeds matrix dimensions.')
         ; end
      for j = 1:length(jj)
        if jj(j) > a.n, error('??? Index exceeds matrix dimensions.
           '); end
        b(i,j) = get_index(a,ii(i),jj(j));
      end
    end
otherwise
   error('Specify value for i and j as A(i,j)')
end

function aij = get_index(mat,i,j)
aij = 0;
ii = find(mat.row==i);
if ~isempty(ii)
  jj = find(mat.col(ii)==j);
  if ~isempty(jj)
    for k=1:length(jj)
      aij = aij + mat.a(ii(jj(k)));
    end
  end
end
```

MATLAB-Funktion: @coomatrix/mtimes.m

```
function r = mtimes(A,q)
% COOMATRIX/MTIMES   Implement A * q for coordinate matrix,
% q vector or scalar
if isa(A,'coomatrix') && isnumeric(q)
  if size(q,1) * size(q,2) == 1
    A.a = q * A.a;
    r = A;
    return
  elseif A.n ~= size(q,1),
    error('Inner matrix dimensions must agree.')
  end
  r = zeros(A.m,size(q,2));
  for j = 1:size(q,2)
    for k = 1:length(A.a)
      r(A.row(k)) = r(A.row(k)) + A.a(k) * q(A.col(k),j);
    end
  end
else
  error('Multiplication not defined for these types.')
end
```

MATLAB-Beispiel:

Durch die Funktion `subsasgn` funktioniert auch der Zugriff auf einzelne Einträge der Matrix wie aus Matlab bekannt. Da ein der Index (i, j) ggf. nicht eindeutig ist, muss die ganze Matrix durchsucht werden.

```
A(1,2)
ans =
    1.2000
>> A(1,3)
ans =
     0
```

Multiplikation einer Matrix im COO-Format mit einem Vektor lässt sich mittels des Überladens des $*$-Operators mit der Funktion `mtimes.m` wie gehabt realisieren.

```
>> A=coomatrix([1,2],[2,3],[1.2,1.4]);
>> b = A * [2;1;3]
b =
    1.2000
    4.2000
```

Aufgabe C.1.2 Realisieren Sie eine Methode `ctranspose`, welches die Verwendung des Operators Transponiert-Operators `'` in Matlab wie bekannt ermöglicht.

C.2 KOMPRIMIERTE ZEILENSPEICHERUNG (CRS-FORMAT)

Das CRS-Format (**C**ompressed **R**ow **S**torage-Format, Komprimierte Zeilenspeicherung) ist bereits eine relativ kompakte Speicherform, die aber nicht von einer bestimmten Struktur der Matrix A ausgeht. Sinn dieses Formates ist es, sehr schnell alle Nichtnullelemente einer bestimmten Zeile zu finden.
Es sei $A \in \mathbb{R}^{m\times n}$, dann benötigt das Format 3 Vektoren, 2 der Länge $\text{nnz}(A)$ und 1 mit der Länge $m+1$:

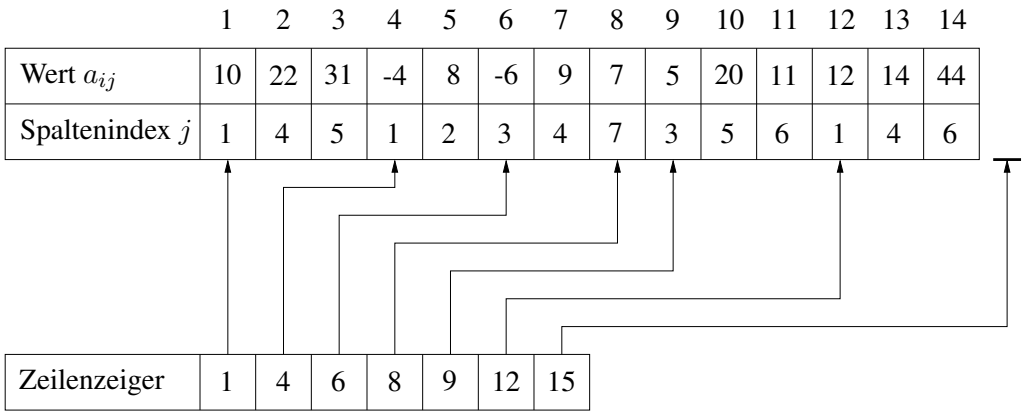
- ein Vektor „Wert“, in dem die Nichtnullelemente von A zeilenweise, von links nach rechts, stehen;
- einen Index-Vektor „Spaltenindex“, in dem zu jedem Element von „Wert“ der entsprechende Spaltenindex aus der Matrix A eingetragen wird;
- einen Index-Vektor „Zeilenzeiger“, in dem der Beginn jeder Matrixzeile im Feld „Wert“ gespeichert wird. Bei Index $m+1$ wird der Wert $\text{nnz}(A)+1$ abgelegt.

Das CRS-Format benötigt gegenüber dem COO-Format deutlich weniger Speicher, kein Eintrag kommt mehrfach vor, ein Index-Vektor ist i. Allg. deutlicher kürzer. Auch das Suchen nach einem bestimmten Matrixelement geht hier deutlich schneller. Werden vom Lösungsalgorithmus aber neue Nichtnullelemente erzeugt, dann ist die Einordnung doch komplizierter und auch mit mehr Aufwand verbunden als beim COO-Format.

Beispiel C.2.1
Die Mattrix

$$A = \begin{pmatrix} 10 & 0 & 0 & 22 & 31 & 0 \\ -4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & -6 & 9 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 5 & 0 & 20 & 11 \\ 12 & 0 & 0 & 14 & 0 & 44 \end{pmatrix}$$

geht über zu



MATLAB-Funktion: @crsmatrix/crsmatrix.m

```
function p = crsmatrix(i,j,a)
%CRSMATRIX Compressed row storage format class constructor.
%   A = CRSMATRIX(i,j,a) creates a compressed matrix object from
%   the vectors i,j,a, containing the entries of a matrix A.
if nargin == 0
   p.a = []; p.col = [];  p.ptr = [];
   p.m = 0;  p.n = 0;
   p = class(p,'crsmatrix');
elseif nargin == 1 && isa(i,'crsmatrix')
   p = i;
elseif nargin == 1 && isnumeric(i)
   [p.col,row,p.a] = find(i');
   p.n = max(p.col); p.m = max(row);
   p.col = p.col'; p.a = p.a';
   p.ptr = find([row(:);row(end)+1]-[0;row(:)])';
   p = class(p,'crsmatrix');
else
   p = crsmatrix(sparse(i,j,a));
end
```

MATLAB-Funktion: @crsmatrix/display.m

```
function display(p)
% CRSMATRIX/DISPLAY Command window display of crs-matrix
formatedspace, disp([inputname(1),' = ']), formatedspace
short = ~(max(p.col)>9999 || p.ptr(end)>9999+1);
for k = 2:length(p.ptr)
  for j = p.ptr(k-1):p.ptr(k)-1
    ii = num2str(k-1); jj = num2str(p.col(j));
    if short
      txt = '     ,     ';
      txt(5-length(i):5) = ['(',ii];
      txt(7:7+length(j)) = [jj,')'];
    else
      txt = '           ,           ';
      txt(11-length(i):11) = ['(',ii];
      txt(13:13+length(j)) = [jj,')'];
    end
    disp([txt,sprintf('%13.4f',p.a(j))])
  end
end
formatedspace

function formatedspace
switch get(0,'FormatSpacing')
  case 'loose'
    disp(' ');
end
```

MATLAB-Funktion: @crsmatrix/mtimes.m

```
function r = mtimes(A,q)
% CRSMATRIX/MTIMES   Implement A * q for crs-format matrix,
% q full matrix of matching size
if isa(A,'crsmatrix') && isnumeric(q)
  if A.n ~= size(q,1),
    error('Inner matrix dimensions must agree.')
  end
  r = zeros(A.m,size(q,2));
  for j = 1:A.m
    for k = A.ptr(j):A.ptr(j+1)-1
      r(j,:) = r(j,:) + A.a(k) * q(A.col(k),:);
    end
  end
else
  error('Multiplication not defined for these types.')
end
```

MATLAB-Beispiel:

Anlegen einer Matrix im COO-Format funktioniert mit der Anweisung `coomatrix`

```
>> A = crsmatrix([3,2,1,2,3,1], ...
>>               [2,2,4,3,1,1], ...
>>               [1,2,3,4,5,6])
A =
    (1,1)          6.0000
    (1,4)          3.0000
    (2,2)          2.0000
    (2,3)          4.0000
    (3,1)          5.0000
    (3,2)          1.0000
```

Aufgabe C.2.2 Realisieren Sie die Methoden `subsref, ctranspose`, welche den Zugriff auf einzelne Einträge und das Transponieren in Matlab wie bekannt ermöglicht.

C.3 MODIFIZIERTE KOMPRIMIERTE ZEILENSPEICHERUNG (MRS-FORMAT)

Beim MRS-Format (**M**odified Compresses **R**ow **S**torage-Format, Modifiziertes CRS-Format) wird nochmals versucht, den ohnehin schon geringen Speicherplatzbedarf des CRS-Formats noch weiter zu senken und direkten Zugriff auf die Diagonalelemente zu haben. denken Sie z.B. an das Jacobi-Verfahren, bei dem man nicht nur eine Matrix-Vektor-Multiplikation mit der Koeffizientenmatrix sondern auch mit der Inversen ihrer Diagonalen benötigt.
Beim MRS-Format geht man davon aus, dass die Hauptdiagonale voll (oder zumindest überdurchschnittlich hoch) besetzt ist, eine Bedingung, die fast alle in der Praxis vorkommenden Matrizen

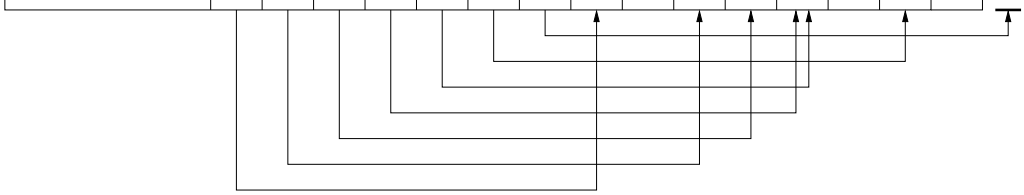
erfüllen. Speichert man nun die Hauptdiagonale extra ab, so wird keinerlei zusätzliche Koordinateninformation dafür benötigt, denn man weiß ja, daß z.B. das dritte Hauptdiagonalelement in der dritten Zeile und der dritten Spalte steht.
Diese Grundidee wird im MRS-Format folgendermaßen umgesetzt:
Es sei $A \in \mathbb{R}^{m\times n}$. In dem Vektor „Wert“ werden zuächst sämtliche Hauptdiagonalelemente eingetragen, danach wird ein Feld freigelassen. Ab Index $m+2$ werden nun wie beim CRS-Format zeilenweise die Nichnullelemente, natürlich ohne die Hauptdiagonalelemente, abgespeichert. Die ersten $m+1$ Elemente des Vektors „Index“ ersetzen das beim CRS-Format seperate Feld „Zeilenzeiger“. Ab Index $m+2$ werden dann wieder die Spaltenindizes der entsprechenden Matrixelemente des Feldes „Wert“ abgespeichert.
Vor- und Nachteile dieser Speicherform sind ähnlich dem CRS-Format. Das MRS-Format ist relativ beliebt.

Beispiel C.3.1

$$A = \begin{pmatrix} 10 & 0 & 0 & 22 & 31 & 0 \\ -4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & -6 & 9 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 5 & 0 & 20 & 11 \\ 12 & 0 & 0 & 14 & 0 & 44 \end{pmatrix}$$

geht über zu

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wert a_{ij}	10	8	-6	7	20	44		22	31	-4	9	5	11	12	14
Index	8	10	11	12	12	14	16	4	5	1	4	3	5	1	4



MATLAB-Funktion: @mrsmatrix/mrsmatrix.m

```
1  function p = coomatrix(i,j,a)
2  %COOMATRIX Coordinate format class constructor.
3  %   A = COOMATRIX(i,j,a) creates a compressed matrix object from
4  %   the vectors i,j,a, containing the entries of a matrix A.
5  if nargin == 0
6     p.a = []; p.ind = [];
7     p.m = 0;  p.n = 0;
8     p = class(p,'mrsmatrix');
9  elseif nargin == 1 && isa(i,'mrsmatrix')
10    p = i;
11 elseif nargin == 1 && isnumeric(i)
12    % dirty, at least one off-diagonal entry neccessary
13    [col,row,a] = find(i');
14    p.n = max(col); p.m = max(row);
15    ptr = find(col==row);
16    p.a = zeros(1,min(p.m,p.n));
17    p.a(col(ptr)) = a(ptr);
```

```
18      ptr = find(col~=row);     row = row(ptr);
19      p.ind = 1+min(p.m,p.n)+find([row(:);row(end)+1]-[0;row(:)])';
20      p.ind = [p.ind,col(ptr)'];
21      p.a   = [p.a,0,a(ptr)'];
22      p = class(p,'mrsmatrix');
23    else
24      p = mrsmatrix(sparse(i,j,a));
25    end
```

MATLAB-Funktion: @mrsmatrix/display.m

```
1   function display(p)
2   % MRSMATRIX/DISPLAY Command window display of mrs-matrix
3   formatedspace
4   disp([inputname(1),' = '])
5   formatedspace
6   short = ~(p.m>9999 || p.n>9999);
7   for k = 1:min(p.m,p.n)
8     if p.a(k)
9       printentries(k,k,p.a(k),short)
10    end
11  end
12  for k = 1:p.m
13    for j = p.ind(k):p.ind(k+1)-1
14      printentries(k,p.ind(j),p.a(j),short)
15    end
16  end
17  formatedspace
18
19  function formatedspace
20  switch get(0,'FormatSpacing')
21    case 'loose'
22      disp(' ');
23  end
24
25  function printentries(ii,jj,aij,short)
26  i = num2str(ii); j = num2str(jj);
27  if short
28    txt = '     ,     ';
29    txt(5-length(i):5) = ['(',i];
30    txt(7:7+length(j)) = [j,')'];
31  else
32    txt = '           ,           ';
33    txt(11-length(i):11) = ['(',i];
34    txt(13:13+length(j)) = [j,')'];
35  end
36  disp([txt,sprintf('%13.4f',aij)])
```

MATLAB-Funktion: @mrsmatrix/mtimes.m

```
function r = mtimes(A,q)
% MRSMATRIX/MTIMES   Implement A * q for mrs-format,
% q vector or matrix of matching size
if isa(A,'mrsmatrix') && isnumeric(q)
  if A.n ~= size(q,1),
    error('Inner matrix dimensions must agree.')
  end
  r = zeros(A.m,size(q,2));
  for j = 1:A.m
    r(j,:) = A.a(j) * q(j,:);
    for k = A.ind(j):A.ind(j+1)-1
      r(j,:) = r(j,:) + A.a(k) * q(A.ind(k),:);
    end
  end
else
  error('Multiplication not defined for these types.')
end
```

MATLAB-Beispiel:

Anlegen einer Matrix im MRS-Format funktioniert z.B. mit der Anweisung `mrsmatrix`. Auch die Multiplikation lässt sich kanonisch realisieren.

```
>> A = mrsmatrix([3,2,1,2,3,1], ...
>>               [2,2,4,3,1,1], ...
>>               [1,2,3,4,5,6])
A =
    (1,1)          6.0000
    (2,2)          2.0000
    (1,4)          3.0000
    (2,3)          4.0000
    (3,1)          5.0000
    (3,2)          1.0000
>> x = [0,1;1,1;2,1;2 0];
>> A*x
ans =
     6     6
    10     6
     1     6
```

Aufgabe C.3.2 Realisieren Sie die Methoden `subsref, ctranspose`, welche den Zugriff auf einzelne Einträge und das Transponieren in Matlab wie bekannt ermöglicht.

Aufgabe C.3.3 Schreiben Sie eine Funktion, die das Jacobi-Verfahren zur Lösung von linearen Gleichungsststemen für in CRS-Format gespeicherte Matrizen umsetzt. (Tipp: Neben `A*x` benötigen Sie noch eine Routine, die `diag(diag(A))\ x` effizient umsetzt.

C.4 HARWELL-BOEING-FORMAT (CCS-FORMAT)

Das Harwell-Boeing-Format oder auch CCS (**C**ompressed-**C**olumn-**S**torage-Format, Komprimierte Spaltenspeicherung) genanntes Format entspricht im wesentlichen dem CRS-Format, nur dass man hier nicht schnell auf eine Zeile, sondern eine komplette Spalte zugreifen möchte. Die Speicherung zeilenweise von links nach rechts wird durch spaltenweise Speicherung von oben nach unten ersetzt.

Literaturverzeichnis

[Deuflhard/Hohmann] P. DEUFLHARD, A. HOHMANN, Numerische Mathematik 1, de Gruyter-Verlag, Berlin, 1993.

[Fischer] G. FISCHER, Linare Algebra, Vieweg-Verlag, Wiesbaden, 1989.

[Analysis I] S. FUNKEN, Skript zur Vorlesung „Analysis I“, gehalten im Wintersemester 07/08 an der Universität Ulm.

[Analysis II] S. FUNKEN, Skript zur Vorlesung „Analysis II“, gehalten im Sommersemester 2008 an der Universität Ulm.

[Discroll/Maki] T. A. DISCROLL, K. L. MAKI: Searching for Rare Growth Factors Using Multicanonical Monte Carlo Methods. SIAM Review. Vol. 49, No. 4, pp. 673-692. 2008.

[Golub/Loan] G. H. GOLUB, C. F. VAN LOAN, Matrix Computations, 3. ed., Hopkins Univ. Press, 1996.

[Hackbusch] W. HACKBUSCH, Iterative Lösung großer schwachbesetzter Gleichungssysteme, 2. Auflage, Teubner-Verlag, Stuttgart, 1993.

[Hämmerlin/Hoffmann] G. HÄMMERLIN, K.-H. HOFFMANN, Numerische Mathematik, 4. Auflage, Springer-Verlag, Berlin, 1994.

[Hanke] M. HANKE-BOURGEOIS, Grundlagen der Numerischen Mathematik und des wissenschaftlichen Rechnens , 1. Auflage, Teubner-Verlag, Stuttgart, 2002.

[Higham] N. J. HIGHAM, How accurate is Gaussian elimination? Numerical Analysis 1989, Proceedings of the 13th Dundee Conference, Vol. 228 of Pitman Research Notes in Mathematics, 137–154.

[Knuth] D.E. KNUTH The Art of Computer Programming, Band 2, Addison-Wesley, 3. Auflage, 1998.

[Niethammer] W. NIETHAMMER, Relaxation bei nichtsymmetrischen Matrizen. Math. Zeitschr. **85** 319-327, 1964.

[Plato] R. PLATO, Numerische Mathematik kompakt, Vieweg-Verlag.

[Quateroni et. al.] A. QUATERONI, R. SACCO, F. SALERI, Numerische Mathematik, Band 1 & 2, Springer-Verlag, Berlin, 2002.

[Schwarz] H. R. SCHWARZ, Numerische Mathematik, 4. Auflage, Teubner-Verlag, Stuttgart, 1997.

[Stör/Bulirsch] J. STÖR, R. BULIRSCH, Numerische Mathematik, Band 1 & 2, Springer-Verlag, Berlin, 1994.

[Törnig/Spellucci] W. TÖRNIG, P. SPELLUCCI, Numerische Mathematik für Ingenieure und Physiker, Band 1 & 2, Springer-Verlag, Berlin, 1988.

[Wilkinson65] J. H. WILKINSON, The Algebraic Eigenvalue Problem, Oxford University Press, 1965. ´

[Wilkinson69] J. H. WILKINSON, Rundungsfehler, Springer-Verlag, Berlin, Heidelberg, New York, 1969.

[Wille] D. WILLE, Repetitorium der Linearen Algebra, Teil 1, 1. Auflage, Feldmann-Verlag, Springe, 1989.

Index