4 Lineare Gleichungssysteme

Dieses Kapitel beschäftigt sich mit der numerischen Lösung von linearen Gleichungssystemen. Wir betrachten hierzu folgende Verfahren zur numerischen Lösung solcher Systeme:

- Determinanten, Gauss-Jordan Elimination
- Matrixinversion
- Vereinfachte Verfahren für Tridiagonalsysteme (Gleichungssysteme mit fast diagonaler Koeffizientenmatrix)
- Iterative Verfahren

Alle Verfahren werden theoretisch begründet und mit konkreten konkreten Algorithmen gezeigt.

Für dieses Kapitel werden die Rechenregeln für Matrizen als bekannt vorausgesetzt. Eine Zusammenfassung der Linearen Algebra ist im Anhang dargelegt.

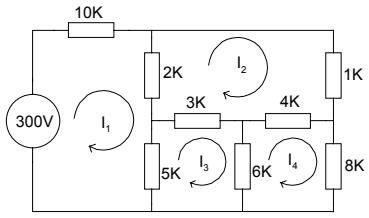
4.1 Einführung

Lineare Gleichungssysteme sind Gleichungssysteme mit mehreren Unbekannten wobei alle einzelnen Bestimmungsgleichungen ersten Grades sind.

$$\begin{array}{ccccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1n}x_n & = & b_1 \\
a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + & \cdots & + & a_{2n}x_n & = & b_2 \\
a_{31}x_1 & + & a_{32}x_1 & + & a_{33}x_3 & + & \cdots & + & a_{3n}x_n & = & b_3 \\
\vdots & & \vdots & & \vdots & & & & \vdots & & \\
a_{i1}x_1 & + & a_{i2}x_2 & + & a_{i3}x_3 & + & \cdots & + & a_{in}x_n & = & b_i \\
\vdots & & \vdots & & \vdots & & & & \vdots & & \vdots \\
a_{n1}x_1 & + & a_{n2}x_2 & + & a_{n3}x_3 & + & \cdots & + & a_{nn}x_n & = & b_n
\end{array} \qquad (4.1)$$

Sind die einzelnen n Bestimmungsgleichungen für die n Unbekannten linear unabhängig, so existiert eine nichttriviale Lösung dieses Gleichungssystems.

Lineare Gleichungssysteme treten in der Elektrotechnik sehr häufig auf. In Netzwerken mit ausschliesslich idealen, passiven Komponenten lassen sich die Ströme nach Kirchhof mit Maschengleichungen beschreiben. Wir erhalten ein lineares Gleichungssystem:



Mit dem Substitution $I_i=x_i$ erhalten wir das lineare Gleichungssystem:

$$\begin{array}{ccccccccc}
17x_1 & - & 2x_2 & - & 5x_3 & & & = & 300 \\
-2x_1 & + & 10x_2 & - & 3x_3 & - & 4x_4 & = & 0 \\
-5x_1 & - & 3x_2 & + & 14x_3 & - & 6x_4 & = & 0 \\
 & & -4x_2 & - & 6x_3 & + & 18x_4 & = & 0
\end{array}$$

Die Auflösung ergibt die Maschenströme $\{I_1$=23.0813mA, I_2=11.6398mA, I_3=13.8204mA, I_4=7.19343mA $\}$

Man kann obiges Gleichungssystem mit Matrizen beschreiben:

$$A \cdot \underline{x} = \begin{pmatrix} 17 & -2 & -5 & 0 \\ -2 & 10 & -3 & -4 \\ -5 & -3 & 14 & -6 \\ 0 & -4 & -6 & 18 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 300 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Die konkrete Lösung ist grundsätzlich auf verschiedene Arten möglich. Die meisten Verfahren haben jedoch eines gemeinsam: Das Gleichungssystem wird solange geeignet umgeformt bis die Koeffizientenmatrix *A* eine Dreieck-, oder besser, Diagonalgestalt hat.

In Elektrosimulationsprogrammen werden in komplexen Netzwerken Gleichungssysteme mit hunderten von Unbekannten aufgebaut und gelöst. Welche Methoden sich zur Lösung solcher Systeme eignen, wollen wir nachfolgend zeigen.

4.2 Gauss-Jordan Elimination

Das Verfahren von Gauss-Jordan ist das bekannteste Verfahren zur systematischen Lösung von linearen Gleichungssystemen und eine Reihe anderer Aufgaben.

Das Gauss-Jordan-Verfahren gestattet es ein lineares Gleichungssystem mit regulärer *nxn*-Koeffizientenmatrix systematisch in ein neues Gleichungssystem mit einer dreieckigen Koeffizientenmatrix umzuformen.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a'_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & a'_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b'_2 \\ \vdots \\ b'_n \end{pmatrix} \tag{4.2}$$

Bei einer dreieckigen Koeffizientenmatrix ist eine Unbekannte immer direkt bestimmbar. Die restlichen Unbekannten können schrittweise durch Einsetzen bestimmt werden.

Dabei werden folgende Elementaroperationen am Gleichungssystem vorgenommen:

Elementaroperationen für Gauss-Jordan Elimination

1. Addieren von 2 Zeilen
2. Multiplizieren einer Zeile mit einem konstanten Faktor.
3. Vertauschen zweier Zeilen

Jede dieser drei Elementaroperationen verändert das Gleichungssystem in der Lösung nicht. Die Operationen werden aber benutzt um das Gleichungssystem schrittweise zu vereinfachen.

Das Vereinfachen erfolgt soweit, dass am Schluss aus der *nxn* Koeffizientenmatrix eine Dreieckmatrix entsteht, die zum Rückwärtseinsetzen benutzt werden kann.

4.2.1 Numerisches Beispiel

Die Anwendung des Gauss-Algorithmus wird am besten an einem konkreten Beispiel gezeigt. Dazu betrachten wir Verfahren in einer einfachen Form, indem die grundsätzliche Arbeitsweise in allen Schritten durchgespielt wird. Für die Programmierung ist jedoch dieses einfache Verfahren nicht gut geeignet, da noch eine Reihe von Bedingungen betrachtet werden müssen um gute Resultate zu erhalten.

Wir betrachten ein Gleichungssystem mit 4 Unbekannten:

$$\begin{array}{rcrcrcrcr} 6x_1 & - & 2x_2 & + & 2x_3 & + & 4x_4 & = & 16 \\ 12x_1 & - & 8x_2 & + & 6x_3 & + & 10x_4 & = & 26 \\ 3x_1 & - & 13x_2 & + & 9x_3 & + & 3x_4 & = & -19 \\ -6x_1 & + & 4x_2 & + & x_3 & - & 18x_4 & = & -34 \end{array}$$

In einem ersten Durchlauf wollen wir alle Elemente der ersten Spalten ausser der ersten Zeile auf Null setzen. Dies erreichen wir durch Anwenden der Elementaroperationen 1,2,3. Konkret heisst dies, dass ein Vielfaches der ersten Zeile zur zweiten Zeile addiert wird und dito zur dritten und vierten Zeile.

Die notwendigen Koeffizienten des Vielfachen sind -2 für die zweite Zeile, -1/2 für die dritte und 1 für die vierte Zeile. Für die Handrechnung schreibt man hierzu zweckmässigerweise eine Tabelle mit der Koeffizientenmatrix und dem Konstantenvektor:

$$\left(\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 12 & -8 & 6 & 10 & 26 \\ 3 & -13 & 9 & 3 & -19 \\ -6 & 4 & 1 & -18 & -34 \end{array}\right)$$

Ein Teilschritt läuft also folgendermassen ab: Wir addieren das (-2)-fache der ersten Zeile zur zweiten, das (-0.5)-fache der ersten zur dritten und das 1-fache der ersten zur vierten Zeile und erhalten:

$$\left(\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & -12 & 8 & 1 & -27 \\ 0 & 2 & 3 & -14 & -18 \end{array}\right)$$

Wir erhalten nun eine neue Matrix deren Elemente $a_{21}..a_{41}$ Null sind. Wir wiederholen das Verfahren anlog für die zweite Zeile und erhalten nun eine Matrix die in der zweiten Spalte reduziert ist ($a_{32}..a_{43}$ ist Null):

$$\left(\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & 0 & 2 & -5 & -9 \\ 0 & 0 & 4 & -13 & -21 \end{array}\right)$$

Dieses Verfahren wiederholen wir solange, bis die Koeffizientenmatrix eine Dreieckform hat. Dazu multiplizieren wir hier die dritte Zeile mit -2, und addieren sie zur vierten Zeile:

$$\left.\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & 0 & 2 & -5 & -9 \\ 0 & 0 & 0 & -3 & -3 \end{array}\right.$$

Die so erhaltene dreieckige Koeffizientenmatrix nennt man wegen der Form **obere Dreieckmatrix**. Die Lösungen werden nun durch **Rückwärtseinsetzen** bestimmt:

$$x_4 = \frac{-3}{-3} = 1$$

Diesen Wert setzen wir nun in der dritten Zeile ein und erhalten x_3:

$$2x_3 - 5 = -9 \qquad \Rightarrow \qquad x_3 = \frac{-4}{2} = -2$$

Für die restlichen Unbekannten wird analog verfahren und wir erhalten als Lösung:

$$x_1 = 3 \qquad x_2 = 1 \qquad x_3 = -2 \qquad x_4 = 1$$

4.2.2 Gauss-Jordan Algorithmus

Wir schreiben das Gleichungssystem in Matrixform wobei die Matrix A (a_{ij}) die quadratische Koeffizientenmatrix darstellt. Die Unbekannten und die Konstanten werden in Spaltenvektoren notiert.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & {}_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix}$$

Lineares Gleichungssystem in Matrixschreibweise (4.3)

oder

$$A\underline{x} = \underline{b} \quad [1]$$

Gleichungssysteme dieser Art haben immer eine eindeutige Lösung wenn die Koeffizientenmatrix regulär ist.

Wir betrachten nun die Arbeitsweise des Gauss-Jordan Eliminationsalgorithmus für Gleichungssysteme mit *n* Gleichungen und *n* Unbekannten.

Das Verfahren besteht aus einer Vorwärtselimination mit (*n-1)* Schritten in (*n-1)* Durchläufen. Der Durchlauf benutzt die erste Gleichung um (*n-1)* Nullen für den Koeffizienten x_1 in den folgenden Zeilen zu erzeugen. Dies wird erreicht durch Subtraktion von geeigneten mehrfachen der ersten Zeile zur aktuellen Zeile. Die erste Zeile bezüglich der wir eliminieren, nennen wir **Pivot-Zeile** und a_{11} ist das **Pivot-Element**.

[1] Mit dem Unterstreichen von x und b bringen wir zum Ausdruck, dass es sich um Vektoren handelt und nicht um normale Zahlen. Vielfach wird aber nur *Ax=b* geschrieben, da man voraussetzt dass man weiss, wie das zu interpretieren ist.

Das grundsätzliche Schema für den ersten Durchlauf ist wie folgt:

$$\begin{aligned} a'_{ij} &\leftarrow a_{ij} - \left(\frac{a_{i1}}{a_{11}}\right) a_{1j} \\ b'_{i} &\leftarrow b_{i} - \left(\frac{a_{i1}}{a_{11}}\right) b_{1} \end{aligned} \qquad (1 \le i \le n) \tag{4.4}$$

In einem Schritt wird also a_{ij} ersetzt durch $a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}$.

Die Quotienten $\frac{a_{i1}}{a_{11}}$ heissen Multiplikatoren. Der neue Koeffizient für x_1 wird demnach in der *i*-ten Zeile Null, weil:

$$a_{i1} - \frac{a_{i1}}{a_{11}} a_{11} = 0$$

Nach dem ersten Durchlauf haben wir also ein System der Form:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} \\ 0 & a'_{32} & a'_{33} & \cdots & a'_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a'_{i2} & a'_{i3} & \cdots & a'_{in} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a'_{n2} & a'_{n3} & \cdots & a'_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_i \\ \vdots \\ b'_n \end{pmatrix}$$

Wir erhalten für die Zeilen 2..n neue Koeffizienten a'_{ij} wobei a_{1i} (i=2..n) immer 0 ist. Wir wiederholen nun den Durchlauf indem wir die zweite Zeile als Pivot-Zeile wählen. Wir erhalten die Koeffizienten für verbleibende Zeilen (Gleichungen):

$$\begin{aligned} a'_{ij} &\leftarrow a_{ij} - \left(\frac{a_{i2}}{a_{22}}\right) a_{2j} \\ b'_{i} &\leftarrow b_{i} - \left(\frac{a_{i2}}{a_{22}}\right) b_{2} \end{aligned} \qquad (2 \le i \le n) \tag{4.5}$$

Wir erhalten nach dem *k*-ten Durchlauf ein System der Form:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & & \cdots & & \cdots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \cdots & & \cdots & & \cdots & a'_{2n} \\ 0 & 0 & a'_{33} & \cdots & & \cdots & & \cdots & a'_{3n} \\ & & & \ddots & & & & & \vdots \\ 0 & \cdots & & 0 & a'_{kk} & \cdots & a'_{kj} & \cdots & a'_{kn} \\ & & & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & & 0 & a'_{ik} & & a'_{ij} & & a'_{in} \\ & & & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & & 0 & a'_{nk} & & a'_{nj} & & a'_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_k \\ \vdots \\ b'_i \\ \vdots \\ b'_n \end{pmatrix}$$

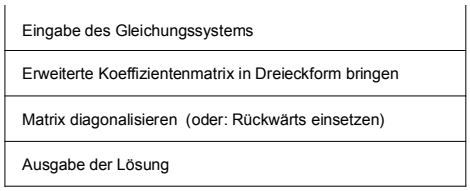
Unten links wurde ein Keil mit Nullen erzeugt und die ersten k Gleichungen sind bearbeitet und fixiert. Benutzen wir die k-te Gleichung als Pivot-Gleichung erhalten wir folgende neue Koeffizienten:

$$\begin{aligned} a_{ij}^{k+1} &\leftarrow a_{ij}^{k} - \left(\frac{a_{ik}^{k}}{a_{kk}^{k}}\right) a_{kj}^{k} \\ b_{i}^{k+1} &\leftarrow b_{i}^{k} - \left(\frac{a_{ik}^{k}}{a_{kk}^{k}}\right) b_{k}^{k} \end{aligned} \qquad (k \le i \le n) \tag{4.6}$$

Zu beachten ist, dass die Divisoren a_{kk} nie Null sein dürfen. Tritt dieser Fall trotzdem auf, so wird nach Regel 3 die Zeile mit dem Null-Pivot mit einer der nachfolgenden Zeilen vertauscht, deren Element nicht 0 ist. Eine Diskussion zur optimalen Pivotierung folgt später.

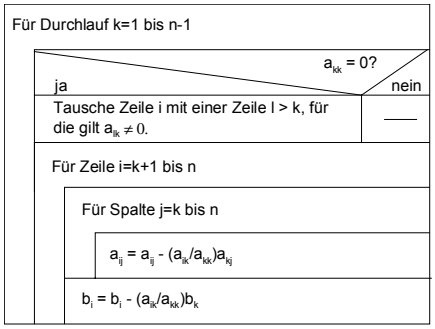
4.2.3 Entwicklung des Algorithmus

Die allgemeine Lösung besteht aus dem Erzeugen der Dreieckmatrix und dem Diagonalisieren derer. Als Top-Design kann etwa folgender Ablauf formuliert werden:

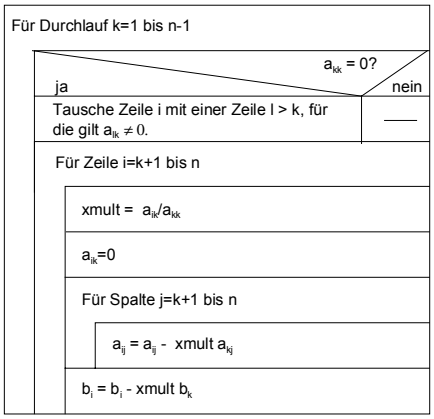


Das Einlesen der Koeffizienten ist an sich problemlos und wird hier nicht näher betrachtet. Grundsätzlich werden dort die Koeffizienten des Gleichungssystems und in einem zweidimensionalen Array a gespeichert. Der Zugriff erfolgt über zweifache Indizierung a_{ij}. Der Konstantenvektor b wird in einem eindimensionalen Array b abgelegt.

In einer ersten Phase entwerfen wir den Algorithmus zur **Vorwärtselimination** gemäss den Erkenntnissen des vorherigen Abschnittes. Vorwärtselimination heisst, die Koeffizientenmatrix wird in Dreieckform gebracht.



Da der Multiplikator $\frac{a_{ik}}{a_{kk}}$ nicht von j abhängig ist, kann dieser aus der Schleife herausgezogen werden. Der Wert für den Koeffizienten a_{ik} im k-ten Durchlauf gleich 0, wenn $j = k$ ist. Somit besteht keine Notwendigkeit diesen Wert zu berechnen. Stattdessen wird gerade die Konstante 0 eingespeichert. Wir verbessern den Entwurf dahingehend:



Wir erhalten hiermit ein Gleichungssystem mit dreieckiger Koeffizientenmatrix, das für das Rückwärtseinsetzen oder Diagonalisieren verwendet wird:

$$\begin{array}{ccccccccccccccc}
a_{11}x_{11} & + & a'_{12}x_2 & + & a'_{12}x_3 & + & \cdots & & & & \cdots & + & a'_{1n}x_n & = & b'_1 \\
 & & a'_{22}x_2 & + & a'_{23}x_3 & + & \cdots & & & & \cdots & + & a'_{2n}x_n & = & b'_2 \\
 & & & & a'_{33}x_3 & + & \cdots & & & & \cdots & + & a'_{3n}x_n & = & b'_3 \\
 & & & & & \ddots & & & & & & & \vdots & \vdots & \vdots \\
 & & & & & & & a'_{ii}x_i & + & a'_{i,i+1}x_{i+1} + & \cdots & + & a'_{in}x_n & = & b'_i \\
 & & & & & & & & \ddots & & & & & \vdots & \vdots \\
 & & & & & & & & & & a'_{n-1,n-1}x_{n-1} & + & a'_{n-1,n}x_n & = & b'_{n-1} \\
 & & & & & & & & & & & & a'_{nn}x_n & = & b'_n
\end{array}$$

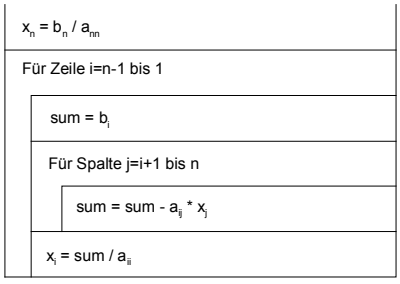
Das Rückwärtseinsetzen bringt die Koeffizientematrix in Diagonalgestalt und beginnt bei der letzten Zeile:

$$x_n = \frac{b_n}{a'_{nn}} \tag{4.7}$$

$$x_{n-1} = \frac{b_{n-1} - a'_{n-1,n}x_n}{a'_{n-1,n-1}} \tag{4.8}$$

$\cdots$

und allgemein: $$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right) \quad (i = n-1, n-2, \cdots, 1) \tag{4.9}$$



Ferner besteht die Möglichkeit den Gauss Algorithmus ein zweites Mal anzuwenden um die Koeffizientenmatrix in die Einheitsmatrix zu überführen.

Beide Methoden benötigen denselben Rechenaufwand von $\frac{n(n-1)}{2}$ Multiplikationen.

4.2.4 Praktische Implementierung

Ein mögliche Lösung zur Implementierung des Gauss-Algorithmus mit Rückwärtseinsetzen ist nachfolgend gezeigt.

```
/* Gauss Jordan Algorithmus zur Lösung linearer Gleichungssysteme.
   Lösung mit Test auf Nullpivot und Zeilenvertauschung.

   Autor: Gerhard Krucker
   Datum: 18.3.1995
   Sprache: MS Visual C V.15 (QuickWin)
*/

#include <stdio.h>

#define MAX  10       /* Maximale Anzahl der Unbekanten */

/* Funktionsprototypen fuer extern definierte Funktionen */
void LGS_write(double A[MAX][MAX], double b[MAX], int n);
void LGS_read(double A[MAX][MAX], double b[MAX], int *n);

main()
{ double A[MAX][MAX];   /* Koeffizientenmatrix */
  double x[MAX];        /* Resultatvektor */
  double b[MAX];        /* Konstantenvektor */
  int n;                /* Anzahl Bestimmungsgleichungen */
  int k;                /* Durchlauf */
  int i;                /* Zeile */
  int j;                /* Spalte */
  double xmult;
  double sum;

  printf("Gauss-Algorithmus für reguläre lineare Gleichungssysteme\n");
  LGS_read(A,b,&n);        /* Koeffizienten und Konstanten einlesen */
  LGS_write(A,b,n);        /* System zur Kontrolle Ausgeben */

  for (k=0; k < n-1;k++)
    { double temp;

      if (A[k][k] == 0)    /* Wenn A[k][k] == 0 dann Zeile tauschen */
       {  i = k+1;
        while ((i < n) && (A[i][k] == 0)) i++;
        if (i >= n)
           { printf("Fehler: System ist singulär\n");
             return 0;
           }
         for (j=k; j < n; j++)
          { temp = A[i][j];    /* Koeffizient vertauschen */
            A[i][j]=A[k][j];
            A[k][j]=temp;
           }
         temp=b[k];            /* Konstante vertauschen */
         b[k]=b[i];
         b[i]=temp;
       }

      for (i=k+1; i < n; i++)        /* Gauss Elimination */
        { xmult = A[i][k]/A[k][k];
          for (j=k+1; j < n; j++)
            A[i][j]=A[i][j]-xmult*A[k][j];
          b[i]=b[i]-xmult*b[k];
          A[i][k]=0.0;
        }
    }

  printf("\nDreieck Matrix: \n");
  LGS_write(A,b,n);

  /* Rueckwaerts einsetzen */
  x[n-1]=b[n-1]/A[n-1][n-1];
  for (i=n-2; i >= 0; i--)
    { sum = b[i];
      for (j=i; j < n; j++)
         sum = sum - A[i][j]* x[j];
      x[i]=sum / A[i][i];
    }

   printf("Rückwärts eingesetzt:\n");
   for (i=0; i<n;i++)
      printf("x%d: %9.5g\n",i+1,x[i]);

   return 0;
}
```

Ein Test des Programmes für das System

$$\begin{pmatrix} -1 & 8 & 3 \\ 2 & 4 & -1 \\ -2 & 1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix} \qquad \Rightarrow \qquad x = \begin{pmatrix} 5 \\ -1 \\ 5 \end{pmatrix}$$

liefert die Ausgabe:

```
Gauss-Algorithmus für reguläre lineare Gleichungssysteme

Anzahl Gleichungen? 3

Matrix A b zeilenweise eingeben.
Zeile 1? -1 8 3 2
Zeile 2? 2 4 -1 1
Zeile 3? -2 1 2 -1
       x1         x2         x3       |
-----------------------------------|----------
       -1          8          3       |          2
        2          4         -1       |          1
       -2          1          2       |         -1


Dreieck Matrix:
       x1         x2         x3       |
-----------------------------------|----------
       -1          8          3       |          2
        0         20          5       |          5
        0          0      -0.25       |      -1.25

Rückwärts eingesetzt:
x1:          5
x2:         -1
x3:          5
```

4.3 Invertieren von Matrizen

Eine der wichtigsten und gleichzeitig aufwendigsten Operationen der Matrizenrechnung ist die Matrixinversion. Die Inverse einer Matrix verkörpert das multiplikativ inverse Element einer Matrix.

Wird eine Matrix mit ihrer Inversen multipliziert, so erhalten wir die Einheitsmatrix, das neutrale Element der Matrixmultiplikation:

$$A \cdot A^{-1} = A^{-1} \cdot A = E \tag{4.10}$$

Es können ausschliesslich reguläre Matrizen invertiert werden, also Matrizen mit Determinante ungleich 0.

Zur Bestimmung der Inversen sind verschiedene Verfahren möglich:

- Entwicklungssatz mit Adjunkten
- Gauss-Algorithmus
- Iterative Verfahren

4.3.1 Lösung mit dem Entwicklungssatz

Diese Methode baut auf der Definition der Inversen auf. Sie ist geeignet um Inverse von kleinen Matrizen zu bestimmen indem man einsetzt:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$A^{-1} = \frac{1}{\det A}(\text{adj } A) = \frac{1}{\det A}\begin{pmatrix} |a_{22}| & -|a_{12}| \\ -|a_{21}| & |a_{11}| \end{pmatrix} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}}\begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} \tag{4.11}$$

Für grössere *n* wird diese Methode schnell aufwendig und bringt keine Vereinfachung.

Beispiel:

Bestimmen der Inversen A^{-1} der Matrix $A = \begin{pmatrix} 3 & 2 \\ 1 & 4 \end{pmatrix}$:

$$A^{-1} = \frac{1}{\det A}\begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} = \frac{1}{10}\begin{pmatrix} 4 & -2 \\ -1 & 3 \end{pmatrix} = \begin{pmatrix} \frac{4}{10} & \frac{-2}{10} \\ \frac{-1}{10} & \frac{3}{10} \end{pmatrix}$$

Kontrolle:

$$A \cdot A^{-1} = \begin{pmatrix} 3 & 2 \\ 1 & 4 \end{pmatrix}\begin{pmatrix} \frac{4}{10} & \frac{-2}{10} \\ \frac{-1}{10} & \frac{3}{10} \end{pmatrix} = \begin{pmatrix} \frac{12}{10} - \frac{2}{10} & \frac{-6}{10} + \frac{6}{10} \\ \frac{4}{10} - \frac{4}{10} & \frac{-2}{10} + \frac{12}{10} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

4.3.2 Bestimmung mit Gauss-Jordan-Algorithmus

Ausser zur Lösung von linearen Gleichungssystemen lässt sich der Gauss-Algorithmus auch zum Invertieren von (regulären) Matrizen verwenden. Grundlage für das Verfahren ist das Lösen der Matrixgleichung

$$A \cdot X = E \tag{4.12}$$

wobei $X = A^{-1}$ bestimmt wird.

Wenn A regulär ist, lässt sich *A* in endlich vielen Schritten von Elementaroperationen in eine Einheitsmatrix umformen. Werden nun gleichzeitig diese Schritte auf die Einheitsmatrix angewandt, so erhalten wir am Schluss die gesuchte Inverse A^{-1}:

$$A \cdot X = E \qquad \rightarrow \qquad E \cdot X = A^{-1} \tag{4.13}$$

Das Verfahren zur Lösung ist analog der Lösung eines linearen Gleichungssystems aus Kapitel 4.2.2, nur dass hier die rechte Seite aus *n* Spalten besteht, also *n* Gleichungssysteme gelöst werden müssen.

4.3.3 Implementierung der Matrixinversion

Wir zeigen nachfolgend eine einfache Implementierung des Verfahrens. Zuerst werden die Koeffizienten der zu invertierenden Matrix eingelesen und in einem Array *A* abgespeichert. Anschliessend wird eine Einheitsmatrix *A1* derselben Dimension, wie die eingelesene Matrix *A* erzeugt. Nun erfolgt eine Gauss-Elimination, wobei *A* auf Dreieckform gebracht wird. Da alle Operationen auch auf die Einheitsmatrix *A1* angewandt werden, wird diese Matrix nun auch dreieckig. In einem weiteren Schritt wird *A* zu einer Einheitsmatrix diagonalisiert. Auch hier werden alle Operationen parallel auf *A1* angewandt. Wir erhalten am Schluss als Resultat die Inverse von *A* im Array *A1*.

```
/* Bestimmen der Inversen einer Matrix mit Gauss-Jordan-Elimination.       (File: INVERS1.C)

   Arbeitsweise:
   Die regulaere Matrix A wird in dem ersten Durchlauf mit dem Gauss-
   Algorithmus in Dreieckform gebracht, wobei alle Operationen parallel
   auf die Einheitsmatrix A1 angewandt werden. In einem zweiten Durchlauf
   wird dann A diagonalisiert. Auch hier werden alle Operationen parallel
   auf A1 angewandt.

   Einschraenkungen:
   Keine optimale Wahl der Pivot-Elemente.

   Autor: Gerhard Krucker
   Datum: 28.3.1995
   Sprache: MS Visual C V1.5 (QuickWin) */

#include <stdio.h>
#define MAX 5      /* Maximale Dimension n der Matrizen */


main()
{ double A[MAX][MAX];     /* Zu invertierende Matrix A */
  double A1[MAX][MAX];   /* Inverse von A */
  int n;                  /* Aktuelle Dimension n der Matrix A*/
  int i,j;
  int k;                  /* Eliminationsdurchlauf # */

  printf("Matrixinversion mit Gauss-Algorithmus\n");

  /* Eingabe der Matrix A */
  printf("Eingabe der Matrixdimension n: ");
  scanf("%d",&n);

  printf("\nMatrix A zeilenweise eingeben:\n");
  for (i=0; i < n; i++)
   { printf("Zeile %d: ",i+1);
     for (j=0; j < n; j++)
       scanf("%lg",&A[i][j]);
   }

  /* A1 mit Einheitsmatrix initialisieren */
  for (i=0; i < n; i++)
     for (j=0; j < n; j++)
        if (i==j)
         A1[i][j] = 1.0;
          else
            A1[i][j] = 0.0;

  /* A in Dreieckform bringen. Parallel alle Operationen auf A1 anwenden */
    for (k=0; k < n-1;k++)
    { double temp;

      if (A[k][k] == 0)    /* Wenn A[k][k] == 0 dann Zeile tauschen */
       {  i = k+1;
        while ((i < n) && (A[i][k] == 0)) i++;
        if (i >= n)
           { printf("Fehler System ist singulär\n");
             return 0;
           }
         for (j=k; j < n; j++)
          { temp = A[i][j];    /* Koeffizient in A vertauschen */
            A[i][j]=A[k][j];
            A[k][j]=temp;
            temp = A1[i][j];    /* Koeffizient in A1  vertauschen */
            A1[i][j]=A1[k][j];
            A1[k][j]=temp;
           }
        }
      for (i=k+1; i < n; i++)        /* Gauss Elimination: Fuer jede Zeile ..*/
        { for (j=k+1; j < n; j++)    /* Dividiere die Zeile k durch Akk (Pivot wird = 1) */
            A[k][j] /= A[k][k];
          for (j=0; j < n;j++)
               A1[k][j] /=A[k][k];
          A[k][k]=1.0;

          for (j=k+1; j < n; j++)    /* Fuer  einen Teil von A */
             A[i][j]=A[i][j]-A[i][k]*A[k][j];

          for (j=0; j < n; j++)    /* Fuer alle Elemente von A1 */
```

```
            A1[i][j]=A1[i][j]-A[i][k]*A1[k][j];
          A[i][k]=0.0;
        }
    }
  /* Letzte Zeile durch Ann dividieren */
  for (j=0; j < n; j++)
       A1[n-1][j] /= A[n-1][n-1];
   A[n-1][n-1]=1.0;

#ifdef DEBUG
  printf("\nDreieck Matrix A: \n");
  for (i=0 ; i< n; i++)
    { printf("Zeile %d ",i+1);
    for (j=0; j <n;j++)
       printf("%lg  ",A[i][j]);
     printf("\n");
    }

  printf("\nDreieck Matrix A1: \n");
  for (i=0 ; i< n; i++)
    { printf("Zeile %d ",i+1);
    for (j=0; j <n;j++)
       printf("%lg  ",A1[i][j]);
     printf("\n");
    }
#endif

  /* Diagonalisieren */
  /* A in Diagonalform bringen. Parallel alle Operationen auf A1 anwenden */
  for (k=n-1; k >= 0 ;k--)
    {
      for (i=k-1; i >= 0; i--)        /* Gauss Elimination: Fuer jede Zeile ..*/
        { for (j=n-1; j > k; j--)
            A[i][j]=A[i][j]-A[i][k]*A[k][j];
          for (j=0; j < n; j++)
            A1[i][j]=A1[i][j]-A[i][k]*A1[k][j];
          A[i][k]=0.0;
        }
    }

  printf("\nInverse Matrix A1: \n");
  for (i=0 ; i < n; i++)
    { printf("Zeile %d ",i+1);
    for (j=0; j < n;j++)
       printf("%lg  ",A1[i][j]);
     printf("\n");
    }

  return 0;
}
```

Ein Testdurchlauf ergibt für die Matrix $\begin{pmatrix} 1 & 2 & 4 \\ 4 & 6 & 8 \\ 2 & 4 & 10 \end{pmatrix}$:

```
Matrixinversion mit Gauss-Algorithmus
Eingabe der Matrixdimension n: 3

Matrix A zeilenweise eingeben:
Zeile 1: 1 2 4
Zeile 2: 4 6 8
Zeile 3: 2 4 10

Inverse Matrix A1:
Zeile 1 -7  1  2
Zeile 2 6  -0.5  -2
Zeile 3 -1  0  0.5
```

4.4 Tridiagonalsysteme

In der Praxis trifft man sehr oft Gleichungssysteme mit fast diagonaler Koeffizientenmatrix.
Das Gleichungssystem $A \cdot \underline{x} = \underline{b}$ hat dann folgende Gestalt:

$$A \cdot x = \begin{pmatrix} & & 0 \\ & \diagdown\diagdown\diagdown & \\ 0 & & \end{pmatrix}$$

In elektrischen Netzwerken treten sehr oft solche fast diagonalen Gleichungssysteme auf. Es liegt auf der Hand, dass solche Gleichungssysteme mit einer geeigneten Methode wahrscheinlich etwas einfacher zu lösen sind, als ein System mit einer vollbesetzten Koeffizientenmatrix.

Wenn wir uns eine Lösung mit dem normalen Gauss-Algorithmus vorstellen, sehen wir, dass hier eine beträchtliche Anzahl Multiplikationen mit Null erfolgen. Ferner wird in der Speicherung Platz verschwendet, wenn man hierzu eine vollständige *nxn*-Matrix verwendet.

Besser wäre also nur die Diagonale und die Nebendiagonalen in einem geeigneten Array zu speichern.

Wir können den Gauss-Algorithmus für tridiagonale Systeme neu formulieren. Für eine Betrachtung mit einer Koeffizientenmatrix mit zwei Nebendiagonalen n_1, n_3 und ohne Pivotierung erhalten wir:

$$\begin{pmatrix} a_{11} & a_{12} & 0 & & \\ a_{21} & a_{22} & a_{23} & 0 & \\ 0 & \cdots & \cdots & \cdots & 0 \\ & 0 & & a_{n-1,n} & n_3 \\ & & 0 \; a_{n,n-1} & a_{nn} & n_2 \\ & & n_1 & & \end{pmatrix}$$

Wiederhole für Zeile von i=1 bis n	
	Dividiere Zeile i durch a_{i2}
	$a_{i+1,1}$ -faches der Zeile i von der Zeile i+1 subtrahieren
Rückwärts einsetzen	

Dieses Verfahren spart sehr viel Rechenaufwand gegenüber einer Lösung mit dem vollständigen Gauss- Algorithmus. Das Verfahren kann natürlich für Systeme mit mehreren Nebendiagonalen entsprechend erweitert werden.

4.4.1 Iterative Verfahren

Die bisher gezeigten Verfahren bestimmten direkt die exakte Lösung eines linearen Gleichungssystems. Besonders für umfangreiche Gleichungssysteme, die aber nur spärlich besetzte Koeffizientenmatrizen haben, sind mit iterativen Lösungen vorteilhafter zu bearbeiten. Solche Systeme haben grosse Koeffizientenmatrizen mit einer Bandstruktur und sind mit iterativen Verfahren meist schneller zu lösen.

Eine weitere Anwendung ist die Nachiteration. Sie verbessert die Genauigkeit von bereits bestimmten Lösungen aus direkten Verfahren. Man setzt sie z.B. ein um Rundungsfehler zu kompensieren.

Bei einem normalen linearen Gleichungssystem der Form $A \cdot \underline{x} = \underline{b}$ kann die Koeffizientenmatrix A in drei Teile zerlegt werden, die aus einer Summe einer oberen Dreieckmatrix, einer unteren Dreieckmatrix und einer Diagonalmatrix bestehen.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & \cdots & a_{2n} \\ & & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & \cdots & \\ a'_{21} & 0 & \cdots & \\ & & \ddots & \\ a'_{n1} & \cdots & a'_{n,n-1} & 0 \end{pmatrix}}_{\text{L – Matrix}} + \underbrace{\begin{pmatrix} a'_{11} & 0 & \cdots & 0 \\ 0 & a'_{22} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}}_{\text{D – Matrix}} + \underbrace{\begin{pmatrix} 0 & a'_{12} & \cdots & a'_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ & & \ddots & a'_{n-1,n} \\ 0 & 0 & \cdots & 0 \end{pmatrix}}_{\text{R – Matrix}}$$

Das Gleichungssystem kann also auch so geschrieben werden:

$$A \cdot \underline{x} = (L + D + R) \cdot \underline{x} = \underline{b} \tag{4.14}$$

Aus der grossen Menge der iterativen Verfahren betrachten wir stellvertretend das Jacobi-Verfahren.

4.4.1.1 Jacobi-Verfahren

Das Jacobi-Verfahren zeichnet sich durch seine Einfachheit aus. Es wurde vom Mathematiker Carl Gustav Jacobi (1804-1851) entwickelt und beruht auf einer Rekursionsvorschrift. Zuerst wird die A-Matrix wird wie in (4.14) gezeigt in eine Summe zerlegt und wie folgt umgeformt:

$$(L + D + R) \cdot \underline{x} = \underline{b} \qquad \rightarrow \qquad D \cdot \underline{x} = \underline{b} - (L + R) \cdot \underline{x}$$

Der Rekursionsschritt wird dann nach der Vorschrift

$$D \cdot \underline{x}^{[n+1]} = \underline{b} - (L + R) \cdot \underline{x}^{[n]} \tag{4.15}$$

durchgeführt, wobei n die nummer des Rekursionschrittes (als hochgestellter Index) verkörpert.
Wir erhalten die einzelnen Komponenten des Vektors $\underline{x}$:

$$x_i^{[n+1]} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^{z} a_{ij} \cdot x_j^{[n]} \right) \tag{4.16}$$

Als Startvektor $\underline{x}^{[0]}$ kann ein beliebiger Vektor, z.B. der Nullvektor gewählt werden, da das Verfahren falls es konvergiert, für jeden beliebigen Startwert konvergiert.

Nachteilig an dem Verfahren ist seine relativ langsame Konvergenz. So werden meist zwischen 30 - 200 Durchläufe für eine gute Genauigkeit benötigt.

Eine konkrete Implementierung des Algorithmus in C ohne Spezialitäten:

```
/* Iterative Loesung eines linearen Gleichungssystems mit dem Jacobi-Verfahren.

  Zum Projekt gehoerendes File: lgs_read.obj

  Autor: Gerhard Krucker
  Datum: 17.4.1994
  Sprache MS-Visual C V1.5 (QuickWin)
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX 10  /* Maximale Anzahl der Unbekannten (muss gleich wie in lgs_read definiert sein!) */
#define NMAX 500  /* Maximale Anzahl Iterationen */
#define EPS 1E-12 /* Geforderte Genauigkeit der Loesung */

/* Funktionsprototypen für die im externen Modul lgs_read definierten Funktionen */
void LGS_write(double A[MAX][MAX], double b[MAX], int n); /* Ausschreiben des Gl-System auf den Bildschirm */
void LGS_read(double A[MAX][MAX], double b[MAX], int *n);  /* Interaktives Einlesen des Gleichungssystems */

main()
{ double A[MAX][MAX];   /* Koeffizientenmatrix A */
  double b[MAX];        /* Konstantenvektor b */
  double x[MAX];        /* Resultatvektor x[n]  */
  double xx[MAX];       /* Resultatvektor x[n+1] */
  int z;                /* Aktuelle Anzahl Unbekannten (Bestimmungsgleichungen) */
  int i,j;
  int n;                /* Iterationszaehler */
  double d;             /* Differenz zwischen zwei Iterationen */

  printf("Loesung eines linearen Gleichungssystems mit dem Jacobi-Verfahren:\n");
  LGS_read(A,b,&z);    /* Gleichungssystem einlesen und Koeffizienten in A, b speichern */

  LGS_write(A,b,z);    /* System zur Kontrolle ausgeben */

  for (i=0; i < z; i++)    /* Testen ob alle Diagonalelemente != 0 */
    if (A[i][i]==0)
       { fprintf(stderr,"Fehler: Nicht alle Diagonalelemente sind von Null verschieden!\n");
         exit(1);
       }
   n=0;
   for (i=0; i < z; i++)   /* Startvektor setzen, hier den Nullvektor */
       xx[i]=0;
   do
    {  n++;
       for (i = 0; i < z; i++)    /* x^[n+1] wird zu x^[n] */
         x[i]=xx[i];
       for (i = 0; i < z; i++)
         { xx[i] = b[i];
           for (j = 0; j < z; j++)
             if (j != i)
                xx[i] -= A[i][j]* x[j];
           xx[i] /= A[i][i];
           }
        d=0;
        for (i = 0; i < z; i++)
          d += fabs(x[i]-xx[i]);
     } while ((d > EPS) && (n < NMAX));

   printf("Loesung nach %d Iterationen: \n",n);
   for (i=0; i < z;i++)
     printf("x%d: %g\n",i+1,xx[i]);

  return 0;
}
```

```
/* Modul LGS_READ:
   Interaktives Einlesen eines linearen Gleichungssystems und Speichern der Koeffizienten
   in Matrizen.
   Ausgeben des Gleichungssystems aus den Matrizeninhalten auf den Bildschirm.
*/

#include <stdio.h>

#define MAX  10      /* Maximale Anzahl der Unbekannten */

/* Ausgeben des in den Arrays A, b gehaltenen linearen Gleichungssystems
   auf den Bildschirm */
void LGS_write(double A[MAX][MAX], double b[MAX], int n)
{ int i,j;

  for (i=1;i <= n;i++)
    printf("       x%d ",i);
  printf("     |\n");
  for (i=0;i < n;i++)
     printf("-----------");
  printf("-|----------\n");
  for (i=0;i < n;i++)
    { for (j=0;j < n; j++)
        printf("%9.5g ",A[i][j]);
      printf("     | %9.5g\n",b[i]);
     }
   putchar('\n');
}
/* Interaktives, zeilenweises Einlesen des Gleichungsssystems.
   Parameter:
      A: Zeiger auf ein zweidimensionales Array, wo die Koeffizienten ein-
         geschrieben werden.
      b: Zeiger auf ein eindimensionales Array mit den Konstanten
      n: Zeiger auf die Variable für die aktuelle Anzahl Bestimmungsgleichungen.
*/
void LGS_read(double A[MAX][MAX], double b[MAX], int *n)
{  int i,j;

   printf("\nAnzahl Gleichungen? ");
   scanf("%d",n);

   printf("\nMatrix A b zeilenweise eingeben.\n");
   for (i=0; i < *n;i++)
      { printf("Zeile %d? ",i+1);
        for (j=0; j < *n;j++)
           scanf("%lg ",&A[i][j]);
        scanf("%lg",&b[i]);
       }
}
```

Ein Testdurchlauf für das System

$$\begin{pmatrix} 11 & 1 & 0 & 2 \\ -1 & 8 & -3 & 3 \\ 0 & 2 & -5 & 1 \\ 1 & 2 & 3 & 7 \end{pmatrix} \cdot \underline{x} = \begin{pmatrix} 12 \\ -36 \\ 5 \\ 6 \end{pmatrix}$$

liefert das Resultat:

```
Loesung eines linearen Gleichungssystems mit dem Jacobi-Verfahren:
Anzahl Gleichungen? 4

Matrix A b zeilenweise eingeben.
Zeile 1? 11 1 0 2 12
Zeile 2? -1 8 -3 3 -36
Zeile 3? 0 2 -5 1 5
Zeile 4? 1 2 3 7 6
       x1        x2        x3        x4       |
-------------------------------------------------|----------
       11         1         0         2       |         12
       -1         8        -3         3       |        -36
        0         2        -5         1       |          5
        1         2         3         7       |          6

Loesung nach 47 Iterationen:
x1: 1
x2: -7
x3: -3
x4: 4
```

4.4.2 Konvergenzkriterien für das Jacobi-Verfahren

Das Verfahren konvergiert nicht für jede beliebige Koeffizientenmatrix. Nach dem Banachschen Fixpunktsatz erhalten wir ein hinreichendes Kriterium für die Konvergenz, wenn eines der folgenden Kriterien erfüllt ist.

Spaltensummen-Norm: $$\|A\|_S = \max_j \sum_{i=1}^{n} |a_{ij}| \quad < \quad 2|a_{jj}| \tag{4.17}$$

Zeilensummen-Norm: $$\|A\|_Z = \max_i \sum_{j=1}^{n} |a_{ij}| \quad < \quad 2|a_{ii}| \tag{4.18}$$

Ist eines dieser Kriterien erfüllt, spricht man von einer **diagonal dominanten Matrix**.

Das Verfahren kann aber auch ohne diagonal dominante Matrix konvergieren, wie das Beispiel zeigt:

$$\begin{pmatrix} 11 & 1 & 0 & 2 \\ -1 & 4 & -3 & 3 \\ 0 & 2 & -5 & 1 \\ 10 & 2 & 3 & 7 \end{pmatrix} \cdot \underline{x} = \begin{pmatrix} 12 \\ -8 \\ 5 \\ 15 \end{pmatrix}$$

```
Loesung nach 107 Iterationen:
x1: 1
x2: -7
x3: -3
x4: 4
```

Ein Programm mit diesem Lösungsverfahren sollte also vor der Rechnung prüfen, ob die Matrix diagonal dominant ist.

Beispiel :
Untersuchen Sie ob für das folgende Gleichungssystem die Konvergenzbedingung für das Jacobi-Verfahren erfüllt ist.

$$\begin{pmatrix} 2 & 1 & 1 \\ 3 & 6 & 1 \\ 1 & 3 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Lösung:
Die Spaltensummennorm wird:

$$\|A_S\| = \max\{(|2|+|3|+|1|),(|1|+|6|+|1|),(|1|+|1|+|-4|)\} = \max\{6,8,6\} = 8 \quad (\text{max bei } j=2)$$

$$\|A_S\| = 8 < 2a_{22} = 12$$

Die Konvergenzbedingung ist erfüllt.

4.5 Optimale Pivotierung für das Gauss-Jordan-Verfahren

Ein in der Praxis nicht zu vernachlässigendes Problem für eine gute Rechengenauigkeit in den Gauss - Eliminationsalgorithmen ist die Wahl des Pivotelementes.

Das Problem stellt sich vor allem dort, wo einerseits die Rechengenauigkeit beschränkt ist und andererseits das Pivotelement sehr klein ist.

Beispiel:

$$\begin{pmatrix} 0.005 & 1 \\ 1 & 1 \end{pmatrix} \cdot \underline{x} = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix} \qquad \textit{exakt}: \quad x_1 = \frac{100}{199} \approx 0.502\ldots \quad x_2 = \frac{99}{199} \approx 0.497\ldots$$

Die beschränkte Rechengenauigkeit liefert uns eine Näherungslösung des exakten Resultates. Wir nehmen an, dass unser Rechner 2 Mantissenstellen und den Exponenten in normalisierter Darstellung exakt darstellt. Wir betrachten die Lösung des Systems:

a_{11} ist $\neq 0$ und kann somit als Pivot gewählt werden:

$$\left(\begin{array}{cc|c} 0.005 & 1 & 0.5 \\ 0 & -199 & -99 \end{array}\right) \xrightarrow{\textit{Rundung}} \left(\begin{array}{cc|c} 0.005 & 1 & 0.5 \\ 0 & -200 & -99 \end{array}\right) \rightarrow \quad x_1 = 0 \quad x_2 = \frac{99}{200} \approx 0.5$$

Wir sehen den grossen Fehler für x_1, der durch die Rundung entstanden ist.

Besser wäre es gewesen, wenn wir die zweite Zeile als Pivotzeile gewählt hätten. Wir können für die Pivotierung folgende Regeln ausfstellen:

> Notwendige Zeilenvertauschungen $\Leftrightarrow$ Pivotelement = 0
> Ratsame Zeilenvertauschungen $\Leftrightarrow$ $|\text{Pivotelement}| \approx 0$

Für die Wahl des optimalen Pivot sind verschiedene Strategien denkbar. Vom Rundungsfehler her gesehen, hat sich die sog. relative Kolonnenmaximumstrategie bewährt. Sie bestimmt das bestgeeignetste Pivot für den nächsten Eliminationsschritt.

4.5.1 Praktische Wahl des optimalen Pivots

Ausgangslage: Im Eliminationsschritt *j* liege die Matrix $A^{(j-1)}$ vor.
Als Pivot wird dasjenige Element der Spalte *j* gewählt, dessen Beträge relativ zur Summe der Beträge aller Elemente seiner Zeile am grössten ist:

Also:

- $s_i := \sum_{k=j}^{n} \left| a_{ik}^{(j-1)} \right|$ (Summe der Beträge aller Elemente der entsprechenden Zeile) (4.19)
- Bestimme den Index $m \geq j$ so, dass $\frac{a_{mj}^{(j-1)}}{s_m} = \max_{i=j}^{n} \frac{\left| a_{ij}^{(j-1)} \right|}{s_i}$ (4.20)
- Verwende $a_{mj}^{(j-1)}$ als Pivot

Dieses Verfahren liefert gute Resultate wenn tatsächlich ein strenges Maximum vorliegt. Wenn mehrere Werte beim Maximum liegen, sollte noch ein weiteres Kriterium beachtet werden, indem die Matrix für eine weitere Betrachtung geeignet skaliert wird, wie in [5] beschrieben.

Rechenbeispiel für die Wahl des optimalen Pivots für obiges Gleichungssystem:

$$s_1 = \sum_{i=1}^{2} \left|a_{1i}^{(1)}\right| = 0.005 + 1 = 1.005 \quad q_1 = \frac{a_{11}}{s_1} = \frac{0.005}{1.005} \approx 0.00497..$$

$$s_2 = \sum_{i=1}^{2} \left|a_{2i}^{(1)}\right| = 1 + 1 = 2 \qquad q_2 = \frac{a_{21}}{s_2} = \frac{1}{2} = 0.5$$

$$\max_{i=1}^{2}\{|q_i|\} = \max\{0.00497, 0.5\} = 0.5 \qquad \Rightarrow a_{21} \text{ als Pivot verwenden}$$

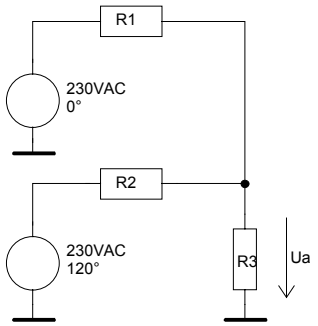
Und die gleiche Rechnung liefert nun bessere Resultate:

$$\begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 0.995 & 0.495 \end{array} \xrightarrow{Rundung} \begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 1 & 0.5 \end{array} \rightarrow \begin{array}{l} x_1 = 0.5 \\ x_2 = 0.5 \end{array}$$

4.6 Lösen linearer Gleichungssysteme mit komplexen Koeffizienten

Dieses etwas von der vorhergehenden Materie isolierte Kapitel soll praktische Hilfestellung geben, wie man Gleichungssysteme mit komplexen Koeffizienten effizient löst.

Illustrationsbeispiel:
Aus dem Dreiphasennetz werden zwei phasenverschobene Spannungen summiert. Wie gross ist die Spannung U_a über dem Widerstand R_3 im Zeitpunkt t_0=0 und zu einem beliebigen Zeitpunkt t_n?
Annahme: Die Spannungsquellen liefern eine Spannung der Form $\hat{U}(cos(\varphi) + j\, sin(\varphi))$.



Welche Lösungswege bieten sich hier grundsätzlich an?

1. Einfache Verfahren, wie Einsetzen, Gleichsetzen, etc. sind für Gleichungssysteme bis mit max. 3 Unbekannten anwendbar. Selbst für Systeme mit 3 Unbekannten wird das Ganze in einer Handrechnung schon recht fehlerträchtig, da alles komplex berechnet werden muss.

2. Über eine Matrizenrechnung. Hier muss aber die Inverse einer komplexen Matrix gebildet werden. Dieses Verfahren scheidet in der Regel für Handarbeit ebenfalls aus. Der Weg ist jedoch mit technischen Hilfsmitteln (Rechner, PC-Programm) der beste Weg. Allerdings sind formale Lösungen schwierig zu erhalten, da die meisten Programme Matrixinversionen nur numerisch berechnen.

3. Umformen des komplexen Gleichungssystems in ein neues Gleichungssystem mit reellen Koeffizienten. Dieses Verfahren ist elegant, verdoppelt aber die Anzahl Bestimmungsgleichungen. Für Handarbeit und in der Programmierung einer Lösung, ist dies jedoch ein durchaus gangbarer Weg.

Die ersten beiden Methoden sind bekannt und werden nicht mehr weiter erläutert. Wir werden jedoch die dritte Methode umfassend vorstellen.

4.6.1.1 Umformen des komplexen Gleichungssystems in ein äquivalentes System mit reellen Koeffizienten

Wesentlich an der Methode ist, dass ein komplexes lineares Gleichungssystem mit *n* (komplexen) Unbekannten in ein äquivalentes lineares Gleichungssystem mit *2n* reellen Koeffizienten und Unbekannten umgeformt wird:

$$\begin{pmatrix} \mathrm{Re}(a_{11}) & -\mathrm{Im}(a_{11}) & \cdots & \mathrm{Re}(a_{1n}) & -\mathrm{Im}(a_{1n}) \\ \mathrm{Im}(a_{11}) & \mathrm{Re}a_{11}) & \cdots & \mathrm{Im}(a_{1n}) & \mathrm{Re}(a_{1n}) \\ \cdots & & & \cdots & \\ \mathrm{Re}(a_{n1}) & -\mathrm{Im}(a_{n1}) & \cdots & \mathrm{Re}(a_{nn}) & -\mathrm{Im}(a_{nn}) \\ \mathrm{Im}(a_{n1}) & \mathrm{Re}(a_{n1}) & \cdots & \mathrm{Im}(a_{nn}) & \mathrm{R\,e}(a_{nn}) \end{pmatrix} \begin{pmatrix} \mathrm{Re}(x_1) \\ \mathrm{Im}(x_1) \\ \cdots \\ \mathrm{Re}(x_n) \\ \mathrm{Im}(x_n) \end{pmatrix} = \begin{pmatrix} \mathrm{Re}(b_1) \\ \mathrm{Im}(b_1) \\ \cdots \\ \mathrm{Re}(x_n) \\ \mathrm{Im}(x_n) \end{pmatrix} \tag{4.21}$$

Begründung:
Die Gleichungen werden zeilenweise vollständig mit ihren Real- und Imaginärteilen ausmultipliziert. Wir erhalten wiederum auf beiden Seiten jeder Zeile Real- und Imaginärteile.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \qquad a_{ii}, x_i, b_i \in \mathbb{C} \tag{4.22}$$

$$\mathrm{Re}(a_{11})\mathrm{Re}(x_1) + j\big(\mathrm{Re}(a_{11})\mathrm{Im}(x_1) + \mathrm{Im}(a_{11})\mathrm{Re}(x_1)\big) - \mathrm{Im}(x_1)\mathrm{Re}(a_{11}) + \cdots - \mathrm{Im}(x_n)\mathrm{Im}(a_{1n}) = \mathrm{Re}(b_1) + j\big(\mathrm{Im}(b_1)\big)$$

$$\vdots$$

$$\mathrm{Re}(a_{n1})\mathrm{Re}(x_1) + j\big(\mathrm{Re}(a_{n1})\mathrm{Im}(x_1) + \mathrm{Im}(a_{n1})\mathrm{Re}(x_1)\big) - \mathrm{Im}(x_1)\mathrm{Re}(a_{n1}) + \cdots - \mathrm{Im}(x_n)\mathrm{Im}(a_{nn}) = \mathrm{Re}(b_n) + j\big(\mathrm{Im}(b_n)\big)$$

Da Gleichheit nur herrscht, wenn Real- und Imaginärteil gleich gross sind, können wir aus jeder Zeile zwei neue Gleichungen formulieren, je eine für den Realteil und eine für den Imaginärteil:

$$\begin{aligned} &\mathrm{Re}(a_{11})\mathrm{Re}(x_1) - \mathrm{Im}(a_{11})\mathrm{Im}(x_1) + \cdots + \mathrm{Re}(a_{1n})\mathrm{Re}(x_n) - \mathrm{Im}(a_{1n})\mathrm{Im}(x_n) = \mathrm{Re}(b_1) \\ &\mathrm{Im}(a_{11})\mathrm{Re}(x_1) + \mathrm{Re}(a_{11})\mathrm{Im}(x_1) + \cdots + \mathrm{Im}(a_{1n})\mathrm{Re}(x_n) + \mathrm{Im}(a_{1n})\mathrm{Re}(x_n) = \mathrm{Im}(b_1) \\ &\qquad\qquad \vdots \\ &\mathrm{Re}(a_{n1})\mathrm{Re}(x_1) - \mathrm{Im}(a_{n1})\mathrm{Im}(x_1) + \cdots + \mathrm{Re}(a_{nn})\mathrm{Re}(x_n) - \mathrm{Im}(a_{nn})\mathrm{Im}(x_n) = \mathrm{Re}(b_n) \\ &\mathrm{Im}(a_{n1})\mathrm{Re}(x_1) + \mathrm{Re}(a_{n1})\mathrm{Im}(x_1) + \cdots + \mathrm{Im}(a_{nn})\mathrm{Re}(x_n) + \mathrm{Im}(a_{nn})\mathrm{Re}(x_n) = \mathrm{Im}(b_n) \end{aligned} \tag{4.23}$$

Beispiel:
Lösung der komplexen Gleichung mit 2 Unbekannten:

$$(1+2j)x+(2+3j)y=(4+1j)$$
$$(7+3j)x+(4+2j)y=(5+4j)$$

Wir setzen in die Matrix 4x4 Matrix ein und rechnen aus:

$$\begin{pmatrix} 1 & -2 & 2 & -3 \\ 2 & 1 & 3 & 2 \\ 7 & -3 & 4 & -2 \\ 3 & 7 & 2 & 4 \end{pmatrix} \begin{pmatrix} x_r \\ x_i \\ y_r \\ y_i \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ 5 \\ 4 \end{pmatrix}$$

Wir bilden dazu die Inverse (mit dem Rechner) und wenden diese in bekannter Manier auf beiden Seiten der Gleichung an:

$$A^{-1} = \frac{1}{314} \begin{pmatrix} -54 & -58 & 61 & 19 \\ 58 & -54 & -19 & 61 \\ 86 & 104 & -39 & -7 \\ -104 & 86 & 7 & -39 \end{pmatrix}$$

$$\begin{pmatrix} x_r \\ x_i \\ y_r \\ y_i \end{pmatrix} = \frac{1}{314} \begin{pmatrix} 107 \\ 327 \\ 225 \\ -451 \end{pmatrix}$$

Damit werden schlussendlich die Lösungen des Gleichungssystems:

$$x = \frac{107 + j327}{314}$$
$$y = \frac{225 - j451}{314}$$

4.7 Aufgaben

Lineare Gleichungssysteme, Determinanten, Matrixinversion

1. Bringen Sie die nachfolgenden Matrizen mit Hilfe des Gauss-Jordan Algorithmus in eine obere Dreieckform:

a.) $A = \begin{pmatrix} -1 & 2 & -3 \\ 2 & 1 & 0 \\ 4 & -2 & 5 \end{pmatrix}$ b.) $A = \begin{pmatrix} 2 & 1 & -1 \\ 0 & 2 & 1 \\ 5 & 2 & -3 \end{pmatrix}$

2. Bestimmen Sie die Determinante der nachfolgenden Matrizen indem Sie den Laplace'schen Entwicklungssatz geeignet anwenden:

a.) $A = \begin{pmatrix} 2 & 0 & -1 \\ 3 & 0 & 2 \\ 4 & -3 & 7 \end{pmatrix}$ b.) $A = \begin{pmatrix} 3 & 2 & -4 \\ 1 & 0 & -2 \\ -2 & 3 & 3 \end{pmatrix}$

3. Bestimmen Sie $|A|$ und adj A der nachfolgenden Matrix:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 1 & 5 & 7 \end{pmatrix}$$

4. Bestimmen Sie die Inversen zu folgenden Matrizen:

a.) $A = \begin{pmatrix} 1 & 2 \\ 3 & \alpha \end{pmatrix}$ b.) $A = \begin{pmatrix} 1 & 3 & 4 \\ 3 & -1 & 6 \\ -1 & 5 & 1 \end{pmatrix}$

5. Bilden Sie die Inversen der folgenden Matrizen. Was ist besonderes festzustellen?

a.) $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$ b.) $\begin{pmatrix} 0 & 1 \\ 8 & 0 \end{pmatrix}$ c.) $\begin{pmatrix} u & 0 & 0 & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & 0 & 0 & x \end{pmatrix}$

6. Für welche Werte von *a* besitzt *A* eine Inverse?

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 2 & -1 & 3 \\ 4 & a & 8 \end{pmatrix}$$

Zeigen Sie zwei grundsätzlich verschiedene Möglichkeiten das *a* zu bestimmen. Welche ist einfacher?

7. Geben Sie die allgemeine Lösung für die (homogenen) Gleichungssysteme:

a.) $\begin{pmatrix} 2 & 4 \\ 1 & 1 \end{pmatrix} x = 0$ b.) $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 2 & 3 & 1 \end{pmatrix} x = 0$

8. Lösen Sie nachfolgende Gleichungssysteme durch Anmultiplizieren der inversen Koeffizientenmatrix:

a.) $\begin{matrix} 2x + y = 7 \\ 3x - 5y = 4 \end{matrix}$ b.) $\begin{matrix} ax - 2by = c \\ 3ax - 5by = 2c \end{matrix}$ wobei $ab \neq 0$

9. Für welche Werte der Parameter α, β hat das folgende Gleichungssystem a.) mehrere, b.) genau eine, c.) keine Lösung?

$$\begin{aligned} x_1 - 3x_2 + 3x_3 &= -1 \\ -x_2 - \alpha x_3 &= 3 \\ 2x_1 + \alpha x_2 + x_3 &= \beta \end{aligned}$$

10. Enthält ein lineares Gleichungssystem weniger Bestimmungsgleichungen als Unbekannte, so beschreibt die allgemeine Lösung eine Basis für einen Lösungsraum (Vektorraum), sofern die Gleichungen widerspruchsfrei sind. Hier können dann (*n*-*k*) Elemente frei gewählt werden, wenn wir bei *n* Unbekannten *k* widerspruchsfreie und unabhängige Bestimmungsgleichungen haben. Eine spezielle Lösung wird dadurch erreicht, indem man für die wählbaren Grössen konkret einen Wert einsetzt und nach den restlichen Unbekannten entwickelt.

    Bestimmen Sie nun den Lösungsraum des folgenden Gleichungssystems und prüfen Sie Ihre Lösung durch Kontrolle mit zwei beliebigen speziellen Lösungen auf numerische Richtigkeit.

$$\begin{aligned} x_1 + 2x_2 - x_3 + 3x_4 + x_5 &= 2 \\ 2x_1 - 4x_2 - 2x_3 + 6x_4 + 3x_5 &= 6 \\ -x_1 - 2x_2 + x_3 - x_4 + 3x_5 &= 4 \end{aligned}$$

11. Lösen Sie folgendes tridiagonales Gleichungssystem:

$$A \cdot \underline{x} = B \qquad \text{mit} \qquad A = \begin{pmatrix} 1 & 3 & 0 & 0 & 0 \\ -3 & -2 & 1 & 0 & 0 \\ 0 & 7 & 2 & 2 & 0 \\ 0 & 0 & 1 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \qquad B = \begin{pmatrix} 7 & 2 \\ -11 & 9 \\ 6 & 15 \\ -6 & -1 \\ 2 & 2 \end{pmatrix}$$

12. Eine Matrix heisst diagonal dominant, falls in jeder Zeile der Betrag des Diagonalelementes grösser ist als die Summe der Beträge der übrigen Matrixelemente derselben Zeile.
Prüfen Sie nachfolgende Matrizen auf diagonale Dominanz und bringen Sie obige Aussage in Relation zur Konvergenzaussage beim Jacobi-Verfahren!

a.) $\begin{pmatrix} 3 & 1 \\ 5 & 4 \end{pmatrix}$ b.) $\begin{pmatrix} 2 & 1 \\ 4 & 5 \end{pmatrix}$ c.) $\begin{pmatrix} 5 & 2 & -4 \\ 1 & 7 & 0 \\ 3 & 2 & 10 \end{pmatrix}$ d.) $\begin{pmatrix} 3 & 1 & 1 \\ 2 & 9 & 5 \\ 2 & 1 & 4 \end{pmatrix}$

13. Welche der Matrizen aus 15. erfüllt demnach das Konvergenzkriterium für das Jacobi-Verfahren?

14. Lösen Sie die nachfolgenden Gleichungssysteme mit optimaler Pivotierung gemäss der relativen Kolonnenmaximumsstrategie. (Die Rechnungsschritte sind mit Ganzzahlarithmetik zu rechnen, d.h. exakt rechnen und nachher auf die nächste Ganzzahl runden)

a.) $\begin{pmatrix} 0.005 & 1 \\ 1 & 1 \end{pmatrix} \underline{x} = \begin{pmatrix} 200 \\ 10 \end{pmatrix}$ b.) $\begin{pmatrix} 10 & -10 \\ 1 & 1 \end{pmatrix} \underline{x} = \begin{pmatrix} 200 \\ 10 \end{pmatrix}$

15. Welche Zeile ist als Pivotzeile für den ersten Eliminationsdurchlauf zu wählen?

a.) $\begin{pmatrix} 0 & 0 & 1 \\ 2 & 1 & 2 \\ 1 & 0 & 2 \end{pmatrix}$ b.) $\begin{pmatrix} 1 & 4 & 2 \\ 3 & 1 & 1 \\ 8 & 3 & 4 \end{pmatrix}$ c.) $\begin{pmatrix} 8 & 2 & 4 \\ 8 & 4 & 2 \\ 8 & 3 & 3 \end{pmatrix}$

16. Eine Möglichkeit eine nichtlineare Ausgleichsfunktion in Form eines Approximationspolynoms zu bestimmen, ist durch Aufstellen eines linearen Gleichungssystems für die Polynomkoeffizienten möglich:
Sei $p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ das Ausgleichspolynom das durch *n+1* Punkte geht und die Funktionswerte in genau diesen Punkten exakt approximiert (interpoliert).
Wir können nun für (*n-1*) Punkte (x_i, y_i) das Gleichungssystem aufstellen: $a_n x_i^n + \cdots + a_2 x_i^2 + a_1 x_i + a_0 = y_i$

Die Unbekannten $a_0 .. a_n$ in dieser Gleichung sind nun durch Lösen des linearen Gleichungssystems zu bestimmen.

Bestimmen Sie nun die Koeffizienten a_i für eine die Polynomfunktion $p_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ deren Werte an den Stellen *x*=-2, -1, 1 , 2 gerade mit der Funktion $f(x) = 2 + \frac{2}{x}$ übereinstimmt. Zeichnen Sie die Originalfunktion und das Ausleichspolynom für x ∈ [-4,4]. Beurteilen Sie rechnerisch die Güte der Ausgleichsfunktion (Interpolationsfunktion).

Programmieraufgaben

17. Implementieren Sie das vereinfachte Gauss-Jordan-Verfahren für tridiagonale Gleichungssysteme.

18. Entwickeln Sie eine Funktion in C die als Parameter eine reguläre quadratische Matrix und die Dimension erhält und als Resultat die Determinante zurückliefert. Die Berechnung soll mittels Gauss-Algorithmus erfolgen.