Kapitel 2

Numerische Methoden zur Lösung linearer, inhomogener Gleichungssysteme

2.1 Das grundsätzliche Problem

Gegeben sind die n Gleichungen

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{2.1}$$

wobei die a_{ij} und die b_i reellwertige Größen sind. Außerdem soll gelten:

$$\mid b_1 \mid + \mid b_2 \mid + \cdots + \mid b_n \mid \neq 0$$

Unter diesen Voraussetzungen stellt (2.1) ein *reellwertiges, lineares, inhomogenes Gleichungssystem n-ter Ordnung* dar. Die Größen $x_1, \ldots, x_n$, welche die gegebenen Gleichungen simultan erfüllen, nennt man die *Lösungen* des Systems.

(2.1) wird gewöhnlich in Form der Matrixgleichung

$$A \cdot \mathbf{x} = \mathbf{b} \tag{2.2}$$

angeschrieben.
Die Lösung derartiger Systeme stellt ein zentrales Problem der numerischen Mathematik dar, weil eine ganze Reihe von numerischen Verfahren, wie z.B. die Interpolationsverfahren, die Least-Squares-Verfahren, die Differenzenverfahren usw. auf die Lösung inhomogener linearer Gleichungssysteme zurückgeführt werden können.

Vom theoretischen Standpunkt aus bereitet die Lösung von (2.1) keinerlei Schwierigkeiten, solange die Determinante der *Koeffizientenmatrix A* nicht verschwindet, solange also das Problem nicht *singulär* ist:

$$\det(A) \neq 0$$

In diesem Fall kann das Problem mit Hilfe der *Cramer'schen Regel* angegangen werden. In der Praxis zeigt sich jedoch, daß die Anwendung dieser Regel für $n \geq 4$ recht kompliziert und aus verschiedenen Gründen für die Anwendung am Computer ungeeignet ist.

Wie auch in anderen Bereichen der numerischen Mathematik gibt es auch hier zwei Gruppen von Methoden:

- **Direkte Methoden:**

  Diese enthalten keine Verfahrensfehler und führen deshalb - abgesehen von Rundungsfehler-Einflüssen - stets zur exakten Lösung. Allerdings können Rundungsfehler gerade bei direkten Verfahren, die häufig sehr rechenintensive Algorithmen aufweisen, massiv in Erscheinung treten.

  Als Beispiel dazu wird im folgenden das *Eliminationsverfahren von Gauss* in der Formulierung von *Doolittle und Crout (LU decomposition)* behandelt.

- **Iterative Methoden**

  zeichnen sich oft durch einen besonders einfachen Rundungsfehler-stabilen und Speicherplatz-sparenden Algorithmus aus, sie haben jedoch den Nachteil, daß sie nicht in jedem Fall konvergieren, also nicht auf alle an sich lösbaren linearen Gleichungssysteme angewendet werden können.

  Außerdem erhält man auch im Falle der Konvergenz nie die exakte Problemlösung, sondern nur eine mit einem Abbruchfehler behaftete Näherungslösung.

  Als Beispiel für ein Iterationsverfahren zur Lösung linearer Gleichungssysteme steht in diesem Kapitel das *Gauss-Seidel-Verfahren.*

2.2 Ziel der direkten Methoden: Überführung der Koeffizientenmatrix in eine obere Dreiecksmatrix.

Bei allen direkten Methoden geht es letzten Endes darum, das gegebene System

$$A \cdot \mathbf{x} = \mathbf{b}$$

in ein äquivalentes (d.h. denselben Lösungsvektor aufweisendes) System

$$U \cdot \mathbf{x} = \mathbf{y} \tag{2.3}$$

überzuführen, wobei U eine sogenannte *obere Dreiecksmatrix* mit der Eigenschaft

$$U = [u_{ij}] \qquad \text{mit} \qquad u_{ij} = 0 \qquad \text{für} \qquad i > j$$

darstellt. Der Grund dafür ist einfach: Systeme der Art (2.3) sind durch *Rück-Substitution* (s. Glg.en (2.12)und (2.13)) leicht aufzulösen.

2.3 Das Eliminationsverfahren von Gauss in der Formulierung von Doolittle und Crout (LU-Aufspaltung).

Die *Gauss'sche Elimination* besteht im wesentlichen aus zwei Schritten, nämlich aus der Überführung von $A\mathbf{x} = \mathbf{b}$ in das äquivalente System $U\mathbf{x} = \mathbf{y}$ und der Lösung dieses Systems mittels einer Rück-Substitution.

Das Verfahren beruht auf einem Satz aus der linearen Algebra:
Ein lineares Gleichungssystem bleibt unverändert, wenn zu einer seiner Gleichungen eine Linearkombination der restlichen Zeilen addiert wird.

Doolittle und Crout haben nun gezeigt, daß der Gauss'sche Algorithmus wie folgt formuliert werden kann: Eine reelle Matrix A kann stets als Produkt zweier reeller Matrizen L und U dargestellt werden, also

$$A = L \cdot U \tag{2.4}$$

wobei U eine *obere* Dreieckmatrix ist:

$$U = \begin{pmatrix} u_{11} & u_{12} & ..... & u_{1n} \\ 0 & u_{22} & ..... & u_{1n} \\ . & & & . \\ . & & & . \\ . & & & . \\ 0 & 0 & ..... & u_{nn} \end{pmatrix}$$

Die Matrix L stellt eine *untere* Dreiecksmatrix der Form

$$L = \begin{pmatrix} 1 & 0 & 0 & ..... & 0 \\ m_{21} & 1 & ..... & 0 & \\ m_{31} & m_{32} & 1 & ..... & 0 \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ m_{n1} & m_{n2} & m_{n3} & ..... & 1 \end{pmatrix}$$

dar.

Man nennt die Zerlegung (2.4) einer Matrix A die *LU- Zerlegung (LU decomposition)* von A.

Ohne Ableitung: ein einfacher Formelapparat für die LU-Zerlegung.

Die Umformung erfolgt spaltenweise, also $j = 1, 2, \ldots, n$:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} m_{ik} u_{kj} \qquad i = 1, \ldots, j-1 \tag{2.5}$$

$$\gamma_{ij} = a_{ij} - \sum_{k=1}^{j-1} m_{ik} u_{kj} \qquad i = j, \ldots, n \tag{2.6}$$

$$u_{jj} = \gamma_{jj} \tag{2.7}$$

$$m_{ij} = \frac{\gamma_{ij}}{\gamma_{jj}} \qquad i = j+1, \ldots, n \tag{2.8}$$

Da die Hauptdiagonalelemente von L stets Einser sind, braucht man diese Werte (die m_{jj}) nicht extra abzuspeichern, sondern man kann alle relevanten Werte in einer n x n-Matrix (der 'LU-Matrix') unterbringen:

$$\text{LU-Matrix} : \begin{pmatrix} u_{11} & u_{12} & u_{13} & ..... & u_{1n} \\ m_{21} & u_{22} & u_{23} & ..... & u_{2n} \\ m_{31} & m_{32} & u_{33} & ..... & u_{3n} \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ m_{n1} & m_{n2} & m_{n3} & ..... & u_{nn} \end{pmatrix} \tag{2.9}$$

Die Aufspaltung der Matrix A in die Matrizen L und U kann nun zur Berechnung des Lösungsvektors herangezogen werden:

$$A \cdot \mathbf{x} \equiv L \cdot U \cdot \mathbf{x} = L \cdot (U \cdot \mathbf{x}) = \mathbf{b}$$

Mit $U \cdot \mathbf{x} \equiv \mathbf{y}$ erhält man das System $L \cdot \mathbf{y} = \mathbf{b}$, das wegen der besonderen Form der Matrix L (untere Dreiecksmatrix) mittels einer *Vorwärts-Substitution* gelöst werden kann:

$$y_1 = b_1 \tag{2.10}$$

$$y_i = b_i - \sum_{j=1}^{i-1} m_{ij} y_j \qquad i = 2, 3, \ldots, n \tag{2.11}$$

Da U eine obere Dreiecksmatrix ist, ist bei Kenntnis des Hilfsvektors $\mathbf{y}$ das System $U \cdot \mathbf{x} = \mathbf{y}$ durch *Rückwärts-Substitution* ebenfalls leicht zu lösen:

$$x_n = \frac{y_n}{u_{nn}} \tag{2.12}$$

$$x_i = \frac{1}{u_{ii}} \left[y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right] \qquad i = n-1, n-2, \ldots, 1 \tag{2.13}$$

Bei der konkreten Anwendung des Doolittle-Crout-Verfahrens geht man am besten so vor, daß man zwei Programme verwendet, von denen eines die LU-Zerlegung der Koeffizientenmatrix durchführt, während das zweite den Lösungsvektor unter Verwendung der Formeln (2.10-2.13) berechnet.

2.3.1 Demonstration einer speicherplatz-sparenden LU-Decomposition

an Hand einer allgemeinen 3x3-Matrix.

Gegeben:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

j = 1: erste Spalte

(2.6) i=1 $\gamma_{11} = a_{11}$
i=2 $\gamma_{21} = a_{21}$
i=3 $\gamma_{31} = a_{31}$

.
.

(2.7) $u_{11} = \gamma_{11}$

.
.

(2.8) i=2 $m_{21} = \gamma_{21}/\gamma_{11}$
i=3 $m_{31} = \gamma_{31}/\gamma_{11}$

Speichert man nun die eben berechneten Koeffizienten u_{11}, m_{21} und m_{31} *anstelle der entsprechenden und nie wieder gebrauchten Koeffizienten der Matrix A ab*, so ergibt sich die Matrix

$$\begin{pmatrix} u_{11} & a_{12} & a_{13} \\ m_{21} & a_{22} & a_{23} \\ m_{31} & a_{32} & a_{33} \end{pmatrix}$$

j=2: zweite Spalte

(2.5) i=1 $u_{12} = a_{12}$

.
.

(2.6) i=2 $\gamma_{22} = a_{22} - m_{21}u_{12}$
i=3 $\gamma_{32} = a_{32} - m_{31}u_{12}$

.
.

(2.7) $u_{22} = \gamma_{22}$

.
.

(2.8) i=3 $m_{32} = \gamma_{32}/\gamma_{22}$

$$\begin{pmatrix} u_{11} & u_{12} & a_{13} \\ m_{21} & u_{22} & a_{23} \\ m_{31} & m_{32} & a_{33} \end{pmatrix}$$

j=3: dritte Spalte

$$
\begin{array}{lll}
(2.5) & \text{i=1} & u_{13} = a_{13} \\
 & \text{i=2} & u_{23} = a_{23} - m_{21}u_{13} \\
 & . & \\
 & . & \\
(2.6) & \text{i=3} & \gamma_{33} = a_{33} - m_{31}u_{13} - m_{32}u_{23} \\
 & . & \\
 & . & \\
(2.7) & & u_{33} = \gamma_{33}
\end{array}
$$

$$
\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ m_{21} & u_{22} & u_{23} \\ m_{31} & m_{32} & u_{33} \end{pmatrix} = \text{LU-Matrix} \quad \text{[s. Glg.(2.9)]}
$$

Entscheidend ist also, daß durch die sukzessive Überspeicherung der gegebenen Koeffizientenmatrix A durch die Spalte für Spalte entstehende LU-Matrix *zu keiner Zeit ein Informationsmanko entsteht!*

Da andererseits auch die neuen u_{ij}- bzw. m_{ij}-Werte stets verschiedene Speicherplätze einnehmen, *kann im Programm auf eine unterschiedliche Bezeichnung der Elemente a_{ij}, u_{ij}, γ_{ij} und m_{ij} überhaupt verzichtet werden, und alle diese Elemente werden im Struktogramm 3 (LUDCMP) auf einem Feld A(,) abgespeichert!*
Man spart damit sehr viel Speicherplatz, allerdings auf Kosten der Lesbarkeit des Programms.

2.3.2 Rundungsfehler-Optimierung durch partielle Pivotisierung.

Angenommen, es existiert ein Programm, das die Auswertung eines inhomogenen Gleichungssystems nach der LU-Zerlegung durchführt. Die mittels dieses Programmes erhaltenen numerischen Ergebnisse enthalten keine Verfahrensfehler (direktes Verfahren!); jede Abweichung von den 'wahren' Ergebnissen muß daher auf Rundungsfehler zurückgehen.

Dazu ein konkretes Beispiel: Gegeben sei das folgende System von 3 Gleichungen:

$$
\begin{aligned}
x_1 + 5923181x_2 + 1608x_3 &= 5924790 \\
5923181x_1 + 337116x_2 - 7x_3 &= 6260290 \\
6114x_1 + 2x_2 + 9101372x_3 &= 9107488
\end{aligned}
\tag{2.14}
$$

Dieses System hat die exakte Lösung $x_1 = x_2 = x_3 = 1$.

Die numerische Auswertung (FORTAN, einfache Genauigkeit, d.h. 4 Byte pro reeller Zahl) liefert die folgende LU-Matrix und den folgenden Lösungsvektor:

	0.1000000E+01	0.5923181E+07	0.1608000E+04
LU-Matrix:	0.5923181E+07	-.3508407E+14	-.9524475E+10
	0.6114000E+04	0.1032216E-02	0.9101371E+07

	0.9398398E+00
x:	0.1000000E+01
	0.9995983E+00

Wie man sieht, ist der Rundungsfehler-Einfluß (besonders beim x_1) beträchtlich!

Wenn man nun die Reihenfolge der Gleichungen in (2.14) abändert (was theoretisch natürlich keinen Einfluß auf den Lösungsvektor hat), und zwar so, daß die ursprüngliche Reihenfolge 1/2/3 übergeht in die Reihenfolge 2/3/1, so ergibt dasselbe Programm:

	0.5923181E+07	0.3371160E+06	-.7000000E+01
LU-Matrix:	0.1032216E-02	-.3459764E+03	0.9101372E+07
	0.1688282E-06	-.1712019E+05	0.1558172E+12

mit

	0.9999961E+00
x:	0.1000068E+01
	0.1000000E+01

bereits wesentlich kleineren Rundungsfehlern. Mit der Zeilenfolge 2/1/3 erhält man den korrekten Lösungsvektor in Maschinengenauigkeit:

	0.5923181E+07	0.3371160E+06	-.7000000E+01
LU-Matrix:	0.1688282E-06	0.5923181E+07	0.1608000E+04
	0.1032216E-02	-.5841058E-04	0.9101372E+07

	0.1000000E+01
x:	0.1000000E+01
	0.1000000E+01

Das bedeutet also, daß die richtige Zeilenfolge eines linearen Gleichungssystems einen entscheidenden Einfluß auf die Größe der Rundungsfehler und somit auf die Qualität der numerischen Lösung hat!

Was bedeutet nun 'richtige' Reihenfolge? Vergleicht man die m_{ij}-Werte in den LU-Matrizen, so erkennt man sofort, daß die Rundungsfehler dann am geringsten sind, *wenn die Beträge der m_{ij}-Werte möglichst klein sind.* Um die m_{ij} so klein als möglich zu halten, müssen die in der Gleichung (2.8) auftretenden γ_{jj}-Werte dem Betrage nach möglichst groß sein. Dies kann einfach dadurch erreicht werden, daß man aus allen γ_{ij} nach Glg. (2.6) das betragsgrößte Element bestimmt und jene Zeile, in welcher dieses Element vorkommt, mit der j-ten Zeile vertauscht. Erst danach werden die Gleichungen (2.7) und (2.8) ausgewertet.
Eine derartige Strategie nennt man *eine LU-Zerlegung mit teilweiser Pivotisierung.* In der Regel hat man – vor allem bei Systemen höherer Ordnung – ohne Pivotisierung keine Chance auf korrekte Ergebnisse. Es gibt aber auch (s. [2], S. 56) Gleichungssysteme, bei welchen die LU-Decomposition auch ohne Pivotisierung numerisch stabil ist, nämlich dann, wenn die Koeffizientenmatrix *symmetrisch, tridiagonal, zyklisch tridiagonal, diagonal-dominant*

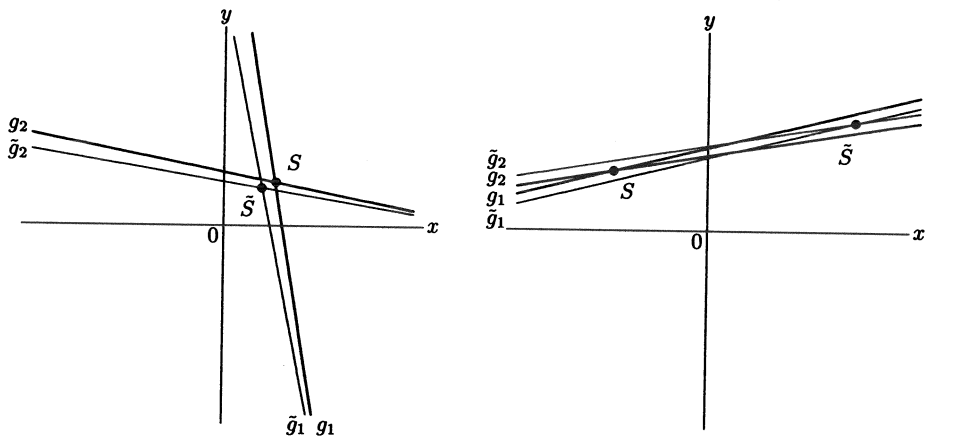


Abbildung 2.1: Gut und schlecht konditioniertes 2x2-System (aus: [22], S. 228).

bzw. allgemein positiv definit ist.

Es ist klar, daß mit der Strategie der teilweisen Pivotisierung auch das Nullwerden eines Diagonalelementes γ_{jj} und die damit auftretenden Probleme in Glg.(2.8) vermieden werden können. Stellt sich nämlich heraus, daß während der Rechnung *alle* $\gamma_{ij}, \quad i = j, \ldots, n$ *simultan* verschwinden, ist die Koeffizientenmatrix *singulär* und das Gleichungssystem mit dem gegebenen Verfahren nicht lösbar!

2.3.3 Kondition eines Gleichungssystems.

Die mit Hilfe direkter Methoden ermittelte Lösung eines linearen Gleichungssystems ist meist nicht die exakte Lösung, weil

- im Verlauf der Rechnung Rundungsfehler auftreten, deren Akkumulation zur Verfälschung der Ergebnisse führen kann.
- Ungenauigkeiten in den Ausgangsgrößen bestehen können, die Ungenauigkeiten in den Lösungen hervorrufen.

Wenn kleine Änderungen in den Ausgangsdaten große Änderungen in der Lösung hervorrufen, spricht man von einem *schlecht konditionierten* System (s. Abb. 2.1).
Die sog. *Konditionszahl* der Koeffizientenmatrix kann dabei als Maß für die Güte der erhaltenen Näherungslösung für den Lösungsvektor verwendet werden.

In der mathematischen Literatur wird eine Reihe von Konditionszahlen diskutiert. Im folgenden Programm LUDCMP ist die Berechnung der sog. *Hadamarschen Konditionszahl* der Matrix A inkludiert, welche in der Form [2]

$$K_{\mathrm{H}}(A) = \frac{|det(A)|}{\alpha_1 \alpha_2 \cdots \alpha_n} \qquad \text{mit} \qquad \alpha_i = \sqrt{a_{i1}^2 + a_{i2}^2 + \ldots + a_{in}^2}$$

gegeben ist. Für die Praxis gelten die Erfahrungswerte

- $K_{\mathrm{H}}(A) < 0.01$ schlechte Kondition,
- $K_{\mathrm{H}}(A) > 0.1$ gute Kondition,
- $0.01 \leq K_{\mathrm{H}}(A) \leq 0.1$ keine Aussage.

2.3.4 Das Programm LUDCMP

Quelle: [9], S.38f; [10], S. 46f (mit geringen Änderungen).

Das Programm LUDCMP (LU DeCoMPosition) führt eine LU-Aufspaltung einer reellen quadratischen Matrix durch.

INPUT Parameter:

A(,): Koeffizientenmatrix des linearen Systems.

N: Ordnung des Systems = Anzahl der Zeilen und Spalten von A.

OUTPUT Parameter:

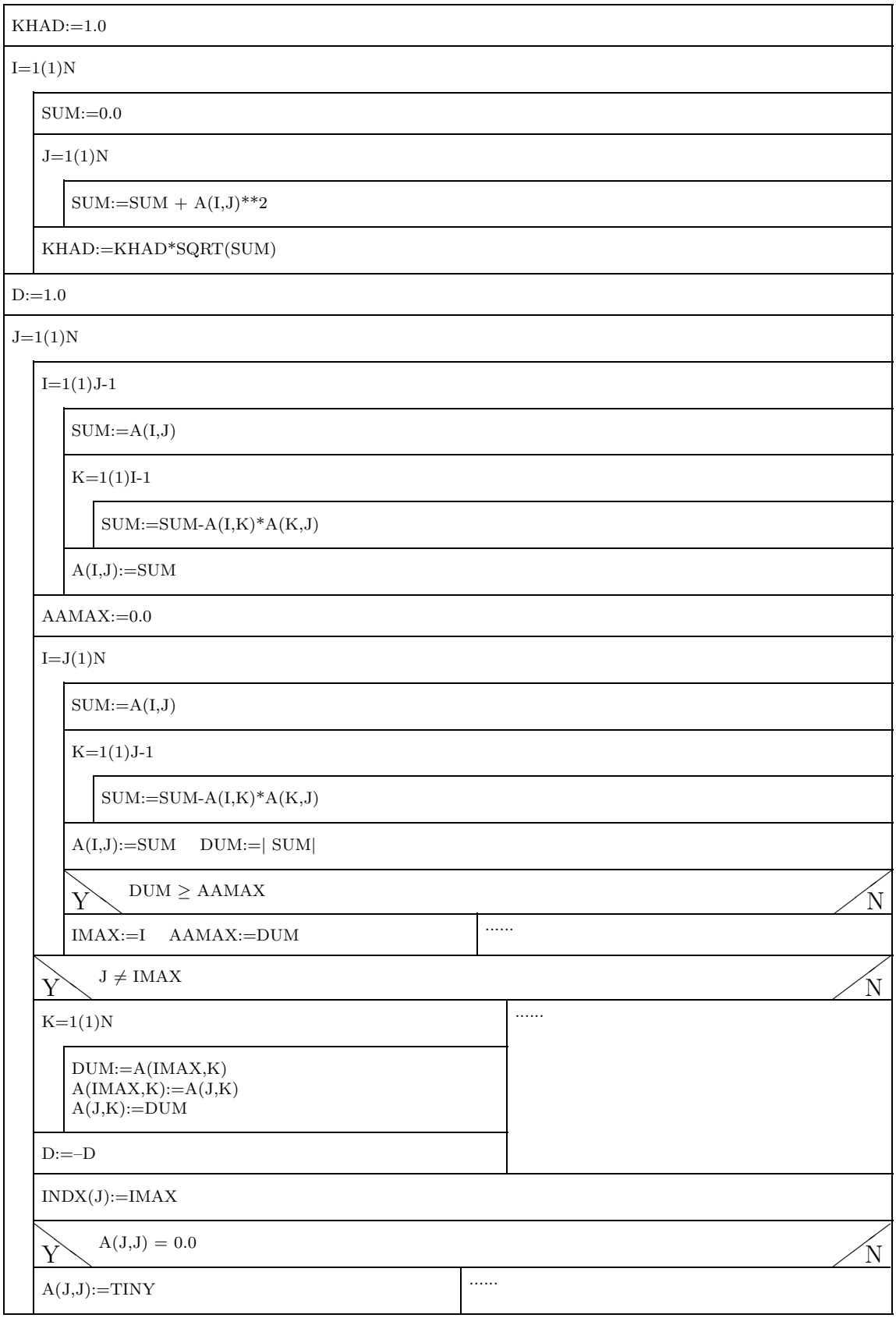
A(,): 'LU Matrix' der ursprünglichen Matrix A (s. Anmerkung 1).

INDX( ): Indexvektor, der die Zeilenvertauschungen auf Grund der partiellen Pivotisierung speichert.

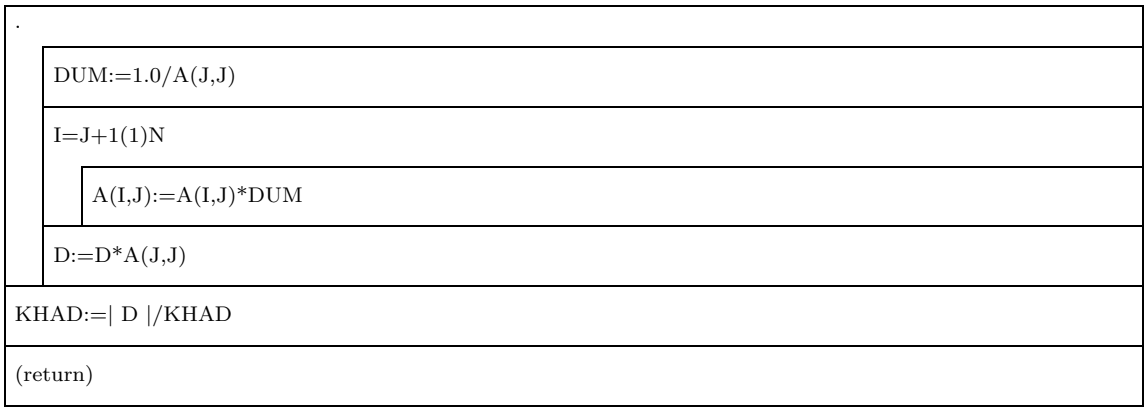
D: Determinante der Koeffizientenmatrix (s. Abschnitt 2.3.6).

KHAD: Hadamardsche Konditionszahl (s. Abschnitt 2.3.2).

Struktogramm 3 — LUDCMP(A,N,INDX,D,KHAD)



. — .



Anmerkungen zum Programm LUDCMP:

1. Um Speicherplatz zu sparen, werden die Koeffizienten der LU Matrix sukzessive auf den entsprechenden Speicherplätzen der gegebenen Matrix A abgespeichert. Dies hat allerdings den Nachteil, daß die gegebene Matrix zerstört wird und daher gegebenenfalls (z.B. wenn A für spätere Rechnungen wieder gebraucht wird) **vor** Aufruf von LUDCMP gesichert werden muß.

2. Der Output-Parameter D hat Bedeutung bei der Berechnung der Determinante von A.

3. Die Konstante TINY ist in [9] mit 10^{-20} definiert. Wie aus dem Struktogramm 3 hervorgeht, verhindert sie ein crash-down des Programmes, wenn ein Divisor in Glg.(2.8) exakt Null wird. Zu diesem Thema siehe Kap.2.5.

4. In [9] werden die Matrixkoeffizienten für die Pivotsuche zeilenweise skaliert. Da optimale Skalierungen von linearen Systemen ein überraschend schwieriges Problem darstellen (s. z.B. [7], S.140ff und [12], S.41ff), wurde diese Skalierung aus dem hier präsentierten Programm herausgenommen.

2.3.5 Das Unterprogramm LUBKSB

Quelle: [9], S. 39; [10], S. 47 (mit Änderungen).

Das Programm LUBKSB (LU BacKSuBstitution) berechnet den Lösungsvektor $\mathbf{x}$ des Systems $LU\mathbf{x} = \mathbf{b}$ unter Verwendung der Gleichungen (2.10 - 2.13).

INPUT-Parameter:

A(,): LU-Matrix der Koeffizientenmatrix, wie sie z.B. vom Programm LUDCMP berechnet wird.

N: Ordnung des Systems = Zeilen und Spalten von A.

INDX( ): Indexvektor, der Informationen über die von LUDCMP vorgenommenen Zeilenvertauschungen enthält.

B( ): inhomogener Vektor des Systems.

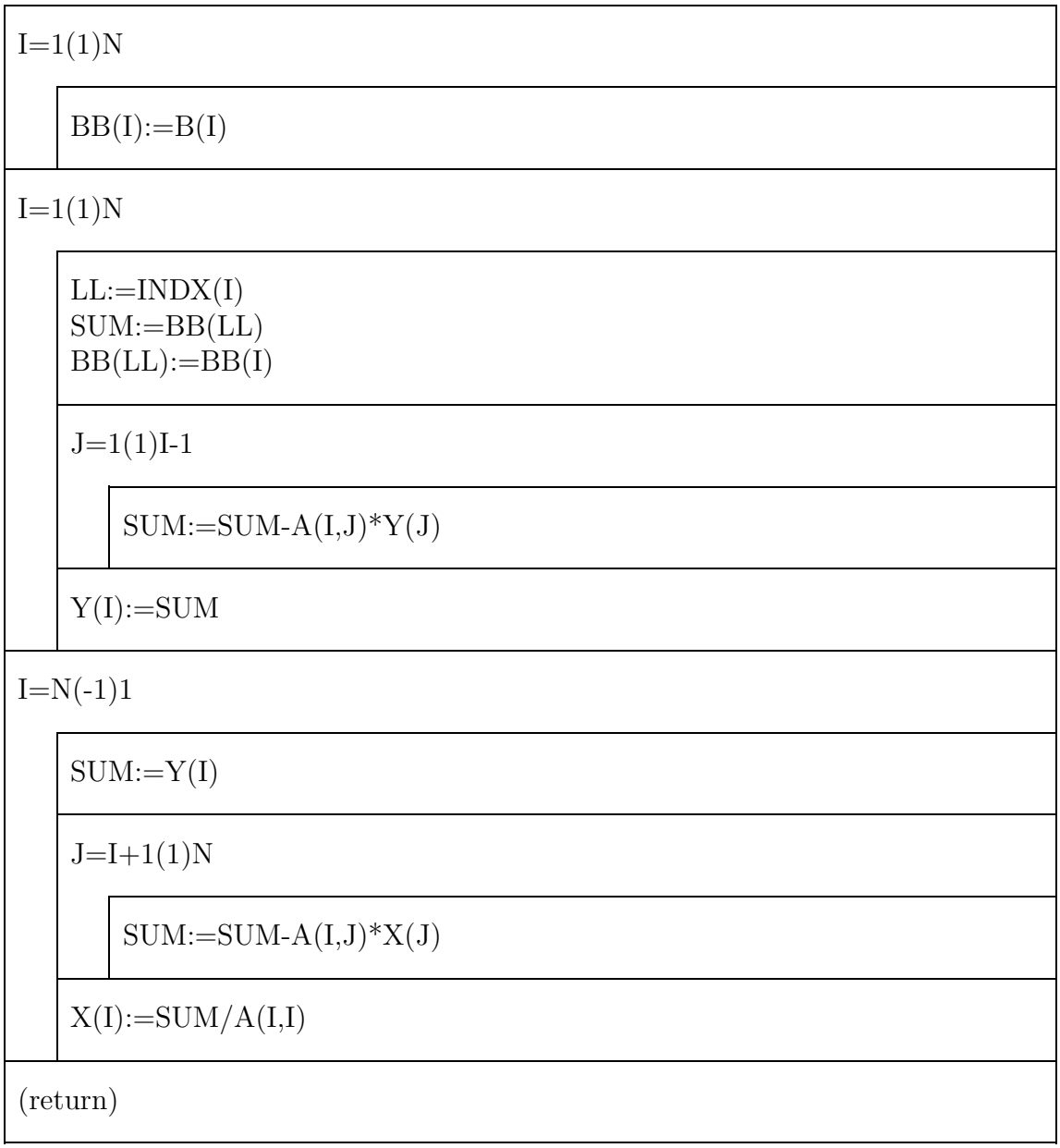
OUTPUT Parameter:

X( ): Lösungsvektor.

Interner Vektor:

Y( ): .

Struktogramm 4 — LUBKSB(A,N,INDX,B,X)



2.3.6 Die Verwendungsmöglichkeiten für die Programme LUDCMP und LUBKSB.

Lösung eines inhomogenen Gleichungssystems $A\mathbf{x} = \mathbf{b}$:

```
||--------------------------||
||  LUDCMP(A,N,INDX,D,KHAD)  ||
||--------------------------||
||   LUBKSB(A,N,INDX,B,X)    ||
||--------------------------||
```

Ein wesentlicher Vorteil des hier vorgestellten Verfahrens besteht darin, daß bei Auftreten verschiedener inhomogener Vektoren $\mathbf{b}_1$, $\mathbf{b}_2$, ..., aber gleichbleibender Koeffizientenmatrix A die LU-Zerlegung nur einmal durchgeführt werden muß:

```
||-------------------------||
|| LUDCMP(A,N,INDX,D,KHAD) ||
||-------------------------||
             |
||-------------------------||
|| LUBKSB(A,N,INDX,B1,X1)  ||
|| LUBKSB(A,N,INDX,B2,X2)  ||
||           .             ||
||           .             ||
||-------------------------||
```

Invertierung einer Matrix:

Mit Hilfe der Programme LUDCMP und LUBKSB kann eine gegebene Matrix A *Spalte für Spalte* invertiert werden:

$$A \cdot X = E \qquad E = \text{Einheitsmatrix} \qquad X \equiv A^{-1}$$

Diese Matrixgleichung kann man nun in eine Reihe von inhomogenen linearen Gleichungssystemen der Art

$$A \begin{pmatrix} x_{11} \\ . \\ . \\ . \\ x_{n1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ . \\ . \\ 0 \end{pmatrix} \quad \cdots \quad A \begin{pmatrix} x_{1n} \\ . \\ . \\ . \\ x_{nn} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ . \\ . \\ 1 \end{pmatrix}$$

zerlegen, d.h. man hat n Systeme mit gleicher Koeffizientenmatrix A und mit wechselnden inhomogenen Vektoren (= Spaltenvektoren der Einheitsmatrix) zu berechnen:

Struktogramm — Invertierung einer Matrix

LUDCMP (A,N,INDX,D,KHAD)		
J=1(1)N		
	I=1(1)N	
		VEKTOR(I):=0.0
	VEKTOR(J):=1.0	
	LUBKSB (A,N,INDX,VEKTOR,X)	
	I=1(1)N	
		AINV(I,J):=X(I)

Berechnung der Determinante einer Matrix:

Für die Determinante der Matrix A gilt

$$det(A) = det(L \cdot U) = det(L)\, det(U)\,.$$

Weil beide Matrizen L und U Dreiecksmatrizen sind, sind deren Determinanten gleich den jeweiligen Produkten der Diagonalelemente. Weil aber alle Diagonalelemente der L-Matrix den Wert 1 haben, gilt

$$det(L) = 1\,,$$

und es ergibt sich

$$det(A) = det(U) = \sum_{i=1}^{n} u_{ii}\,.$$

Die Berechnung von $det(U)$ erfolgt während der LU-Dekomposition im Programm LUDCMP; dabei ist zu berücksichtigen, daß bei jeder Zeilenvertauschung sich das Vorzeichen der Determinante ändert.

2.3.7 Testbeispiele für die Programme LUDCMP und LUBKSB.

Gegeben ist die Matrix mit vier Zeilen und Spalten

$$A = \begin{pmatrix} 1.1161 & 0.1254 & 0.1397 & 0.1490 \\ 0.1582 & 1.1675 & 0.1768 & 0.1871 \\ 0.1968 & 0.2071 & 1.2168 & 0.2271 \\ 0.2368 & 0.2471 & 0.2568 & 1.2671 \end{pmatrix}$$

Die Konditionszahl lautet $K_{\mathrm{H}} = 0.752$, das System ist also *gut konditioniert!*

- Wie lautet der Lösungsvektor $\mathbf{x}$ des linearen inhomogenen Gleichungssystems

$$A \cdot \mathbf{x} = (-1.8367, 1.1944, 3.2368, -0.7232)^T \quad ?$$

  Das numerische Ergebnis ist bis auf Maschinengenauigkeit exakt:

  $\mathbf{x} = (-.2000000E{+}01, 0.1000000E{+}01, 0.3000000E{+}01, -.1000000E{+}01)^T$

- Wie lautet die inverse Matrix von A?

  Das numerische Ergebnis ist fast bis auf Maschinengenauigkeit exakt (bis auf maximal zwei Einheiten in der siebenten Stelle hinterm Komma) und lautet:

```
Invertierte Matrix:

   .9379443E+00   -.6843720E-01   -.7960770E-01   -.8592076E-01
  -.8852433E-01    .9059826E+00   -.9919081E-01   -.1055899E+00
  -.1113511E+00   -.1169667E+00    .8784252E+00   -.1270733E+00
  -.1354557E+00   -.1401826E+00   -.1438075E+00    .8516058E+00

Test-Matrix:

   .9999999E+00    .1126516E-07    .2518815E-08   -.1120211E-08
  -.4888310E-08    .1000000E+01   -.6621389E-08   -.3829147E-08
  -.2647183E-08    .5092004E-08    .9999999E+00    .1665558E-07
   .7356128E-08   -.1727934E-07   -.1442864E-07    .1000000E+01
```

  Die Test-Matrix stellt einfach das Multiplikations-Ergebnis: gegebene Matrix mal (numerisch) invertierte Matrix dar. Zu erwarten wäre im Idealfall die Einheitsmatrix.

- Welchen Wert hat die Determinante der Matrix A?

  Das numerische Ergebnis (auf Maschinengenauigkeit exakt) lautet

$$\text{Determinante} = 0.1758306E + 01$$

2.4 Verbesserung eines Lösungsvektors durch Nachiteration.

Es ist klar, daß die Genauigkeit der numerischen Lösung eines linearen Gleichungssystems durch die Maschinengenauigkeit begrenzt ist. Leider ist es in vielen Fällen so, daß diese Genauigkeit wegen auftretender Rundungsfehler-Kumulationen nicht erreicht wird.

In solchen Fällen gibt es die Möglichkeit einer *iterativen Verbesserung* der numerischen Lösung, deren (sehr einfache) Theorie nun kurz erläutert wird:

Die numerische Lösung des Systems

$$A \cdot \mathbf{x} = \mathbf{b}$$

sei der durch Rundungsfehler verfälschte Vektor $\bar{\mathbf{x}} = \mathbf{x} + \delta\mathbf{x}$. Setzt man nun $\bar{\mathbf{x}}$ in das Gleichungssystem ein, so erhält man den verfälschten inhomogenen Vektor $\mathbf{b} + \delta\mathbf{b}$:

$$A \cdot \bar{\mathbf{x}} = A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$$

bzw.

$$\underbrace{(A\mathbf{x} - \mathbf{b})}_{=0} + A \cdot \delta\mathbf{x} = \delta\mathbf{b}\,.$$

Somit ergibt sich der Fehlervektor $\delta\mathbf{x}$ als Lösung des Systems

$$A \cdot \delta\mathbf{x} = \delta\mathbf{b} \quad ,$$

wobei A die *unveränderte* Koeffizientenmatrix und $\delta\mathbf{b}$ den sogenannten *Residuenvektor* $A\bar{\mathbf{x}} - \mathbf{b}$ darstellt.

Das Struktogramm 5 zeigt die Lösung eines linearen Gleichungssystems mit anschließender iterativer Verbesserung des Lösungsvektors:

$$\mathbf{x} = \bar{\mathbf{x}} - \delta\mathbf{x} \quad .$$

Dazu noch einige Anmerkungen:

1. Da beim ersten Aufruf von LUDCMP die Matrix A zerstört wird, muß diese Größe rechtzeitig gesichert werden.

2. Wegen der großen Gefahr einer *subtractive cancellation* sollte die Berechnung des Residuenvektors - wenn irgend möglich - mit doppelter Genauigkeit erfolgen.

3. Manche Programm-Bibliotheken (z.B. NAG) enthalten Programme zur Lösung inhomogener Gleichungssysteme, die eine iterative Verbesserung bereits inkludiert haben.

Struktogramm 5 — Iterative Verbesserung des Lösungsvektors

```
I=1(1)N
    J=1(1)N
        ASP(I,J):=A(I,J)
LUDCMP (A,N,INDX,D,KHAD)
LUBKSB (A,N,INDX,B,X)
    I=1(1)N
        SUMDP:=-B(I)
        J=1(1)N
            SUMDP:=SUMDP + DBLE(ASP(I,J))*DBLE(X(J))
        RES(I):=SUMDP
    LUBKSB (A,N,INDX,RES,DELX)
    I=1(1)N
        X(I):=X(I)-DELX(I)
(Bedingung, unter welcher die Iteration
verlassen wird)
(Ausgabe und return)
```

2.5 Rundungsfehler-Probleme mit schlecht konditionierten bzw. singulären Systemen.

Die Methode der partiellen Pivotisierung bewirkt eine entscheidende Reduktion der beim Gauss'schen Verfahren auftretenden Rundungsfehler. Dennoch kann bei *schlecht konditionierten* Gleichungssystemen der numerisch ermittelte Lösungsvektor deutlich fehlerbehaftet sein, wie das folgende Beispiel zeigt:

```
0.10000E+01 0.50000E+00 0.33333E+00 0.25000E+00 0.20000E+00   0.228333E+01

0.50000E+00 0.33333E+00 0.25000E+00 0.20000E+00 0.16667E+00   0.145000E+01

0.33333E+00 0.25000E+00 0.20000E+00 0.16667E+00 0.14286E+00   0.109286E+01

0.25000E+00 0.20000E+00 0.16667E+00 0.14286E+00 0.12500E+00   0.884530E+00

0.20000E+00 0.16667E+00 0.14286E+00 0.12500E+00 0.11111E+00   0.745640E+00
```

Die Koeffizientenmatrix stellt eine *Hilbert-Matrix* 5. Ordnung mit den auf 5 signifikante Stellen gerundeten Komponenten

$$a_{ij} = \frac{1}{i+j-1}$$

dar, und der inhomogene Vektor ist so gewählt, daß die exakte Lösung

$$x_1 = x_2 = x_3 = x_4 = x_5 = 1$$

lautet. Die Konditionszahl lautet $K_{\mathrm{H}} = 0.55{\cdot}10^{-10}$; es liegt also ein sehr schlecht konditioniertes System vor, und Schwierigkeiten sind zu erwarten!

Das exakte Ergebnis wird im folgenden dem numerischen Ergebnis $\bar{\mathbf{x}}$ gegenübergestellt (FORTRAN, single precision). Weiters ist zur Kontrolle der Vektor $A \cdot \bar{\mathbf{x}}$ ausgegeben:

$\mathbf{x}$ (exakt)	$\bar{\mathbf{x}}$ (numerisch)	$A \cdot \bar{\mathbf{x}}$
1.	0.9999459E+00	0.2283330E+01
1.	0.1000955E+01	0.1450000E+01
1.	0.9960009E+00	0.1092860E+01
1.	0.1005933E+01	0.8845300E+00
1.	0.9971328E+00	0.7456400E+00

Das Dilemma ist deutlich zu sehen: Obwohl einige Komponenten des numerisch erhaltenen Lösungsvektors beträchtlich vom exakten Wert abweichen, ist das lineare Gleichungssystem mit Maschinengenauigkeit erfüllt! Auch eine Nachiteration bringt keine Verbesserung der Ergebnisse.

Dieses Beispiel enthüllt noch ein weiteres ernstes Problem bei der Verwendung numerischer Methoden zur Lösung linearer Systeme: Es gibt trotz zahlreicher theoretischer Untersuchungen (s. z.B. [12], S.109ff) keine einfache, also in der Praxis einsetzbare Abschätzung des im Lösungsvektor enthaltenen Rundungsfehlers.

Eine weitere unangenehme Rundungsfehler-Konsequenz besteht darin, daß es bei der numerischen Auswertung praktisch unmöglich ist, klar zwischen *singulären* und *fast-singulären* Koeffizientenmatrizen zu unterscheiden. Im Programm wird eine Matrix dann als singulär erkannt, wenn alle Elemente einer Pivot-Spalte exakt Null sind. Da aber diese Elemente ihrerseits Ergebnisse arithmetischer Operationen sind, ist wegen der dabei auftretenden

Rundungsfehler *auch im Falle der Singularität der Koeffizientenmatrix* nicht mit exakten Nullen zu rechnen (das folgende aus [12], S.63):

Die Lage wird dadurch besonders kompliziert, daß die scharfe mathematische Unterscheidung von singulären und nicht-singulären Matrizen nur in der Idealwelt des Mathematikers existiert. Sobald wir mit Matrizen in gerundeter Arithmetik arbeiten, wird die Unterscheidung notwendigerweise ungerechtfertigt. So können gewisse nicht-singuläre Matrizen als Ergebnis von durch die Rundungsfehler eingeführten Störungen singulär werden. Noch wahrscheinlicher wird eine wirklich singuläre vorgegebene Matrix durch den Rundungsfehler in eine benachbarte nicht-singuläre Matrix abgeändert.

Aus diesem Grund verzichten viele Programme auf eine Singularitäts-Diagnose und überlassen diese dem Benutzer.
Im Programm LUDCMP wird zwar abgefragt, ob im Laufe der Rechnung irgendein Diagonalelement γ_{jj} exakt(!) Null ist, aber nur, um in diesem (unwahrscheinlichen) Fall eine Division durch Null zu verhindern!

2.6 Direkte Lösungsverfahren für Systeme mit speziellen Koeffizientenmatrizen.

Nachdem direkte Verfahren zur Lösung linearer Gleichungssysteme relativ rechen- und speicherplatz-intensiv sind, ist es in der Praxis von sehr großer Bedeutung, spezielle Koeffizientenmatrizen auszunutzen, um vereinfachte Algorithmen zu formulieren.

2.6.1 Lösung von Gleichungssystemen mit tridiagonaler Koeffizientenmatrix.

Eine Reihe wichtiger numerischer Verfahren (z.B. die Spline-Interpolation) führen zu linearen Gleichungssystemen der Form

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 \\ . & & & & & . \\ . & & & & & . \\ . & & & & & c_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ . \\ . \\ . \\ r_n \end{pmatrix} \tag{2.15}$$

wobei die *tridiagonale* Koeffizientenmatrix eindeutig durch die drei Vektoren **b** (Hauptdiagonale) sowie **a** und **c** (untere bzw. obere Nebendiagonale) gegeben ist. Wendet man in diesem Fall die LU-Zerlegung (ohne Pivotisierung, s.u.) an, so erhält man die Struktur

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_2 & 1 & 0 & \cdots & 0 \\ 0 & m_3 & 1 & \cdots & 0 \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \underbrace{\begin{pmatrix} u_1 & c_1 & 0 & \cdots & 0 \\ 0 & u_2 & c_2 & \cdots & 0 \\ . & & & & . \\ . & & & & . \\ . & & & & c_{n-1} \\ 0 & 0 & 0 & \cdots & u_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix}}_{\equiv \mathbf{y}} = \begin{pmatrix} r_1 \\ r_2 \\ . \\ . \\ . \\ r_n \end{pmatrix}$$

wobei sich bei Anwendung der Glg.en (2.5-2.13) die folgenden einfachen Formeln ergeben:

$$\begin{aligned} u_1 &= b_1 \\ y_1 &= r_1 \end{aligned}$$

$$\begin{aligned} m_j &= a_j/u_{j-1} \\ u_j &= b_j - m_j \cdot c_{j-1} \qquad j = 2, \ldots, n \\ y_j &= r_j - m_j \cdot y_{j-1} \end{aligned} \tag{2.16}$$

Die Komponenten des Lösungsvektors ergeben sich wieder mittels Rück-Substitution:

$$\begin{aligned} x_n &= y_n/u_n \\ x_j &= (y_j - c_j \cdot x_{j+1})/u_j \qquad j = n-1, \ldots, 1 \end{aligned} \tag{2.17}$$

Diese Gleichungen sind die Basis für das folgende Programm.

2.6.2 Das Programm TRID.

Quelle: [9], S. 43 (mit Änderungen).

Das Programm TRID berechnet den Lösungsvektor eines linearen, inhomogenen Gleichungssystems mit tridiagonaler Koeffizientenmatrix.

INPUT-Parameter:

A( ),B( ),C( ): Vektoren der tridiagonalen Matrix.

R( ): inhomogener Vektor.

N: Ordnung des Systems.

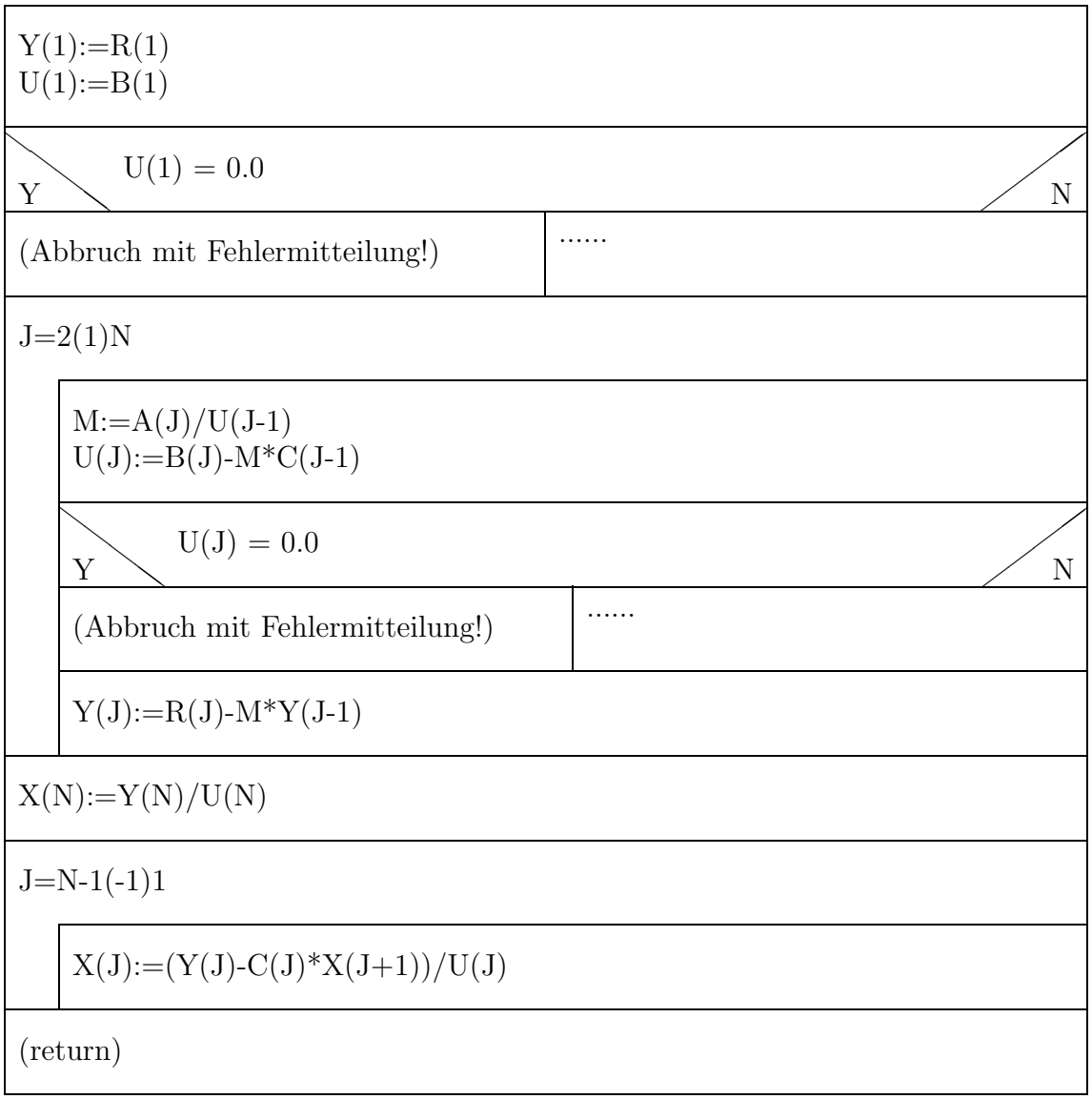
OUTPUT-Parameter:

X( ): Lösungsvektor.

Interne Vektoren:

U( ), Y( ): .

Struktogramm 6 — TRID(A,B,C,R,N,X)



Anmerkungen zu TRID:

1. Der Hauptvorteil von TRID liegt natürlich in einer beträchtlichen Speicherplatz-Ersparnis, da ja als Input-Informationen nur die vier eindimensionalen Felder A, B, C und R eingegeben werden müssen. Im Programm kommen dazu noch die ebenfalls eindimensionalen internen Felder U und Y sowie der Lösungsvektor X. Bei einem Problem n-ter Ordnung bedeutet das einen Speicherplatzbedarf von $7n$ gegenüber $n^2 + n$ beim normalen LU-Verfahren.

2. Ein weiterer Vorteil gegenüber dem normalen LU-Verfahren ist die beträchtlich verminderte Rechenzeit.

3. Allerdings bedeutet der Verzicht auf die Pivotisierung, daß es auch bei *nicht-singulären* Problemen zu einem Abbruch des Verfahrens kommen kann! Prinzipiell würde der Einbau einer Pivotisierung ins Programm TRID möglich sein, es würde sich jedoch dabei die Bandbreite der Matrix erhöhen (und zwar im ungünstigsten Fall verdoppeln).

   Es kann jedoch gezeigt werden, daß bei allen sogenannten *diagonal- dominanten* tridiagonalen Matrizen, d.h. bei solchen, die die Eigenschaft $\mid b_j \mid > \mid a_j \mid + \mid c_j \mid$ für alle $j = 1, \ldots, n$ haben, keine derartigen Probleme auftreten. Diese Feststellung ist von Bedeutung, weil viele in der Praxis auftretenden tridiagonalen Matrizen diese Eigenschaft der *Diagonal-Dominanz* haben!

4. Viele Programmbibliotheken enthalten Programme für die Lösung linearer Gleichungssysteme mit tridiagonalen Koeffizientenmatrizen: siehe z.B. [2], S.304 und [9], S.42.

2.6.3 Weitere Sonderformen der Koeffizientenmatrix.

In der Literatur findet man noch eine große Anzahl von Algorithmen bzw. Programmen für die numerische Behandlung von linearen Gleichungssystemen, deren Koeffizientenmatrizen besondere Strukturen haben. Insbesondere soll auf [2] und auf [9] hingewiesen werden, wo man Informationen zu den Themen: symmetrische Matrizen (Cholesky-Verfahren), zyklisch-tridiagonale Matrizen, fünfdiagonale Matrizen, allgemeine Bandmatrizen, Blockmatrizen usw. erhält.

2.7 Das Gauss-Seidel-Verfahren.

2.7.1 Allgemeines.

Das *Gauss-Seidel-Verfahren* (sowie seine Variationen, die in der Literatur unter den Namen *Einzelschrittverfahren, Gesamtschrittverfahren, Relaxationsverfahren, SOR-Verfahren usw.* zu finden sind) ist ein *iteratives Verfahren zur Lösung von linearen inhomogenen Gleichungssystemen.*

Wie schon zu Beginn dieses Kapitels erwähnt wurde, weisen *iterative* Verfahren gegenüber den *direkten* Verfahren einige Vorteile (Einfachheit des Algorithmus usw.), aber auch Nachteile (eventuell Konvergenzprobleme) auf.

Der wichtigste Vorteil der iterativen Verfahren liegt aber wohl in der Tatsache, daß *die gegebene Koeffizientenmatrix während des Iterationsprozesses nicht verändert wird.* Dieser Vorteil wirkt sich - wie im folgenden gezeigt wird - ganz besonders positiv aus, wenn man lineare Gleichungssysteme von hoher Ordnung auszuwerten hat, die *schwach besetzte Koeffizientenmatrizen* haben.

Eine Matrix heißt schwach besetzt, wenn nur wenige ihrer Koeffizienten von Null verschieden sind.

Da sich einige sehr wichtige Probleme der Numerischen Mathematik (z.B. die Lösung von Randwertproblemen mittels der Differenzenmethode) auf lineare Gleichungssysteme mit schwach besetzten Matrizen zurückführen lassen, soll im folgenden vor allem auf derartige Probleme eingegangen werden.

2.7.2 Die Grundprinzipien des Gauss-Seidel-Verfahrens.

Ausgangspunkt ist wieder das lineare System (2.1):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= f_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= f_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= f_n \end{aligned}$$

Unter der Voraussetzung, daß *alle* Diagonalelemente der Koeffizientenmatrix ungleich Null sind, kann nun jede Gleichung des Systems nach der zum jeweiligen Diagonalelement gehörenden x-Komponente aufgelöst werden:

$$x_1 = -\frac{1}{a_{11}}\left[a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n - f_1\right]$$

$$x_2 = -\frac{1}{a_{22}}\left[a_{21}x_1 + a_{23}x_3 + \cdots + a_{2n}x_n - f_2\right]$$

bzw. allgemein

$$x_i = -\frac{1}{a_{ii}}\left[\sum_{j=1(j\neq i)}^{n} a_{ij}x_j - f_i\right] \quad . \tag{2.18}$$

Das so umgeformte Gleichungssystem kann als Matrixgleichung

$$\mathbf{x} = C \cdot \mathbf{x} + \mathbf{b}$$

mit

$$C = \{c_{ij}\} \quad \text{mit} \quad c_{ij} = \begin{cases} -a_{ij}/a_{ii} & i \neq j \\ 0 & i = j \end{cases} \qquad \text{sowie} \qquad b_i = \frac{f_i}{a_{ii}} \tag{2.19}$$

geschrieben werden. Mit der trivialen Umformung von Glg. (2.18)

$$x_i = x_i - \left(x_i + \frac{1}{a_{ii}} \left[\sum_{j=1(j\neq i)}^{n} a_{ij} x_j - f_i \right] \right)$$

erhält man sofort die *Iterationsvorschrift für das Gauss-Seidel-Verfahren.* Setzt man nämlich in die rechte Seite der obigen Gleichung irgendwelche Näherungswerte für die x_i ein, so ergeben sich - Konvergenz vorausgesetzt - *verbesserte* Näherungswerte für die Komponenten des Lösungsvektors:

$$x_i^{(t+1)} = x_i^{(t)} - \Delta x_i^{(t)} \qquad (i = 1, \ldots, n) \tag{2.20}$$

mit

$$\Delta x_i^t = x_i^{(t)} + \frac{1}{a_{ii}} \left[\sum_{j=1(j\neq i)}^{n} a_{ij} x_j^{(t)} - f_i \right] \quad . \tag{2.21}$$

Auf diese Weise kann im Falle einer Konvergenz eine *Folge von Näherungsvektoren*

$$\mathbf{x}^{(0)} \quad ; \quad \mathbf{x}^{(1)} \quad ; \quad \mathbf{x}^{(2)} \quad ; \quad \ldots \quad ; \quad \mathbf{x}^{(t)}$$

für den exakten Lösungsvektor $\mathbf{x}$ berechnet werden, wobei man diesen zwar niemals erhält, sich aber - abgesehen von Rundungsfehlern - beliebig nahe an ihn 'heranarbeiten' kann:

$$\lim_{t\to\infty} \mathbf{x}^{(t)} \to \mathbf{x}$$

Der Vektor $\mathbf{x}^{(0)}$ heißt der *Startvektor* der Iteration.

Typisch für alle Iterationsverfahren sind die sogenannten *Abbruchbedingungen*, die festlegen, wann eine Iteration zu beenden ist.

In jedem Iterationsprogramm sollten zumindest die beiden folgenden Abbruchbedingungen enthalten sein:

- Die Iteration ist zu beenden, wenn die Ergebnisse eine vorgegebene Genauigkeit erreicht haben.
- Die Iteration ist in jedem Fall zu beenden, wenn eine vorgegebene maximale Zahl von Iterationsschritten erreicht ist ('Notausgang' im Falle einer Divergenz bzw. einer sehr langsamen Konvergenz).

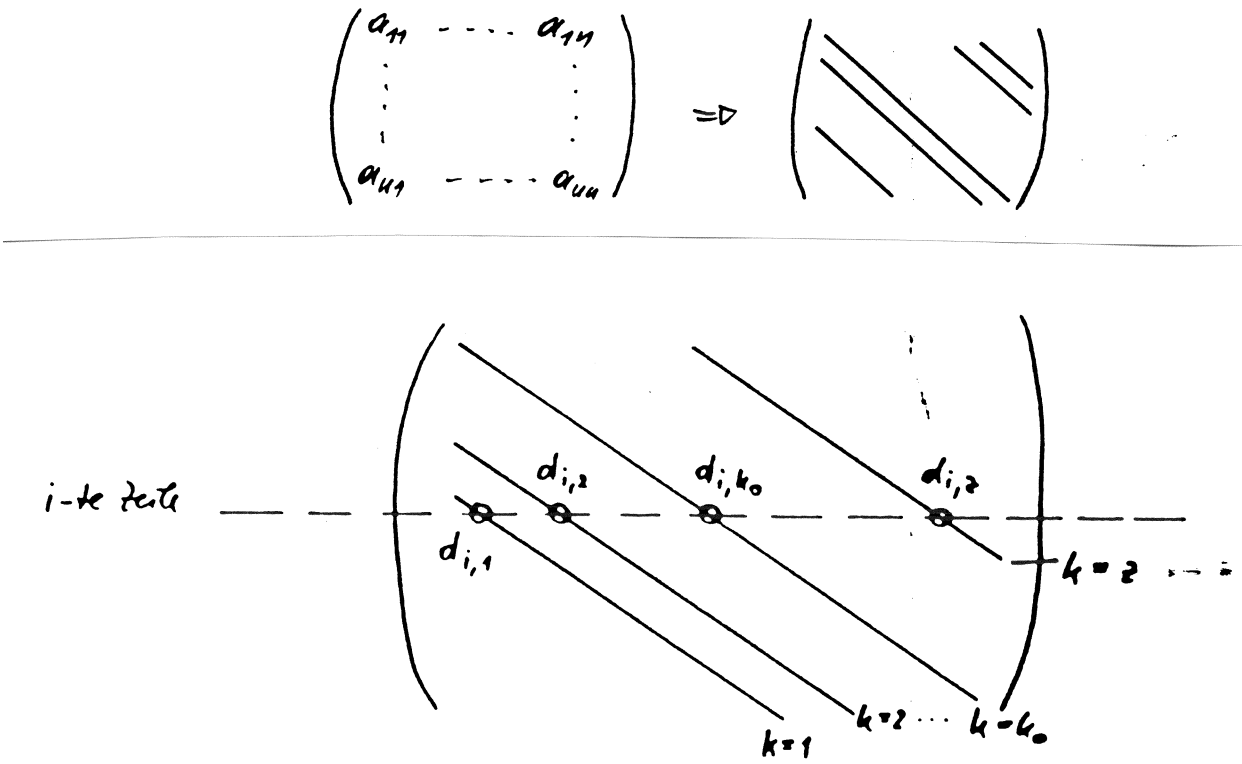


Abbildung 2.2: Zur Abspeicherung einer Bandmatrix.

2.7.3 Das Gauss-Seidel-Verfahren für Bandmatrizen.

Eine Bandmatrix ist eine Matrix, die nur entlang einiger Diagonalen Komponenten enthält, die von Null verschieden sind (s. Abbildung 2.2 (oben)).

Es ist sofort einzusehen, daß es vom Standpunkt der Speicherplatz-Ökonomie ein Nonsens ist, eine solche Matrix als (n x n)-Matrix abzuspeichern. Die gesamte Information kann nämlich viel platzsparender abgespeichert werden, wenn man *die Diagonalen als Vektoren speichert und über die Lage der einzelnen Diagonalen innerhalb der Matrix buchführt.*
Wie dies z.B. geschehen kann, ist in der Abbildung 2.2 (unten) schematisch dargestellt:
z sei die Anzahl der in der Matrix vorkommenden Diagonalen. Die Lage jeder Diagonale wird nun durch eine ganze Zahl angegeben, die angibt, wo sich diese Diagonale *relativ zur Hauptdiagonale* befindet. Damit kann die Abspeicherung der gesamten Information in einem zweidimensionalen Feld d_{ik} erfolgen, wobei der Index i die jeweilige Matrixzeile bedeutet. Der zweite Index (k) gibt an, um welche der z Diagonalen es sich handelt. Zusätzlich wird ein INTEGER-Feld $s(k)$ mit den relativen Positionen definiert:

$$s(k) = \begin{cases} 0 & \text{Hauptdiagonale} \\ +(-)1 & \text{erste obere(untere) Nebendiagonale} \\ +(-)t & t\text{–te obere(untere) Nebendiagonale} \end{cases}$$

Da beim Gauss-Seidel-Verfahren die Elemente der Hauptdiagonale eine besonders wichtige Rolle spielen, soll der Index k, der die Hauptdiagonale bezeichnet, als k_o hervorgehoben werden. Es gilt also

$$s(k = k_o) = 0$$

Die Zuordnung der Matrixelemente a_{ij} (i-te Zeile, j-te Spalte) zu den Elementen d_{ik} ist somit gegeben durch

$$a_{ij} = d_{ik} \quad \text{mit} \quad j = s(k) + i \quad , \tag{2.22}$$

wobei $i = 1, \ldots, n$ und $k = 1, \ldots, z$.

Auf Grund all dieser Überlegungen ist es kein Problem, die Iterationsvorschrift (2.21) im Hinblick auf (2.22) zu variieren:

$$x_i^{(t+1)} = x_i^{(t)} - \Delta x_i^{(t)} \tag{2.23}$$

mit

$$\Delta x_i^{(t)} = x_i^{(t)} + \frac{1}{d_{i,k_o}} \left[\sum_{k=1(k \neq k_o)}^{z} d_{ik} x_{s(k)+i}^{(t)} - f_i \right] \tag{2.24}$$

Anmerkung: In der Summe in Glg.(2.24)werden nur jene Summanden zugelassen, für welche gilt:

$$0 < s(k) + i \leq n$$

2.7.4 Konvergenzkriterien und Fehlerverhalten.

Es erhebt sich nun die wichtige Frage, ob man es einem gegebenen linearen Gleichungssystem schon a priori 'ansieht', daß es Schwierigkeiten bei der Konvergenz geben wird.

Nun gibt es in der Literatur in der Tat eine ganze Reihe von *Konvergenz-Kriterien*, d.h. von Bedingungen, die im Falle ihrer Erfüllung die Konvergenz der Gauss-Seidel-Iteration sicherstellen. Solche Kriterien sind zwar *hinreichend*, aber i.a. nicht *notwendig*, d.h. auch manche Systeme, die diese Kriterien nicht erfüllen, können mittels des Gauss-Seidel- Verfahrens gelöst werden.
Aus diesem Grund möchte ich hier auf die Angabe von mathematisch formulierten Konvergenz-Kriterien verzichten und mich auf eine in der Praxis bewährte Faustregel beschränken:

Alle linearen Gleichungssysteme, in deren Koeffizientenmatrizen die Hauptdiagonalelemente die anderen Matrixelemente dominieren, haben gute Chancen auf eine Gauss-Seidel-Konvergenz.

Ein konvergentes Verhalten der Iteration bedeutet natürlich, daß der auftretende Verfahrensfehler ϵ_V mit fortschreitender Iteration immer kleiner wird, daß also gilt:

$$\lim_{t \to \infty} \epsilon_V^{(t)} \to 0$$

In der Praxis wird solange iteriert, bis der Korrekturvektor $\Delta\mathbf{x}$ [Glg. (2.24)] genügend klein geworden ist. Man verwendet in diesem Fall als Abbruchkriterien die *absolute* Fehlerabfrage

$$\max \mid \Delta x_i \mid < \text{GEN}$$

bzw. die *relative* Fehlerabfrage

$$\max \mid \frac{\Delta x_i}{x_i} \mid < \text{GEN}$$

wobei GEN eine vom Programmbenutzer festgesetzte Fehler-Toleranz ist.

Achten Sie darauf, die Größe GEN nicht zu klein anzusetzen, da im Falle von schlecht konditionierten Problemen die entsprechende Rechengenauigkeit durch auftretende Rundungsfehler nicht erreicht werden kann!

Nun gibt es jedoch die Aussage, daß iterative Methoden i.a. wesentlich geringere Rundungsfehler aufweisen als die direkten Methoden. Es ist insbesondere eine Eigenschaft von Iterationsmethoden, daß jeweils nur die Rundungsfehler *des letzten Iterationsschrittes* einen Effekt haben: es gibt also keine ϵ_R-Kumulation über alle Iterationen hinweg! Auf Grund dieser Tatsache ist die Verwendung des Gauss-Seidel-Verfahrens in manchen Fällen auch dann einem direkten Verfahren vorzuziehen, wenn es höhere Rechenzeiten erfordert.

2.7.5 Das Unterprogramm GAUSEI.

Das Programm GAUSEI (GAUss-SEIdel) berechnet den Lösungsvektor eines inhomogenen linearen Gleichungssystems mit einer allgemeinen Bandmatrix als Koeffizientenmatrix.

INPUT-Parameter:

N: Ordnung des Systems.

NDIAG: Anzahl der in der Bandmatrix vorkommenden Diagonalen.

S( ): INTEGER-Feld mit den relativen Positionen der Diagonalen.

DIAG(,): Feld mit den Matrix-Koeffizienten: der erste Index spezifiziert die Matrixzeile, der zweite die Diagonale.

F( ): inhomogener Vektor des Systems.

TMAX: maximale Anzahl der Iterationsschritte.

W: Relaxationsparameter (s. Abschnitt 2.7.6).

IREL: IREL $\neq 1$: *absolute* Fehlerabfrage,
IREL $= 1$: *relative* Fehlerabfrage.

GEN: absoluter oder relativer Fehler, der im Laufe der Iteration zu unterschreiten ist.

OUTPUT-Parameter:

LOES( ): Lösungsvektor.

T: Anzahl der von GAUSEI durchgeführten Iterationsschritte.

FEHLER: logische Variable zur Fehlerdiagnostik:

FEHLER hat nach der Exekution von GAUSEI den Wert 'falsch', wenn die geforderte Genauigkeit erreicht wurde, und den Wert 'wahr', wenn

- nicht alle Werte der Hauptdiagonalen von Null verschieden sind.
- innerhalb von TMAX Iterationsschritten die geforderte Genauigkeit nicht erreicht wurde.

wichtige interne Variable:

K0: Index der Hauptdiagonalen.

DX: Iterations-Korrekturwert gemäß Glg.(2.24).

ISCH: Steuervariable bei der Genauigkeitsüberprüfung.

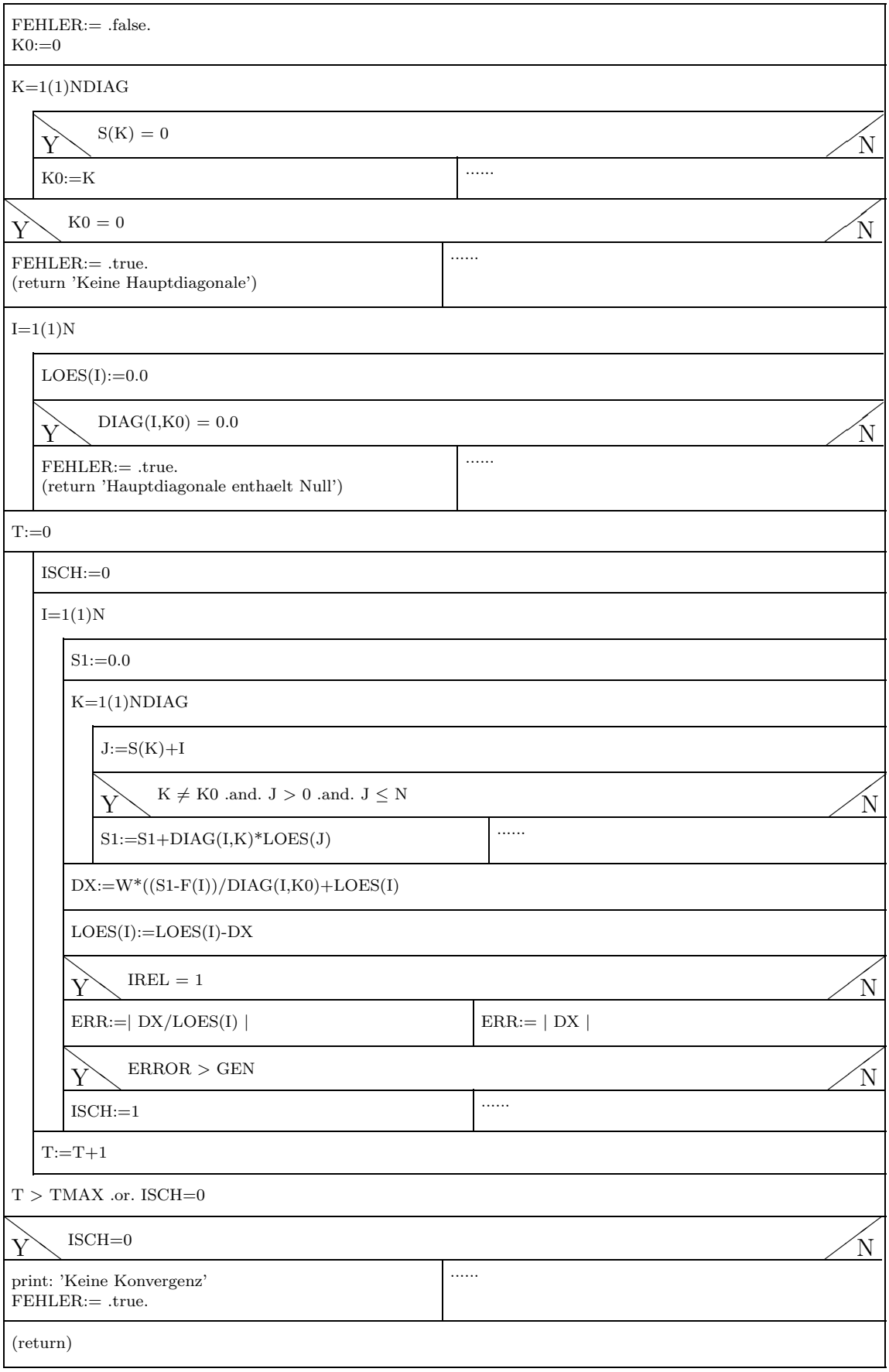
Programmstruktur von GAUSEI:

1. Check, welche der übergebenen Diagonalen die Hauptdiagonale ist. Der entsprechende Index k wird als K0 abgespeichert. Ist keine der übergebenen Diagonalen eine Hauptdiagonale, wird das Programm abgebrochen.

2. Check, ob die Hauptdiagonale Nullen enthält. Wenn ja, wird das Programm abgebrochen. Gleichzeitig wird der Startvektor der Iteration (= Nullvektor) definiert.

3. Durchführung der Gauss-Seidel-Iteration mit Fehlerabfrage.

Struktogramm 8 — GAUSEI(N,NDIAG,S,DIAG,F,TMAX,W,IREL,GEN, LOES,T,FEHLER)



2.7.6 Eine Variation des Gauss-Seidel-Verfahrens.

Die Iterationsvorschrift (2.23)

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \Delta\mathbf{x}^{(t)}$$

kann durch die formale Einführung eines Faktors ω im Sinne von

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \omega \cdot \Delta\mathbf{x}^{(t)} \tag{2.25}$$

erweitert werden. Im Falle von $\omega = 1$ liegt also die normale Gauss-Seidel-Iteration vor, bei $\omega > 1$ spricht man von einer *Überrelaxationsmethode*, bei $\omega < 1$ von einer *Unterrelaxationsmethode*.

Durch die richtige Wahl des Relaxationsparameters ω kann in manchen Fällen die Konvergenzgeschwindigkeit außerordentlich erhöht werden. Ein gutes Beispiel für diesen Effekt findet man in [7], S.192ff. Iteriert man das einfache lineare System

$$\begin{aligned} x + 2y &= 3 \\ x - 4y &= -3 \end{aligned}$$

bis zum Unterschreiten des absoluten Fehlers GEN=10^{-8} (C, double precision), so erhält man die folgende Abhängigkeit der erforderlichen Zahl von Iterationsschritten t von ω:

ω	t
0.65	20
0.70	18
0.75	15
0.8	14
0.85	12
0.9	12
0.95	21
1.0	31
1.05	48

Es wäre eine enorme Effizienzsteigerung der Gauss-Seidel-Methode, wenn man den *idealen Relaxationsparameter* ω_{ideal} *a priori* bestimmen könnte. Zu diesem Thema folgen nun einige Anmerkungen:

- Die Relaxationsmethode ist nur konvergent für $0 \leq \omega \leq 2$.
- 'Unter gewissen mathematischen Restriktionen', die i.a. von Matrizen erfüllt werden, welche bei der Anwendung von Differenzenverfahren (s. Kap. 9) zur Lösung von (insbesondere elliptischen) Randwertproblemen auftreten, liegt ω_{ideal} stets im Bereich zwischen 1 und 2, d.h. es liegt immer eine Überrelaxation vor.
- In solchen Fällen gilt:

$$\omega_{\text{ideal}} = \frac{2}{1 + \sqrt{1 - \lambda_1^2}}\,, \tag{2.26}$$

  wobei λ_1 den größten Eigenwert der Matrix C [Glg. (2.19)] darstellt.

- Da jedoch die genaue Berechnung von λ_1 häufig zu aufwendig ist, verweise ich auf einen Vorschlag in Ref. [14], S. 63, der sich in der Praxis gut bewährt hat und sehr einfach ins Programm GAUSEI eingebaut werden kann:

  Man kann zeigen, daß der Ausdruck

  $$\frac{|\Delta \mathbf{x}^{(t)}|}{|\Delta \mathbf{x}^{(t-1)}|} \tag{2.27}$$

  für $t \to \infty$ den Grenzwert λ_1^2 hat. Wenn man z.B. im Programm GAUSEI die Iteration mit $\omega = 1$ beginnt und die ersten t_0 Iterationen durchführt[1], so kann man mittels Glg. (2.27) in vielen Fällen schon eine ausreichend gute Approximation für λ_1^2 und damit mittels Glg (2.26) auch für ω_{ideal} erhalten. Mit diesem ω-Wert wird dann die Iteration fortgesetzt.

Wie gut diese Strategie funktioniert, wird in den Übungen zu dieser Vorlesung demonstriert.

2.7.7 Effizienz des Gauss-Seidel-Verfahrens

In den vorigen Abschnitten wurde als Vorteil der Gauss-Seidel-Methode angegeben, daß die Koeffizientenmatrix sich während der Iteration nicht ändert, was vor allem im Falle schwach besetzter Bandmatrizen günstig ist, weil man nur die besetzten Diagonalen abspeichern muß.

Dies soll an Hand eines konkreten Beispiels dokumentiert werden, und zwar für die numerische Auswertung einer sehr bedeutenden partiellen Differentialgleichung, nämlich der Laplace-Gleichung. Unter Verwendung der Differenzenmethode wird diese Differentialgleichung samt ihren Nebenbedingungen in ein System von linearen Gleichungen umgewandelt. Die Ordnung dieses Systems ergibt sich dabei aus der Zahl der Stützpunkte im Grundgebiet der zu lösenden Laplace-Gleichung. Die folgende Abbildung 2.3 (a) zeigt, wie viele Prozent der Elemente der Koeffizientenmatrix als Funktion der Stützpunktzahl tatsächlich besetzt sind. Man sieht, daß dieser Wert ab etwa 200 Stützpunkten deutlich unter 10 Prozent liegt.
Wie sich das auf die im Arbeitsspeicher des Computers nötigen Speicherplätze auswirkt, ist in Abb. 2.3 (b) dargestellt.

Überhuber ([22], S. 392) gibt ein Beispiel aus der industriellen Anwendung linearer Methoden (Karosserie-Festigkeitsberechnung im Automobilbau): Eine Matrix BCSSTK32 hat die Ordnung 44609, also $2 \cdot 10^9$ Elemente; von diesen sind aber 'nur' 1029655 Nicht-Nullelemente, d. h. der Besetzungsgrad beträgt 0.05 %.

[1] In der Praxis wird für t_0 ein Wert zwischen 20 und 100 genommen.

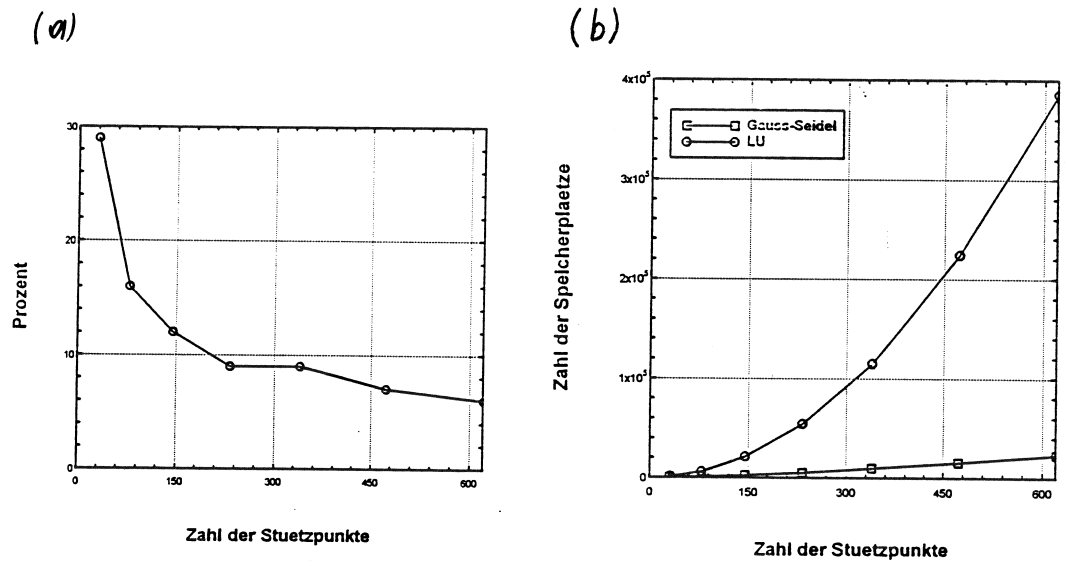


Abbildung 2.3: (a) Prozentueller Anteil der besetzten Matrixplätze als Funktion der Zahl der Stützpunkte. (b) Erforderliche Speicherplätze als Funktion der Zahl der Stützpunkte.

2.8 Software-Angebot

Am Ende jedes Hauptkapitels dieses Skriptums finden Sie unter dem Titel 'Software-Angebot' Hinweise auf Programme, die Sie sich entweder kostenlos übers Internet beschaffen können (*public domain* (p.d.) Programme), oder die kommerziell vertrieben werden.

Quellen: (1) Bücher von Ch. Überhuber [21], [22].
(2) Erfahrungen des Autors dieses Skriptums.

Diese Anmerkungen können auf Grund des riesigen Software-Angebotes nur erste Anregungen für Sie sein, dieses Angebot zu nutzen.

Zitat aus [21], S. 322:
Von herausragender Bedeutung für die rasche, einfache und effiziente Beschaffung von frei erhältlicher (public domain)-Software aus dem Numerik-Bereich ist der Internet-Dienst NETLIB (Dongara, Grosse, 'Distribution of Mathematical Software via Electronic Mail', Comm. ACM 30 (1987), 403).

Auf Programme aus dem NETLIB kann über verschiedene Internet-Dienste zugegriffen werden (email, FTP, Internet); am einfachsten ist ein 'Downladen' aus dem www-System. Ich werde mich im folgenden auf diese Möglichkeit beschränken.
Zuvor noch ein weiteres Zitat ([22], S. 183) mit speziellem Bezug auf das Kap. 2 dieses Skriptums:
Für die numerische Lösung linearer Gleichungssysteme gibt es, mehr als in jedem anderen Gebiet der Numerischen Datenverarbeitung, eine Fülle qualitativ hochwertiger Softwareprodukte: LAPACK, TEMPLATES, ITPACK, NSPCG, SLAP, UMFPACK, Teile der IMSL- und NAG-Bibliotheken. Die eigene Entwicklung von Software zur Lösung linearer Gleichungssysteme wäre aus diesem Grund ein sehr unökonomisches Unterfangen.

- Zugriff auf NETLIB: *www.netlib.org*
- Einen ersten Überblick über das Angebot:
  Klick auf *Browse the Netlib repository*

Die wichtigsten Libraries für den Problemkreis Lineare Gleichungssysteme**ß**ind die folgenden:

LAPACK:

Umfangreiche Sammlung von F77-Programmen zu den Themen 'Lösung linearer Gleichungssysteme', 'Eigenwertprobleme' u.a.
Die komplette aktuelle Bibliothek (31.5.2000) inklusive aller sog. BLAS-Hilfsprogramme[2] umfaßt ca. 5 MByte und kann durch Anklicken von
lapack.tgz downgeladen werden. Danach einfach auf Ihrem LINUX-Rechner dekomprimieren und in die einzelnen Programme aufteilen:

[2]BLAS: Basic Linear Algebra Subprograms.

```
gunzip -c lapack.tgz | tar xvf -
```

Achtung: Alle Programme (ohne OBJ- und EXE-Files) in den 4 Datentypen REAL, DOUBLE PRECISION, COMPLEX und COMPLEX*16 benötigen ca. 34 MByte.

- Wenn Sie LAPACK-Programme intensiv verwenden wollen, ist es vernünftig, sich den *LAPACK Users' Guide* (E. Anderson et al, Society for Industrial and Applied Mathematics, 1999) zuzulegen. Dieses Manual ist auch im Internet unter *www.netlib.org/lapack/lug/lapack_lug.html* erhältlich.

- Wenn Sie das ganze LAPACK auf Ihren Rechner installieren wollen, ist es auch nützlich, den *LAPACK Installation Guide* (ps-File, 232 kByte, 25 Seiten) downzuladen.

- Wenn Sie eine bestimmte LAPACK-Routine laden wollen, gehen Sie wie folgt vor: Klicken Sie im LAPACK-Verzeichnis *netlib.org/lapack/index.html* die entsprechende Eintragung in den *Individual Routines* an.
  Angenommen, Sie wollen ein DOUBLE-PRECISION-Lösungsprogramm für lineare Gleichungssysteme in Ihren Rechner laden. In diesem Fall klicken Sie auf *lib double*; als erste Eintragung in der erscheinenden Programmliste finden Sie:

  ```
  file dgesv.f   dgesv.f plus dependencies
  prec double
  for  Solves a general system of linear equations AX=B.
  ```

  Sie könnten nun durch Anklicken von *dgesv.f* das Programm speichern. Allerdings hat das i.a. wenig Sinn, da die meisten LAPACK-Programme eine ganze Kaskade von weiteren Programmen aufrufen. Es ist daher besser, diese ganze 'Programmfamilie' durch Anklicken von *dgesv.f plus dependencies* zu laden. Im erscheinenden Dialogfenster wählen Sie als Packaging formatTar-gzip, und die nötigen BLAS-Files laden Sie am besten gleich dazu.
  Wenn das geklappt hat, haben Sie in Ihrem Verzeichnis den File *netlibfiles.tgz*. Die weitere Verarbeitung lautet (s.o.):

  ```
  gunzip -c netlibfiles.tgz | tar xvf -
  ```

  Damit haben Sie sofort, auf drei neu erstellte Verzeichnisse aufgeteilt, die gesamte Programmfamilie von *dgesv.f* zur Verfügung:

  ```
  blas/blas2test.f
  blas/dgemm.f
  blas/dger.f
  blas/dscal.f
  blas/dswap.f
  ```

```
blas/dtrsm.f
blas/idamax.f
blas/xerbla.f
lapack/double/dgesv.f
lapack/double/dgetf2.f
lapack/double/dgetrf.f
lapack/double/dgetrs.f
lapack/double/dlaswp.f
lapack/util/ieeeck.f
lapack/util/ilaenv.f
```

Die folgende knappe Zusammenstellung gibt Ihnen eine Übersicht über das LAPACK-Angebot ([22], S. 266). Die Begriffe *simple drivers* und *expert drivers* bedeuten:
simple drivers: *black box*–Programme, die primär die numerische Lösung der jeweiligen Problemstellung liefern.
expert drivers liefern neben den Lösungen auch noch zusätzliche Informationen (Konditionsabschätzungen, Fehlerschranken usw.).

Matrixtyp (Speicherung)	*driver*	REAL	COMPLEX
allgemeine Matrix	*simple* *expert*	`sgesv` `sgesvx`	`cgesv` `cgesvx`
allgemeine Bandmatrix	*simple* *expert*	`sgbsv` `sgbsvx`	`cgbsv` `cgbsvx`
allgemeine Tridiagonalmatrix	*simple* *expert*	`sgtsv` `sgtsvx`	`cgtsv` `cgtsvx`
symmetrische/Hermitesche positiv definite Matrix	*simple* *expert*	`sposv` `sposvx`	`cposv` `cposvx`
symmetrische/Hermitesche positiv definite Matrix (gepackte Speicherung)	*simple* *expert*	`sppsv` `sppsvx`	`cppsv` `cppsvx`
symmetrische/Hermitesche positiv definite Bandmatrix	*simple* *expert*	`spbsv` `spbsvx`	`cpbsv` `cpbsvx`
symmetrische/Hermitesche positiv definite Tridiagonalmatrix	*simple* *expert*	`sptsv` `sptsvx`	`cptsv` `cptsvx`
symmetrische/Hermitesche indefinite Matrix	*simple* *expert*	`ssysv` `ssysvx`	`chesv` `chesvx`
symmetrische/Hermitesche indefinite Matrix (gepackte Speicherung)	*simple* *expert*	`sspsv` `sspsvx`	`chpsv` `chpsvx`
komplexe symmetrische Matrix	*simple* *expert*		`csysv` `csysvx`
komplexe symmetrische Matrix (gepackte Speicherung)	*simple* *expert*		`cspsv` `cspsvx`

LAPACK bietet nur direkte Verfahren in FORTRAN-77. Diese Programmiersprache ist zwar immer noch weitverbreitet im technisch-naturwissenschaftlichen Bereich; da aber neue Fortranversionen wie F90, F95

sowie andere Sprachen wie C und C++ immer wichtiger werden, gibt es inzwischen auch schon die entsprechenden LAPACK-Routinen:

LAPACK95

stellt ein Interface zu F77 dar, d. h. Sie benötigen auch die LAPACK-F77 Programme. LAPACK95 erhalten Sie über
www.netlib.org/lapack95/index.html.
Downladen von *lapack95.tgz* und nachfolgende Dekomprimierung erzeugt in Ihrer Directory das Unterverzeichnis LAPACK95. In diesem gibt es das Verzeichnis SRC mit allen zur Verfügung stehenden Routinen.

CLAPACK

erhalten Sie über *www.netlib.org/clapack*.
Es handelt sich dabei um eine *vollständige* C-Implementierung von LAPACK, wobei die Konvertierung von F77 in C automatisch erfolgte. Der Nachteil dieses Vorgehens ist allerdings, daß CLAPACK wertvolle Eigenschaften von C, wie z. B. dynamische Speicherallokation, nicht nutzt.
Darüber hinaus können Sie nicht einfach irgendein einzelnes CLAPACK-Programm herunterladen, sondern es ist eine komplette Installation von
clapack.tgz (Version 3.0, ca. 6 MByte) erforderlich. Dieser File enthält auch die Installationshilfen
readme.install und *readme.maintain* sowie den Header-File *clapack.h*

LAPACK++

stellt eine auf C++ basierende objektorientierte Schnittstelle zu LAPACK dar. Der Sourcecode heißt *lapack++.tgz* und enthält in komprimierter Form 66 kByte. Genauere Information über dieses System sind in den folgenden Postcript-Files zu finden:
overview.ps *classref.ps* *install.ps* *user.ps*

Eine weitere interessante Internet-Adresse zum Thema LAPACK++:
http://math.nist.gov/lapack++

SCALAPACK

Anwendung von verteilten und parallelen Rechnersystemen zur Lösung von Gleichungssystemen, Ausgleichsproblemen und Eigenwertproblemen mit sehr großen Koeffizientenmatrizen ($n > 5000$).
Eine kurze und prägnante Einführung in dieses Thema (Stand: 1995) s. [22], S. 283-286.

Mathematica und MATLAB:

Mathematica erlaubt sowohl symbolische als auch numerische Behandlung wichtiger Probleme der Linearen Algebra, wie z.B.:

Inverse[m] Matrix-Inversion
Det[m] Determinante
LinearSolve[m,b] Lösung des inhomogenen linearen Systems.

MATLAB:[3] im Prinzip bietet diese Benutzersoftware die einfachste Realisierung der Lösung des linearen Gleichungssystems A.x=f :

```
>>% Eingabe der Koeffizientenmatrix A:
>> A=[ ... ; ... ];
>>% Eingabe des inhomogenen Vektors f:
>> f=[ ... ],
>>% Loesung des Gleichungssystems:
>> x=A\f
```

Eine alternative Lösungsmöglichkeit bietet Matlab durch folgende Befehle:

```
>>% Der inhomogene Vektor wird zur Koeffizientenmatrix als letzte Spalte
>>% hinzugefuegt:
>> Aplus=[A f];
>>% Diese erweiterte Matrix wird als Argument der Funktion  rref
>>% verwendet. Nun wird die urspruengliche Matrix mittels des Verfahrens
>>% von Gauss-Jacobi in eine Einheitsmatrix umgewandelt, und die letzte
>>% Spalte enthaelt den Loesungsvektor:
>> Aneu=rref(Aplus);
>> x=Aneu(:,end)
>>% Dieses Verfahren hat den Vorteil, dass es etwas stabiler gegenueber
>>% Rundungsfehlern ist als die obige "einfache" Loesung.
```

2.8.1 Iterative Lösung schwach-besetzter Systeme

TEMPLATES

Diese NETLIB-Library enthält eine Reihe von Programmpaketen mit iterativen Lösungsmethoden für lineare Gleichungssysteme:

Dateiname	Inhalt der Datei
sctemplates.tgz	C-Routinen, einfache Genauigkeit
dctemplates.tgz	C-Routinen, doppelte Genauigkeit
cpptemplates.tgz	C++-Routinen
sftemplates.tgz	F77-Routinen, einfache Genauigkeit
dftemplates.tgz	F77-Routinen, doppelte Genauigkeit
mltemplates.tgz	Matlab-Routinen

Eine interessante Matlab-Routine ist *sor.m*. Dieses Programm löst ein lineares Gleichungssystem mittels der Sukzessiven Überrelaxationsmethode (allerdings ohne Berücksichtigung einer eventuellen Bandstruktur der Koeffizientenmatrix).

[3] MATrix LABoratory.