Kapitel 2

Das Gaußsche Eliminationsverfahren für lineare Gleichungssysteme

2.1 Einleitung

Reelle lineare Gleichungssysteme lassen sich schreiben in der Form

$$Ax = b \tag{2.1}$$

wobei $A \in \mathbb{R}^{n,m}$ eine Matrix ist, $b \in \mathbb{R}^n$ die rechte Seite und $x \in \mathbb{R}^m$ die gesuchte Lösung. Falls $m = n$ und A invertierbar ist (was wir ab jetzt annehmen werden), gilt für die eindeutige Lösung $x = A^{-1}b$.

Ein Beispiel

Wir betrachten das Gleichungssystem

$$\begin{array}{rcrcrcr} x_1 & + & x_2 & + & x_3 & = & 3 \\ 2x_1 & + & 3x_2 & + & 4x_3 & = & 9 \\ 3x_1 & + & 1x_2 & + & 2x_3 & = & 6 \end{array} \tag{2.2}$$

mit

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 3 & 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 9 \\ 6 \end{pmatrix}, \quad x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Die Idee des Gaußschen Eliminationsverfahrens zur Lösung von (2.1) ist uralt. Sie basiert darauf, geeignete Vielfache einer Gleichung von anderen Gleichungen zu subtrahieren und so schließlich auf ein äquivalentes System zu kommen, in welchem in jeder Gleichung nur noch eine Variable vorkommt. In (2.2) könnte man etwa folgende Transformationen durchführen:

$$(2.2) \Rightarrow \begin{array}{rcrcrcr} x_1 & + & x_2 & + & x_3 & = & 3 \\ & & x_2 & + & 2x_3 & = & 3 \\ & & -2x_2 & - & x_3 & = & -3 \end{array} \Rightarrow \begin{array}{rcrcrcr} x_1 & + & x_2 & + & x_3 & = & 3 \\ & & x_2 & + & 2x_3 & = & 3 \\ & & & & 3x_3 & = & 3 \end{array}$$

Dabei haben wir im ersten Schritt das 2-fache der ersten Zeile von der zweiten Zeile und das 3-fache der ersten Zeile von der dritten Zeile und im zweiten Schritt das -2-fache der zweiten Zeile von der dritten Zeile subtrahiert. In Matrix-Schreibweise lassen sich die Eliminationsschritte einfacher darstellen als

$$(A|b) = \left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 2 & 3 & 4 & 9 \\ 3 & 1 & 2 & 6 \end{array}\right) \to \left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & -2 & -1 & -3 \end{array}\right) \to \left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 3 & 3 \end{array}\right) := (R|y)$$

Aus dem transformierten System $(R|y)$ läßt sich die Lösung x nun sehr einfach bestimmen, indem man sukzessive die Zeilen 3, 2 und 1 durch *Rückwärtseinsetzen* eliminiert. So erhält man

$$\begin{aligned} &\text{Zeile 3: } 3x_3 = 3 \Rightarrow x_3 = 1 \\ \Rightarrow &\text{Zeile 2: } x_2 + 2x_3 = 3 \Rightarrow x_2 = 1 \\ \Rightarrow &\text{Zeile 1: } x_1 + x_2 + x_3 = 3 \Rightarrow x_1 = 1 \end{aligned}$$

Der Grund, warum sich das transformierte System $(R|y)$ so einfach auflösen läßt, ist offensichtlich, daß $R_{i,j} = 0$ für $i > j$, daß also R sogenannte *rechte Dreiecksgestalt* besitzt.

Gaußelimination für allgemeines n

Bei größeren Gleichungssystemen, d.h. für allgemeines n, verfährt man im Prinzip genauso: Erst wird das System $(A|b)$ durch $n-1$ Eliminationsschritte auf rechte Dreiecksgestalt $(R|y)$ transformiert. Nach dem k-ten Schritt erhalten wir ein System $(A^{(k)}|b^{(k)})$ mit

$$a_{i,j}^{(k)} = \begin{cases} a_{i,j}^{(k-1)} & \text{falls } i \le k \text{ oder } j \le k-1 \\ a_{i,j}^{(k-1)} - a_{k,j}^{(k-1)} a_{i,k}^{(k-1)} / a_{k,k}^{(k-1)} & \text{falls } i \ge k+1 \text{ und } j \ge k \end{cases} \tag{2.3}$$

und $b^{(k)}$ die rechte Seite nach dem k-ten Schritt. Dabei ist $(A^{(0)}|b^{(0)}) := (A|b)$. Man verifiziere, daß durch (2.3) die Einträge $a_{k+1,k}^{(k)}, \dots, a_{n,k}^{(k)}$ eliminiert werden. Es gilt also

$$A^{(k)} = \left(\begin{array}{cccc|cccc} a_{1,1}^{(k)} & \cdots & \cdots & a_{1,k}^{(k)} & a_{1,k+1}^{(k)} & \cdots & \cdots & a_{1,n}^{(k)} \\ 0 & \ddots & & \vdots & \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & a_{k,k}^{(k)} & a_{k,k+1}^{(k)} & \cdots & \cdots & a_{k,n}^{(k)} \\ \hline 0 & \cdots & \cdots & 0 & a_{k+1,k+1}^{(k)} & \cdots & \cdots & a_{k+1,n}^{(k)} \\ \vdots & \ddots & & \vdots & \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{n,k+1}^{(k)} & \cdots & \cdots & a_{n,n}^{(k)} \end{array}\right) := \left(\begin{array}{c|c} A_{1,1}^{(k)} & A_{1,2}^{(k)} \\ \hline 0 & A_{2,2}^{(k)} \end{array}\right) \in \left(\begin{array}{c|c} \mathbb{R}^{k,k} & \mathbb{R}^{k,n-k} \\ \mathbb{R}^{n-k,k} & \mathbb{R}^{n-k,n-k} \end{array}\right). \tag{2.4}$$

```
do k=1,n-1                         ! n - 1 Eliminationsschritte
  do i=k+1,n                       ! Transformiere Zeilen k + 1 bis n
    r = A(i,k)/A(k,k)              ! Faktor für i-te Zeile
    do j=k,n                       ! Ziehe r-faches der k-ten Zeile von der
      A(i,j) = A(i,j) - r*A(k,j)   ! i-ten Zeile ab, a_{i,k}^{(k)} wird dadurch 0,
    end do                         ! a_{i,j}^{(k)} wurde schon zu 0 für j < k in
    b(i) = b(i) - r*b(k)           ! vorherigen Eliminationsschritten
  end do                           ! Schleifenende
end do                             ! Schleifenende
```

Algorithmus 2.1: Transformation eines linearen Gleichungssystems auf rechte Dreiecksgestalt

```
do i=n,1,-1                        ! Bestimme sukzessive x_n bis x_1
  s = b(i)                         ! Berechne s = y_i - \sum_{j=i+1}^n R_{i,j} x_j
  do j=i+1,n                       ! Die x_{i+1} bis x_i in dieser Summe
    s = s - A(i,j)*x(j)            ! wurden vorher schon berechnet
  end do                           ! Schleifenende
  x(i) = s/A(i,i)                  ! x_i kann nun berechnet werden
end do                             ! Schleifenende
```

Algorithmus 2.2: Bestimmung der Lösung durch Rückwärtseinsetzen

Der linke obere Teil $A_{1,1}^{(k)} \in \mathbb{R}^{k,k}$ hat bereits rechte Dreiecksgestalt. Im $k+1$en Schritt ändert sich nur noch $A_{2,2}^{(k)} \in \mathbb{R}^{n-k,n-k}$, d.h. es werden die Einträge $a_{k+2,k+1}^{(k)}, \dots, a_{n,k+1}^{(k)}$ eliminiert. Nach dem $k+1$en Schritt ist $A_{1,1}^{(k+1)} \in \mathbb{R}^{k+1,k+1}$ und $A_{2,2}^{(k+1)} \in \mathbb{R}^{n-k-1,n-k-1}$. Man fährt fort, bis schließlich nach dem $n-1$en Schritt die Matrix vollständig auf rechte Dreiecksgestalt gebracht ist, man das System also nach $(A^{(n-1)}|b^{(n-1)}) = (R|y)$ transformiert hat. Algorithmus 2.1 beschreibt die Vorgehensweise für beliebiges n.

Nach Durchführung dieses Algorithmus enthalten die `A(i,j)`, `b(j)` die Koeffizienten des Systems $(R|y)$, welches wie oben beschrieben leicht durch Rückwärtseinsetzen gelöst werden kann: Die i-te Gleichung des Systems $(R|y)$ ist

$$\sum_{j=1}^{n} R_{i,j} x_j = \sum_{j=i}^{n} R_{i,j} x_j = y_i \Leftrightarrow x_i = \left(y_i - \sum_{j=i+1}^{n} R_{i,j} x_j \right) / R_{i,i}. \tag{2.5}$$

Für $i = n$ ist die Summe $\sum_{j=i+1}^{n}$ einfach Null, man erhält also $x_n = y_n / R_{n,n}$. Für $i < n$, wenn man $x_{i+1}, \dots, x_n$ schon kennt, kann man aus (2.5) x_i berechnen. Das ganze ist in (Algorithmus 2.2) zusammengefaßt.

Rechenaufwand

Satz 2.1 (Rechenaufwand des Gaußverfahrens) *Algorithmus 2.1 benötigt* $n^3/3 + O(n^2)$ *Rechenoperationen. Dabei ist eine Rechenoperation definiert als eine Addition/Subtraktion plus eine Multiplikation/Division.*

Beweis: Algorithmus 2.1 enthält drei ineinander verschachtelte do-Schleifen: Die äußere Schleife (Zeile 1-9) enthält eine innere Schleife (Zeile 2-8). Diese enthält wiederum zwei Operationen (Zeile 3 und 7) sowie eine innerste Schleife (Zeile 4-6) mit einer Operation (Zeile 5). Die Gesamtzahl $\mathcal{A}(n)$ aller im Algorithmus ausgeführten Anweisungen ist also gegeben durch

$$\begin{aligned}
\mathcal{A}(n) &= \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} \left(1 + \left(\sum_{j=k+1}^{n} 1\right) + 1\right) = \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} 2 + n - k \\
&= \sum_{k=1}^{n-1} (n-k)(2+n-k) = \sum_{k=1}^{n-1} (n-k)^2 + 2(n-k) = \sum_{k=1}^{n-1} k^2 + 2k \\
&=^{(*)} \frac{n^3}{3} + \frac{n^2}{2} + \frac{5n}{6} = \frac{n^3}{3} + O(n^2).
\end{aligned}$$

Dabei kann man (*) z.B. mit Induktion über n zeigen. □

Im Prinzip kann mit dieser einfachsten Version der Gaußschen Elimination jedes Gleichungssystem für nichtsinguläres $A \in \mathbb{R}^{n,n}$ exakt gelöst werden. In der Praxis sind allerdings noch einige Punkte zu berücksichtigen:

- Es können Einträge $a_{k,k}^{(k)}$ (die sogenannten *Pivotelemente*[1]) in Algorithmus 2.1 während der Elimination 0 sein. Man hätte dann eine Division durch Null.

- Es können Rundungsfehler auftreten, so daß man nicht erwarten kann, eine exakte Lösung zu erhalten. In ungünstigen Fällen können Rundungsfehler das Ergebnis sehr stark verfälschen.

- Es ist zweckmäßig, die Eliminationsschritte irgendwie zu speichern, um mehrere Gleichungssysteme mit gleichem A aber verschiedenen rechten Seiten b lösen zu können, ohne daß die Eliminationsschritte immer wieder neu berechnet werden müssen. Dabei kann man z.B die bei der Gaußelimination auftretenden Faktoren `A(i,k)/A(k,k)` in den „freiwerdenden“ Einträgen von `A` speichern.

Wir werden im folgenden nur die ersten beiden dieser Punkte diskutieren.

[1]frz. pivot: Drehpunkt, Angelpunkt

2.2 Pivotsuche

Wie vorher schon bemerkt, läßt sich der k-te Gaußschritt (2.3) nur durchführen, wenn das Pivotelement $a_{k,k}^{(k-1)}$ nicht gleich Null ist. Falls $a_{k,k}^{(k-1)} = 0$, kann man das aber leicht dadurch beheben, indem man vor dem eigentlichen Gaußschritt die k-te Zeile von $A^{(k-1)}$ (das entspricht der ersten Zeile der Untermatrix $A_{2,2}^{(k-1)}$) mit einer geeigneten i-ten Zeile vertauscht, für die $i > k$ und $a_{i,k}^{(k-1)} \neq 0$ gilt. Es bietet sich in diesem Fall an, ein $i > k$ auszuwählen mit

$$a_{i,k}^{(k-1)} = \max_{l>k} |a_{l,k}^{(k-1)}| > 0,$$

also das betragsgrößte Element der ersten Spalte von $A_{2,2}^{(k-1)}$. Man kann sich leicht überlegen, daß nicht alle $a_{i,k}^{(k-1)}$ Null sein können, weil sonst A singulär wäre.

Diesen Prozeß der Suche nach dem betragsgrößten Element bezeichnet man als *Spaltenpivotsuche*. Es gibt auch noch die Möglichkeit der *Totalpivotsuche*, wobei die gesamte Untermatrix $A_{2,2}^{(k-1)}$ nach dem betragsgrößten Eintrag durchsucht wird. Das erfordert allerdings auch Vertauschungen von Spalten und somit eine Umnumerierung der Unbekannten x_i des Gleichungssystems während der Elimination. Wir werden im folgenden nur die Spaltenpivotsuche betrachten.

Man sollte Spaltenpivotsuche auch durchführen, wenn $a_{k,k}^{(k-1)} \neq 0$ ist. Damit erreicht man, das alle Faktoren $a_{i,k}^{(k-1)}/a_{k,k}^{(k-1)}$ betragsmäßig ≤ 1 sind, was numerisch günstiger ist, wie wir an einem Beispiel demonstrieren werden.

Algorithmus 2.3 beschreibt die Transformation eines linearen Gleichungssystems auf rechte Dreiecksgestalt mit Spaltenpivotsuche. Im Gegensatz zu Algorithmus 2.1 liefert er, wenn man Rundungsfehler nicht berücksichtigt, für jede nichtsinguläre Matrix A und jede rechte Seite b die Reduktion auf Dreiecksgestalt.

Instabilität des Gaußverfahrens ohne Pivotsuche — Ein Beispiel

Das Gaußverfahren ohne Pivotsuche bricht bei Null-Pivotelementen $a_{k,k}^{(k)} = 0$ mit einer Fehlermeldung „Division durch Null“ ab, wie man an folgendem Beispiel sieht:

$$(A|b) = \left(\begin{array}{ccc|c} 13 & 13 & 1 & 42 \\ 7 & 7 & 2 & 27 \\ 10 & 1 & 1 & 15 \end{array}\right) \tag{2.6}$$

hat die eindeutige Lösung $x = (1,2,3)^T$. Ohne Pivotsuche, bei exakter Rechnung (z.B. per Hand), bricht das Verfahren nach dem ersten Schritt ab, da $a_{2,2}^{(1)} = 0$:

$$(A|b) \to \left(\begin{array}{ccc|c} 13 & 13 & 1 & 42 \\ 0 & 0 & 19/3 & 57/13 \\ 0 & -9 & 3/13 & -225/13 \end{array}\right) \to \text{„Fehler: Division durch Null“}$$

```
do k=1,n-1                             ! n − 1 Eliminationsschritte
  imax = k                             ! Bestimme Index i_max für Pivotelement
  do i=k+1,n                           ! i_max ist Zeile mit dem größten Element
    if abs(A(i,k))>abs(A(imax,k)) imax=i
  end do
  do j=k,n                             ! Vertausche Zeilen k und i_max
    r = A(k,j); A(k,j) = A(imax,j); A(imax,j) = r
  end do
  r = b(k); b(k) = b(imax); b(imax) = r    ! Vertausche b_k und b_i_max
  do i=k+1,n                           ! ab hier identisch mit Alg. 2.1
    r = A(i,k)/A(k,k)
    do j=k,n
      A(i,j) = A(i,j) - r*A(k,j)
    end do
    b(i) = b(i) - r*b(k)
  end do
end do
```

Algorithmus 2.3: Transformation eines linearen Gleichungssystems auf rechte Dreiecksgestalt mit Spaltenpivotsuche

Nun schauen wir uns einmal an, was passiert, wenn man die Rechnung nicht exakt, sondern auf einem Computer mit IEEE-Arithmetik durchführt. In diesem Fall entsteht bei der Berechnung von

$$a_{2,2}^{(1)} = a_{2,2} - \frac{a_{2,1}}{a_{1,1}} a_{1,2} = 7 - \frac{7}{13} 13 \tag{2.7}$$

ein Rundungsfehler: (2.7) wird nicht als 0 ausgewertet wie bei exakter Rechnung, sondern als ein $\epsilon \neq 0$. Somit erhalten wir im zweiten Schritt keine Division durch Null (und damit eine Fehlermeldung), sondern eine „legale“ Division durch ein „fast-null“-Pivotelement ϵ:

$$(A|b) \to \left(\begin{array}{ccc|c} 13 & 13 & 1 & 42 \\ 0 & \epsilon & 19/13 & 57/13 \\ 0 & -9 & 3/13 & -225/13 \end{array}\right) \to \left(\begin{array}{ccc|c} 13 & 13 & 1 & 42 \\ 0 & \epsilon & 19/13 & 57/13 \\ 0 & 0 & r_1 & r_2 \end{array}\right) = (R|y)$$

mit

$$r_1 = \frac{3 + 171\epsilon^{-1}}{13}, \quad r_2 = \frac{-225 + 513\epsilon^{-1}}{13}.$$

Zur Vereinfachung haben wir dabei eventuell auftretende weitere Rundungsfehler bei der Berechnung der übrigen Brüche 19/13, 57/13 etc. sowie r_1, r_2 nicht berücksichtigt. Beim Rückwärtseinsetzen erhält man nun

$$\begin{aligned} x_3 &= \frac{-225 + 513\epsilon^{-1}}{3 + 171\epsilon^{-1}} = \frac{513}{171} + \epsilon' = 3 + \epsilon' \text{ mit } \epsilon' = O(\epsilon), \\ x_2 &= \frac{57 - 19(3 + \epsilon')}{13\epsilon} = -\frac{19\epsilon'}{13\epsilon}. \end{aligned}$$

(Auch diese Rechnung ist bedingt durch Rundungsfehler nicht ganz exakt.) Wie x_1 aussieht, ist hier schon egal: Wir sehen, daß x_2 allein (bis auf einen Faktor) durch den Quotienten der entstehenden Rundungsfehler bestimmt ist. Es ist i.A. nicht zu erwarten, daß dieser Quotient genau oder auch nur ungefähr gleich $-2 \cdot 13/19$ ist, daß also für x_2 der korrekte Wert 2 angenommen wird.

Ein Lauf von Algorithmus 2.1 bestätigt diese Befürchtung: Für $(A|b)$ wie in (2.6) erhält man (und zwar **ohne Fehlermeldung oder Warnung!**) bei einfach genauer Rechnung als (falsches) Ergebnis (hier nur auf 4 Stellen genau angegeben)

$$(R|y) = \left(\begin{array}{ccc|c} \text{1.300e+01} & \text{1.300e+01} & \text{1.000e+00} & \text{4.200e+01} \\ \text{0.000e+00} & \text{-4.768e-07} & \text{1.462e+00} & \text{4.385e+00} \\ \text{0.000e+00} & \text{0.000e+00} & \text{-2.759e+07} & \text{-8.276e+07} \end{array}\right), \quad x = \begin{pmatrix} \text{2.000e+00} \\ \text{1.000e+00} \\ \text{3.000e+00} \end{pmatrix}.$$

(also $\epsilon \simeq -4.708 \cdot 10^{-7}$, $r_1 \simeq -2.759 \cdot 10^7$, $r_2 \simeq -8.276 \cdot 10^7$.)

Mit Pivotsuche und exakter Rechnung hingegen liefert der Gauß-Algorithmus

$$(A|b) \to \left(\begin{array}{ccc|c} 13 & 13 & 1 & 42 \\ 0 & 0 & 19/13 & 57/13 \\ 0 & -9 & 3/13 & -225/13 \end{array}\right) \xrightarrow[\text{Pivotsuche}]{} \left(\begin{array}{ccc|c} 13 & 13 & 1 & 42 \\ 0 & -9 & 3/13 & -225/13 \\ 0 & 0 & 19/13 & 57/13 \end{array}\right) = (R|y).$$

(Im letzten Schritt passiert wegen $a_{3,2}^{(2)} = 0$ nichts mehr.) Die folgenschwere Division durch $\epsilon \simeq 0$ wird hier vermieden, so daß man erwarten kann, daß Algorithmus 2.3 ein Ergebnis liefert, welches auf einige Stellen genau ist:

$$(R|y) = \left(\begin{array}{ccc|c} \text{1.300e+01} & \text{1.300e+01} & \text{1.000e+00} & \text{4.200e+01} \\ \text{0.000e+00} & \text{-9.000e+00} & \text{2.308e-01} & \text{-1.731e+01} \\ \text{0.000e+00} & \text{0.000e+00} & \text{1.462e+00} & \text{4.385e+00} \end{array}\right), \quad x = \begin{pmatrix} \text{1.000e+00} \\ \text{2.000e+00} \\ \text{3.000e+00} \end{pmatrix}.$$

2.3 Spezielle Lineare Gleichungssysteme

Wir haben in den bisherigen Abschnitten gesehen, daß die Lösung eines allgemeinen linearen Gleichungssystems etwa $n^3/3$ Operationen erfordert. Viele Gleichungssysteme haben aber eine spezielle Struktur, die man bei der Lösung nicht außer acht lassen sollte, weil sie oft hilft, den Lösungsaufwand erheblich zu minimieren. In diesem Abschnitt wollen wir zwei spezielle Gleichungssysteme vorstellen.

Tridiagonalsysteme

In vielen Anwendungen treten Gleichungssysteme auf, deren Matrizen *schwachbesetzt* sind. Das bedeutet, daß „die meisten" Einträge $a_{i,j}$ gleich Null sind. Oft haben die wenigen von Null verschiedenen Einträge dann noch eine spezielle Struktur, wie zum Beispiel bei *Bandmatrizen*. Bei diesen sind alle Nicht-Null-Einträge in der Nähe der Diagonalen konzentriert: Es gibt ein (kleines) $m \in \mathbb{N}$ mit $a_{i,j} = 0$ für $|i - j| > m$. Die Zahl $2m - 1$

heißt dann die *Bandbreite* von A. Matrizen der Bandbreite 1 ($m = 1$) sind einfach Diagonalmatrizen, solche der Bandbreite 3 ($m = 2$) *Tridiagonalmatrizen*. Eine 5×5-Tridiagonalmatrix sieht wie folgt aus:

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

Es gilt:

Satz 2.2 (Rechenaufwand für Tridiagonalsysteme) *Tridiagonale Gleichungssysteme sind (mit Spaltenpivotsuche) mit $O(n)$ Rechenoperationen lösbar.*

Beweis: Man mache sich klar, daß bei Tridiagonalsystemen im k-ten Gauß-Eliminationsschritt nur eine einzige, nämlich die $k+1$-te Zeile zu transformieren ist (im Gegensatz zu den Zeilen $k+1, \ldots, n$ bei allgemeinen Systemen). In dieser $k+1$-ten Zeile stehen höchstens 3 zu transformierende Nicht-Null-Einträge, nämlich $a_{k+1,k}$, $a_{k+1,k+1}$ und (für $k < n-1$) $a_{k+1,k+2}$. Die Tridiagonalgestalt bleibt bei allen $A^{(k)}$ während der Gaußelimination erhalten. Ebenso beschränkt sich die Spaltenpivotsuche auf die Zeilen k und $k+1$.
Im k-ten Gauß-Schritt sind also nur $O(1)$ Rechenoperationen durchzuführen (im Gegensatz zu den $O((n-k)^2)$ Rechenoperationen bei allgemeinen Systemen). Da $n-1$ Schritte durchgeführt werden, folgern wir zunächst, daß ein Tridiagonalsystem $(A|b)$ mit $O(n)$ Operationen auf rechte Dreiecksgestalt $(R|y)$ zu bringen ist.
Da alle $A^{(k)}$ tridiagonal sind, ist auch R tridiagonal. Tridiagonale rechte Dreieckssysteme lassen sich in $O(n)$ lösen, wie man sich leicht überlegt.
Insgesamt haben wir also einen Aufwand von $O(n)$ (für die Dreiecks-Reduktion) plus $O(n)$ (für die Rückwärtssubstitution), also insgesamt $O(n)$. □

Man sieht an dieser Stelle, wie wichtig es ist, für jede Anwendung die Auwahl des Algorithmus sorgfältig zu treffen. Es wäre eine große Verschwendung an Rechenzeit, tridiagonale Systeme mit dem allgemeinen Gauß-Algorithmus, welcher nicht die tridiagonale Struktur berücksichtigt, zu lösen.

Das Cholesky-Verfahren für symmetrische, positiv definite Systeme

Eine symmetrische Matrix A heißt *positiv definit*, falls alle Eigenwerte von A positiv sind (symmetrische Matrizen haben nur reelle Eigenwerte). Insbesondere sind positiv definite Matrizen nichtsingulär und haben positive Determinanten. Positive Definitheit von A ist äquivalent zu folgender Bedingung: $\langle Ax, x \rangle > 0$ für alle $x \neq 0$. Das *Cholesky-Verfahren* baut auf folgendem Satz auf:

Satz 2.3 (Cholesky-Zerlegung) *Für jede symmetrische, positiv definite Matrix $A \in \mathbb{R}^{n,n}$ gibt es genau eine linke Dreiecksmatrix L, die nur positive Werte auf der Diagonalen hat, so daß $A = LL^T$.*

Beweis: Wir zeigen den Satz mit Induktion über n.

1. $n = 1$: Positiv definite 1×1-Matrizen sind einfach positive Zahlen $A > 0$, für die $A = LL^T = L^2$, $L > 0$ genau dann gilt, wenn $L = +\sqrt{A} > 0$.

2. $n - 1 \to n$: Wir schreiben $A, L \in \mathbb{R}^{n,n}$ als Blockmatrizen
$$A = \begin{pmatrix} A' & b^T \\ b & a_{n,n} \end{pmatrix}, \quad L = \begin{pmatrix} L' & 0 \\ c & l_{n,n} \end{pmatrix}$$
mit $A', L' \in \mathbb{R}^{n-1,n-1}$, $b, c \in \mathbb{R}^{1,n-1}$, $a_{n,n}, l_{n,n} \in \mathbb{R}$, A' symmetrisch, positiv definit, L' linke Dreiecksmatrix. L ist hierbei noch derart zu bestimmen, daß $A = LL^T$, daß also
$$\begin{pmatrix} A' & b^T \\ b & a_{n,n} \end{pmatrix} = \begin{pmatrix} L' & 0 \\ c & l_{n,n} \end{pmatrix} \begin{pmatrix} (L')^T & c^T \\ 0 & l_{n,n} \end{pmatrix} = \begin{pmatrix} L'(L')^T & L'c^T \\ c(L')^T & cc^T + l_{n,n}^2 \end{pmatrix},$$
woraus wir drei Bestimmungsgleichungen
$$A' = L'(L')^T, \quad L'c^T = b^T, \quad a_{n,n} = cc^T + l_{n,n}^2.$$
für L', c und $l_{n,n}$ erhalten ($b = c(L')^T$ ist äquivalent zu $b^T = L'c^T$).

   Die erste dieser drei Gleichungen ist nach Induktionsvoraussetzung eindeutig nach einer linken Dreiecksmatrix L' mit positiven Diagonalelementen aufzulösen. Die zweite Gleichung ist dann ein lineares Gleichungssystem für c^T, welches sich eindeutig lösen läßt: $c^T = (L')^{-1}b^T$. Die dritte Gleichung schreiben wir als $l_{n,n}^2 = a_{n,n} - cc^T := \alpha$. Sie läßt sich auf jeden Fall (evtl. in $\mathbb{C}$) nach $l_{n,n}$ auflösen. Hat man aber so erst einmal $l_{n,n} \in \mathbb{C}$ gewonnen, sieht man, daß $l_{n,n}$ sogar reell und positiv ist: Aus
$$0 < \det(A) = \det(L) \cdot \det(L^T) = \det(L') \cdot \det((L')^T) \cdot l_{n,n}^2$$
folgt wegen $\det(L'), \det((L')^T) > 0$ (positive Diagonalelemente!): $l_{n,n}^2 = \alpha > 0$. Die dritte Gleichung besitzt also eine reelle (und eindeutige, da man $l_{n,n} > 0$ fordert) Lösung. Die Matrix L ist dann offensichtlich eine linke Dreiecksmatrix mit positiven Diagonalelementen.

□

Zur praktischen Konstruktion benutzt man die Formel für das Matrixprodukt $A = LL^T$, um die $a_{i,j}$ für $i \geq j$ darzustellen:
$$a_{i,j} = \sum_{k=1}^{n} l_{i,k}l_{j,k} = \sum_{k=1}^{j} l_{i,k}l_{j,k} = \sum_{k=1}^{j-1} l_{i,k}l_{j,k} + l_{i,j}l_{j,j} \text{ für } i \geq j.$$

```
do j=1,n                                ! Bestimme die l_{i,j} spaltenweise
  s = A(j,j)                            ! Zunächst l_{j,j}
  do k=1,j-1
    s = s - L(j,k)*L(j,k)
  end do
  L(j,j) = sqrt(s)
  do i=j+1,n                            ! Jetzt die übrigen l_{i,j} für i > j
    s = A(i,j)
    do k=1,j-1
      s = s - L(i,k)*L(j,k)
    end do
    L(i,j) = s/L(j,j)
  end do
end do
```

Algorithmus 2.4: Cholesky-Zerlegung einer positiv definiten Matrix

Mit einer Fallunterscheidung $i = j$ oder $i > j$ erhalten wir daraus

$$l_{j,j} = \sqrt{a_{j,j} - \sum_{k=1}^{j-1} l_{j,k}^2}, \quad l_{i,j} = \left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} l_{j,k}\right) l_{j,j}^{-1} \text{ für } i > j. \tag{2.8}$$

Dies legt eine „spaltenweise“ Berechnung der $l_{i,j}$ nahe: Für $j = 1$ (erste Spalte) verschwinden die Summen in (2.8), und man erhält $l_{1,1} = \sqrt{a_{1,1}}$, $l_{i,1} = a_{i,1}/l_{1,1}$. Für $j > 1$ kann man direkt (2.8) benutzen, denn die Summen enthalten nur $l_{i,k}$ mit $k < j$, also schon vorher berechnete. Insgesamt liefert (2.8) Algorithmus 2.4:

Bemerkung: In Algorithmus 2.4 wird L in einem Feld `L(i,j)` gespeichert. Man hätte die Speicherung auch „sparsam“ durchführen können, d.h. die $l_{i,j}$ an nicht mehr benötigten Einträgen von A speichern können.

Hat man einmal die Cholesky-Zerlegung berechnet, so kann man $Ax = b$ wegen

$$Ax = b \Leftrightarrow L^T x = y, \text{ wobei } y \text{ eindeutige Lösung von } Ly = b \text{ ist}$$

wie folgt lösen:
1. Schritt: Löse $Ly = b$ nach y
2. Schritt: Löse $L^T x = y$ nach x
Diese Gleichungssysteme lassen sich in $O(n^2)$ Rechenoperationen lösen, da L und L^T Dreiecksgestalt haben.

Für den Rechenaufwand der Zerlegung selber gilt:

Satz 2.4 (Rechenaufwand der Cholesky-Zerlegung) *Die Cholesky-Zerlegung ist mit Algorithmus 2.4 in $n^3/6 + O(n^2)$ Rechenoperationen (plus n Auswertungen von Quadratwurzeln) durchführbar.*

Den Beweis sparen wir uns hier, er läuft analog zu den bisherigen Laufzeitabschätzungen. Als Ergebnis erhalten wir also, daß man um ungefähr einen Faktor 2 schneller ist ($n^3/6$ gegenüber $n^3/3$), wenn man die positive Definitheit eines symmetrischen linearen Gleichungssystems berücksichtigt.

2.4 Einfluß von Rundungsfehlern

Das Gaußverfahren mit Spaltenpivotsuche liefert zu jedem nichtsingulären linearen Gleichungssystem $Ax = b$ die eindeutige Lösung x — theoretisch. In der Praxis, das heißt, wenn man mit Gleitpunktarithmetik rechnet, entstehen Rundungsfehler, die das Ergebnis sehr stark beeinflussen können, wie wir an einem Beispiel gesehen haben. Ziel dieses Abschnitts ist es, einige Resultate zu entwickeln, die es erlauben, die „Güte" einer mit dem Gaußverfahren unter Gleitpunktarithmetik berechneten Lösung abzuschätzen. Wir wollen wissen, wie groß der Fehler im schlimmsten Fall sein kann, und welche Rolle die „Eingangsdaten", also A und b, dabei spielen. Dazu benötigen wir eine *Norm* $\|\cdot\|$, einen verallgemeinerten Absolutbetrag, mit dem man Längen in allgemeinen Vektorräumen wie dem $\mathbb{R}^n$ mißt.

Vektornorm, Matrixnorm und Konditionszahl

Definition 2.1 *Eine Norm $\|\cdot\|$ auf einem reellen Vektorraum V ist eine Abbildung von V nach $\mathbb{R}$ mit folgenden Eigenschaften: Für alle $x, y \in V$ und alle $\lambda \in \mathbb{R}$ gilt:*

1. $\|x\| = 0 \Leftrightarrow x = 0$.
2. $\|\lambda x\| = |\lambda| \cdot \|x\|$.
3. $\|x + y\| \leq \|x\| + \|y\|$ *(Dreiecksungleichung).*

Arbeitet man mit verschiedenen Normen auf V, kennzeichnet man diese meistens durch einen Index. Die gebräuchlichsten Normen im $\mathbb{R}^n$ sind die 1–, 2– und ∞–Normen

$$\|x\|_1 = \sum_{i=1}^n |x_i|, \quad \|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}, \quad \|x\|_\infty = \max_i |x_i|,$$

welche alle Spezialfälle der p-Norm

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{\frac{1}{p}},$$

$1 \leq p < \infty$, sind (man kann zeigen, daß in einem gewissen Sinne $||\cdot||_p \to ||\cdot||_\infty$ für $p \to \infty$ gilt). Die 2–Norm ist die aus der Geometrie bekannte *euklidische Norm.* Die ∞–Norm nenn man auch *Maximumsnorm* oder *Supremumsnorm.*

Da man $n \times n$–Matrizen einfach als „große Vektoren“ betrachten kann, also als Elemente des Vektorraums $\mathbb{R}^{n^2}$, ist es klar, daß man auch für Matrizen Normen hat. Die 2-Norm einer Matrix wäre somit definiert als die Wurzel aus der Summe der Quadrate aller n^2 Matrixkomponenten. Allerdings sind diese Normen nicht so gebräuchlich. Hat man auf $\mathbb{R}^n$ eine Vektornorm, so betrachtet man vielmehr die sogenannte *von der Vektornorm induzierte Matrixnorm* oder einfacher *die zu der Vektornorm gehörende Matrixnorm*

$$||A|| = \sup_{x \in V, x \neq 0} \frac{||Ax||}{||x||}. \tag{2.9}$$

Man beachte, daß für die Vektornorm und ihre zugehörige Matrixnorm dasselbe Symbol $||\cdot||$ verwendet wird. Verwechselungen sind aber dennoch nicht zu befürchten, wenn man weiß, um welche Objekte es sich handelt, um deren Norm es geht. So ist mit $||A||$ die Matrixnorm gemeint, weil A eine Matrix ist, und mit $||Ax||$ und $||x||$ die Vektornorm, weil Ax und x Vektoren sind.

Zwei einfache Abschätzungen, die sehr oft gebraucht werden, sind

Satz 2.5 *Es seien $A, B \in \mathbb{R}^{n,n}$ und $x \in \mathbb{R}^n$. Es gilt*

1. $||Ax|| \leq ||A|| \cdot ||x||$

2. $||AB|| \leq ||A|| \cdot ||B||$

Beweis:

1. Für $x = 0$ ist die Behauptung klar. Für $x \neq 0$ ist nach Definition der Matrixnorm $||A|| \geq ||Ax||/||x||$.

2. Es gilt

$$||AB|| = \sup_{x \neq 0} \frac{||ABx||}{||x||} = \sup_{x \neq 0} \frac{||A(Bx)||}{||Bx||} \frac{||Bx||}{||x||} \leq \sup_{y \neq 0} \frac{||Ay||}{||y||} \sup_{x \neq 0} \frac{||Bx||}{||x||} = ||A|| \cdot ||B||.$$

□

Zur praktischen Berechnung einer konkreten Matrixnorm ist (2.9) eher ungeeignet. Für die 1–, 2– und ∞–Matrixnormen gibt es folgende Darstellungen, deren Beweis hier nicht geführt werden soll:

Satz 2.6 1. $||A||_1 = \max_{j=1,\dots,n} \sum_{i=1}^n |a_{i,j}|$ *(Spaltensummennorm)*

2. $\|A\|_2 = \sqrt{|\lambda_{max}(A^T A)|}$, *wobei* $\lambda_{max}(M)$ *für eine quadratische Matrix* M *definiert ist als der Eigenwert von* M *mit dem größten Betrag*

3. $\|A\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^n |a_{i,j}|$ *(Zeilensummennorm)*

Bemerkung:

1. Für die bisher eingeführten Normen muß überprüft werden, ob sie tatsächlich die drei Grundeigenschaften einer Norm erfüllen. Wir führen diese Überprüfung hier nicht durch.

2. Die Eigenschaften der induzierten Matrixnorm in Lemma 2.5 gelten im allgemeinen nicht für beliebige Matrixnormen. Beim Beweis wurden im Wesentlichen (2.9) benutzt.

3. Die 1– und ∞–Matrixnormen in ihrer obigen Darstellung kann man leicht verwechseln. Man kann sich mit folgender Eselsbrücke behelfen: Die „1“ ist ein „stehendes“ Symbol, dementsprechend wird die Summe über die Spalten (die nebeneinander in der Matrix „stehen“) genommen. Demgegenüber ist „∞“ ein „liegendes Symbol“, und die Summe wird über die Zeilen (die übereinander in der Matrix „liegen“) genommen.

Bevor wir nun eine Abschätzung für den Fehler geben, benötigen wir noch den Begriff der *Konditionszahl*:

Definition 2.2 *Die Konditionszahl einer nichtsingulären Matrix* A *ist definiert als* $cond(A) = \|A\| \cdot \|A^{-1}\|$.

Man beachte, daß die Konditionszahl von der verwendeten Matrixnorm abhängt. Bezieht sie sich z.B auf die euklidische Norm, so schreibt man oft $\text{cond}_2(A)$.
Eine wichtige Eigenschaft der Konditionszahl ist

Satz 2.7 $cond(A) \geq 1$.

Beweis: Für $x \in \mathbb{R}^n$ gilt: $\|x\| = \|AA^{-1}x\| \leq \|A\| \cdot \|A^{-1}\| \cdot \|x\|$. □

Fehlerabschätzung

Eine erste Möglichkeit zur Beurteilung einer numerischen Lösung $\bar{x}$ des linearen Gleichungssystems $Ax = b$ ist, das *Residuum* $r := b - A\bar{x}$ zu berechnen. r gibt an, wie gut die berechnete Lösung das Gleichungssystem erfüllt. Allerdings ist ein kleines Residuum allein noch keine Garantie für einen kleinen Fehler:

Satz 2.8 (Fehlerabschätzung) *Es sei $A \in \mathbb{R}^{n,n}$ nichtsingulär, $0 \neq b \in \mathbb{R}^n$, $0 \neq x \in \mathbb{R}^n$ die eindeutige Lösung des linearen Gleichungssystems $(A|b)$, $\bar{x}$ eine numerische Lösung und $r = b - A\bar{x}$ das Residuum. Dann gilt für den relativen Fehler:*

$$\frac{||x - \bar{x}||}{||x||} \leq cond(A) \frac{||r||}{||b||}. \tag{2.10}$$

Beweis:

$$r = b - A\bar{x} = Ax - A\bar{x} \Rightarrow x - \bar{x} = A^{-1}r \Rightarrow ||x - \bar{x}|| \leq ||A^{-1}|| \cdot ||r||.$$
$$b = Ax \Rightarrow ||b|| \leq ||A|| \cdot ||x|| \Rightarrow \frac{1}{||x||} \leq \frac{||A||}{||b||}.$$

Insgesamt folgt die Behauptung. □

Wir sehen, daß die obere Schranke für den relativen Fehler von $\bar{x}$ aus den Faktoren $||r||/||b||$ (das „relative Residuum") und cond(A) besteht. Die Fehlerabschätzung sagt also aus, daß der relative Fehler um den Faktor cond(A) größer sein kann als das relative Residuum. Das bedeutet, daß bei großen Konditionszahlen der relative Fehler selbst dann groß sein kann, wenn das Residuum klein ist. Matrizen mit großer Konditionszahl heißen demnach *schlechtkonditioniert.* Wir wollen an einem Beispiel demonstrieren, wie dramatisch sich tatsächlich eine schlechte Konditionszahl auf den Fehler auswirken kann.

Hilbertmatrizen

Die n-te *Hilbertmatrix* $H_n \in \mathbb{R}^{n,n}$ ist definiert durch $h_{i,j} = 1/(i + j - 1)$. Hilbertmatrizen sind das berüchtigtste Beispiel für schlechtkonditionierte Matrizen. Man kann zeigen, daß cond$_2(H_n)$ exponentiell mit n wächst, so daß schon z.B. cond$_2(H_{10}) \simeq 1.6 \cdot 10^{13}$.

Wir lösen für $n = 10$ numerisch das System $(H_n|b)$, wobei b so gewählt wird, daß

$$x = (1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0)^T$$

Lösung ist. Bei einfacher Genauigkeit erhalten wir mit dem Gaußverfahren mit Spaltenpivotsuche aber die numerische Lösung (hier nur auf 4 Stellen genau angegeben)

$$\bar{x} = (1.001,\ 0.9269,\ 1.783,\text{- } 2.412,\ 7.901,\ \text{-}4.811,\ 0.7891,\ 3.917,\ 0.1244,\ 0.7788)^T,$$

die so ungenau ist, daß man wahrscheinlich durch grobes Ausprobieren eine bessere Lösung bekommen hätte — und das, obwohl alle Komponenten des Residuums r betragsmäßig $< 10^{-7}$ sind! Man rechnet aus, daß in der 2-Norm

$$\frac{||r||}{||b||} \simeq 3.07 \cdot 10^{-8}, \quad \frac{||x - \bar{x}||}{||x||} \simeq 3.21,$$

der relative Fehler also mehr als 10^8 mal so groß ist wie das (relative) Residuum!