Technische Universität Ilmenau
Institut für Mathematik

Preprint No. M 99/20

MATLAB - Teil I, Vektoren, Matrizen und lineare Gleichungssysteme

Neundorf, Werner

1999

Impressum:
Hrsg.: Leiter des Instituts für Mathematik
Weimarer Straße 25
98693 Ilmenau
Tel.: +49 3677 69 3621
Fax: +49 3677 69 3270
http://www.tu-ilmenau.de/ifm/

ISSN xxxx-xxxx

ilmedia

Zusammenfassung

This is a tutorial on teaching and programming in MATLAB.
It is based on scripts and exercises in the course of numerical mathematics for students of the faculties Electrical Engineering and Information Technology and Computer Science and Automation after first term.
This part I contains basic aspects and elements of numerical linear algebra, especially methods for systems of linear equations. Further parts give MATLAB-based algorithms and programming tools for eigenvalue problems, nonlinear equations and systems of equations, polynomial and spline interpolations and approximation.

Vorwort

MATLAB is an interactive, matrix-based system for scientific and engineering calculations. You can solve complex numerical problems without actually writing a program. The name MATLAB is an abbreviation for MATrix LABoratory.

A few words to those who are familiar with other programming languages.

- MATLAB is a user-friendly high-level programming language and important technical computing environment. It also includes a number of functional programming constructs, modeling, simulation and prototyping, application development and design.
- MATLAB has a rich environment of powerful toolboxes and data visualization. It offers programming structures like control flows, selections, decisions and m-files functions.
- Students can affordably use this powerful numeric computation, data analysis and visualization software in their undergraduate and graduate studies. The student edition encapsulates a wide range of disciplines.
- MATLAB is not strongly typed like C and Pascal. No declarations are required. It is more like Basic and Lisp in this respect. You can dynamically link C or FORTRAN subroutines. Some type checking is done at run time.
- MATLAB suitable for running numerically intensive programs with double-precision numerical calculations. On the other site there are, based on Maple V, many symbolic tools and symbolic functions to combine, simplify, differentiate, integrate, and solve algebraic and differential equations. The symbolic toolbox and the subroutines concept of MATLAB seems to be not so efficiently as is described in Maple.
- MATLAB is available for a number of environments: Sun/Apollo/VAXstation/HP workstations, VAX, MicroVAX, Gould, PC, Apple Macintosh, and several parallel machines.

The aim is to show how you can write simple instructions, commands or programs in MATLAB for doing numerical calculations, linear algebra, and programs for simplifying or transforming expressions, equations, mathematical formulas or arrays.

It is assumed that the reader is familiar with using MATLAB interactively. For beginners we propose the introductional tutorial, the so called Primer. The purpose of this Primer based on the version 3.5 is to help you begin to use MATLAB. They can best be used hands-on. You are encouraged to work at the computer as you read the Primer and freely experiment with examples.

This document is based on **MATLAB version 4.2c1**.
MATLAB development continues. New versions come out every one or two years which contain not only changes to the mathematical capabilities of MATLAB, but also changes to the programming language and user interface. The MATLAB 5.2 and 5.3 highlights you can find on the web site

`http://www.mathworks.com/products/matlab/highlights.shtml` .

We are pleased and somewhat surprised to see how quickly this movement is already happening. For this reason, I have applied constructs in the language that will be probable in the language in future versions of MATLAB.

You should liberally use the on-line help facility for more detailed information. After entering MATLAB the command `help` will display a list of functions for which on-line help is available. The command `help` *functionname* will give information about a specific function. You can preview some of the features of MATLAB by entering the command `demo`. Even better, access help from the menu.
In the bibliography there are given some useful sources of information and supplemental workbooks for MATLAB.

Inhaltsverzeichnis

1 Vektoren und Matrizen

Hinweise zur Arbeit mit Vektoren und Matrizen findet der Leser ebenfalls in MATLAB-Files oder Demonstrationen wie

```
PC-MATLAB-Verzeichnis   Matlab\Toolbox\Matlab\Demos\*.m
```

```
http://www.mathworks.com/products/demos/
```

Felder können auf verschiedene Art und Weise eingeführt werden:
explizit durch Auflisten der Elemente, mit built-in-Anweisungen und Funktionen, mittels m-Files, durch Laden von externen Dateien.
Die wichtigsten Varianten seien hier vorgestellt.

1.1 Definition von Vektoren

Wir definieren Zeilenvektoren, über die Transposition erhält man entsprechende Spaltenvektoren.

```
% Zeilenvektoren
x = [1 2 3 4]
x = [1, 2, 3, 4]
x = [1; 2; 3; 4]'
x = 1:4
x = 1:1:4
x = (1:4)
x = linspace(1,4,4)
y = []; n =4; for i = 1:n, y = [y i^2]; end, y

% Spaltenvektor
z = x'
```

Der Ergebnis für den Vektor x lautet in jedem Fall

```
x =

 1     2     3     4
```

1.2 Generierung von Matrizen

```
% Explizite Angabe der Elemente

A1 = [ 2 2 3; 3 4 5; 0 1 0 ]

A2 = [ 2 2 3
       3 4 5
       0 1 0 ]
```

```
% Zeilenweise Generierung
A3 = [];
for i = 1:3
 x = [];
 for j = 1:3
   x = [x, (i+j)^2];
 end
 A3 = [A3 ; x];
end
A3

% Hilbertmatrix
for i = 1:3  for j = 1:3  A4(i,j) = 1/(i+j-1); end, end
A4
A5 = hilb(3)
% Belegung der Matrixelemente
for i = 1:3, for j = 1:3,  A6(i,j) = 1/(i+j-1); end, end
A6

% Spezielle Matrizen
% Einheitsmatrix
A7 = eye(3)
% Nullmatrix
A8 = zeros(3) % zeros(3,3), zeros([3,3])
% Einsmatrix
A9 = ones(3)  % ones(3,3), ones([3,3])
% Diagonalmatrix
A10 = diag([1 2 3])
% Magisches Quadrat
A11 = magic(3)
% Zufallsmatrizen
A12 = rand(3) % Zahlen aus [0,1)
A13 = randn(3)

% Blockstrukturierte Matrix
A14 = [A12 zeros(3); ones(3) eye(3)]
% Pascalsches Dreieck als Matrix
A15 = pascal(5)

A16 = [1:3 ; 4:2:8 ; 9:-1:7]
% Diagonale einer Matrix als Spaltenvektor
d = diag(A16)
% Matrix spaltenweise aus Spaltenvektoren
A17 = [ d d d ]
```

Manche LGS haben Koeffizientenmatrizen von tridiagonaler Struktur. Dafür gibt es spezielle Lösungsverfahren, so daß die Tridiagonalmatrix nicht explizit erzeugt werden muß. Trotzdem soll hier beispielhaft ein solches System generiert werden.

```
% Definition eines LGS mit Tridiagonalmatrix und rechter Seite
n = 5
A18 = zeros(n)
a = []; for i=1:n, a = [a -1]; end;
b = []; for i=1:n, b = [b  2]; end;
c = []; for i=1:n, c = [c -1]; end;

A18(1,1)   = b(1);
A18(1,2)   = c(1);
A18(n,n-1) = a(n);
A18(n,n)   = b(n);
for i=2:n-1, A18(i,i-1)= a(i); A18(i,i)= b(i); A18(i,i+1)= c(i); end;
d = []; for i=1:n, d = [d i]; end;
d = d'
A18
```

Wenn Matrix und Vektor im Skript-File (m-File) MATRX4.M vorliegen in der Form

```
% MATRIX4.M

A = [ 1  1  0  1
      2  1 -1  1
     -1  2  3 -1
      3 -1 -1  2 ];

a = [ 2  1  4 -3 ]';
```

dann kann man diese einfach ins aktuelle Skriptfile einbeziehen und verwenden, indem man das Kommando `matrix4` notiert.

Natürlich kann man im MATLAB Command Window über das Menü

File $\Rightarrow$ *Open M–file...* $\Rightarrow$ *Open* $\rightarrow$ *Dateiname*

das m-File auswählen und editieren.
Eine andere Variante wäre die !-Eigenschaft (!-feature), was die direkte Ausführung von Systembefehlen wie Editieren, Kopieren, Drucken usw. erlaubt. Man schreibt also zwecks Editieren von m-Files in einem anderen Fenster nach dem Promptzeichen das MATLAB-Kommando mit dem verfügbaren Editor.

`!ed matrix4.m` bzw.
`!edit matrix4.m` (DOS Window EDIT).

1.3 Teilvektoren und Untermatrizen

Betrachten wir die zuletzt definierten Felder A und a.

```
% Teilvektoren
a(1:2)
a([3,2])
a(3:-1:2)
a(3:2)          % [] leere Liste
b = a(4:-1:1)  % Komponenten in umgekehrter Reihenfolge

% Untermatrizen
A(:,3)          % 3.Spalte
A(1,:)          % 1.Zeile
a1 = A(1,:)
A(:,[2 4])      % Block aus Spalten 2 und 4
B = A(:,2:4)
C = A(2:3,[3,2])
```

Diese Art der Notation verringert die Anwendung von Schleifenkonstrukten, vereinfacht den Programmkode und verbessert somit die Lesbarkeit von Anweisungen mit Feldern.

Der kleinste Index bzw. die unteren Indizes sind immer 1. Das hat zur Folge, daß beim Umspeichern von Feldern in einen "höheren" Indexbereich die Anfangskomponenten zwar belegt sind (meist mit 0), aber eigentlich keine wohl definierten Werte erhalten haben. Die Kommandos

```
d(4:5) = a(1:2)
D(2:5,2:5) = A
```

liefern z.B.

```
d =
    0     0     0     2     1

D =
    0     1     1     0     1
    0     1     1     0     1
    0     2     1    -1     1
    0    -1     2     3    -1
    0     3    -1    -1     2
```

2 Operationen auf Vektoren und Matrizen

Bei einigen Operationen und Funktionen auf Feldern ist zu beachten, daß diese oft für ein vektorielles Argument ausgelegt sind. Die Anwendung auf Matrizen bedeutet zunächst nur die Verarbeitung der einzelnen Matrixspalten (column-by-column fashion). Es entsteht somit ein Zeilenvektor, auf den wiederum operiert werden muß.
Ein typisches Beispiel dafür ist die Funktion `max(x)` für einen Vektor x sowie für eine Rechteckmatrix $A(n, m)$

$$\max(\max(A)) = \max_{1 \le j \le m} \max_{1 \le i \le n} a_{ij}.$$

Für die vertauschte Reihenfolge (row-by-row action) ist die Matrix zu transponieren.
Einige dieser Funktionen sind

`max, min, sort, sum, prod, median, mean, std, any, all.`

Des weiteren beachte man, daß die Funktionen ein oder mehrere Ergebnisparameter haben können.
Die Vorzug von MATLAB liegt gerade in der Leistungsfähigkeit seiner Matrixfunktionen.

2.1 Operationen auf Vektoren

Im weiteren werden für ausgewählte Operationen die Matrix A und Spaltenvektor a aus dem m-File MATRIX4.M verwendet.

```
A = [ 1  1  0  1
      2  1 -1  1
     -1  2  3 -1
      3 -1 -1  2 ];

a = [ 2  1  4 -3 ]';

matrix4
% Transposition
x = eye(3,1)   % 1.Einheitsvektor
y = x'
b = a'
% groesster und kleinster Index der Komponenten
size(a)
max(size(a))
% groesste Komponente
max(a)

em = eye(3,3)
e2 = em(:,2)   % 2.Einheitsvektor
e3 = em(:,3)   % 3.Einheitsvektor
```

```
% Skalarprodukt
s = a'*a

% dyadisches Produkt
p = a*a'
```

Spezielle Operationen auf Vektoren sind Normen.
Sei x ein n-dimensionaler Vektor. Wir betrachten folgende Vektornormen.

```
n = 5
x = 1:n
% Euklidische Norm
norm(x,2)
norm(x)

% Betragssummennorm
norm(x,1)
sum(abs(x))

% Maximumnorm
norm(x,inf)
max(abs(x))
% p-Norm, Hoelder-Norm
p = 7
norm(x,p)
```

2.2 Operationen auf Matrizen

Einfache Operationen (auf die Array-Operatoren wird nicht weiter eingegangen).

```
matrix4
Atrans = A'
% Addition
C = A+Atrans              % Subtraktion analog
% Multiplikation
C = A*Atrans
```

Funktionen über Matrizen.

```
max(A)       % Spaltenmaxima
max(max(A))  % maximales Matrixelement, analog  min
sum(A)       % Spaltensummen, analog  prod
sum(sum(A))  % Summe aller Matrixelemente
sort(A)      % Spalten von A aufsteigend sortiert
all(A)       % Nichtnulltest der Matrixelemente spaltenweise, alle?
any(A)       % Nichtnulltest der Matrixelemente spaltenweise, irgendein?
```

```
% Zeilen- und Spaltenanzahl
size(A)
[n m] = size(A)
C = rand(size(A))   % Zufallsmatrix C mit Dimension von A
% Maximum der Zeilen- oder Spaltenanzahl
length(A)

% Determinante
det(A)
% Inverse
B = inv(A)
A*B    % Kontrolle
% Rang
rank(A)
rank(a)          % =1
rank(a*a')       % =1
% Diagonale von A als Spaltenvektor
diag(A)
% Spur, Summe der Diagonalelemente
trace(A)
sum(diag(A))
```

Man kann nicht nur rechteckige Untermatrizen aus einer gegebenen Matrix A auswählen, sondern auch Band-, Dreiecks- und Hessenbergmatrizen.

```
% Diagonalmatrix mit Diagonale von A
diag(diag(A))

% obere Dreiecksmatrix von A
triu(A)    % triu(A,0)
% untere
tril(A)    % tril(A,0)

% obere Hessenbergform
triu(A,-1)
% untere Hessenbergform
tril(A,1)

% Tridiagonalmatrix mit Subdiagonale, Diagonale und Superdiagonale von A
triu(tril(A,1),-1)
% Matrix A mit Diagonale=0
A-diag(diag(A))
triu(A,1)+tril(A,-1)
```

Spezielle Operationen auf Matrizen sind Normen.
Sei A ein $n \times n$-Matrix. Wir betrachten folgende Matrixnormen. Dazu notieren jeweils die Umschreibung des Kommandos.

```
matrix4
% Spaltensummennorm
norm(A,1)
max(sum(abs(A)))
% Zeilensummennorm
norm(A,'inf')     % auch norm(A,inf)
max(sum(abs(A')))
% Frobeniusnorm
norm(A,'fro')
sqrt(sum(diag(A'*A)))
sqrt(sum(sum(A.^2)))
% Spektralnorm
norm(A,2)
norm(A)
sqrt(max(eig(A'*A)))  %  eig(B) liefert Vektor aller Eigenwerte
```

Der Spektralradius ρ einer (n,n)-Matrix A ist der Betrag ihres betragsmäßig größten Eigenwerts.

```
rho = max(abs(eig(A)))
```

Mittels Normen kann man (relative) Konditionszahlen definieren.

```
% Spektrale Kondition
cond(A)
norm(A)*norm(inv(A))

% Selbstdefinierte Konditionszahlen
% cond_1(A)
norm(A,1)*norm(inv(A),1)
% cond_inf(A)
norm(A,inf)*norm(inv(A),inf)

% Test der Kondition der Hilbertmatrix
for k = 4:7
 cond(hilb(k))
end;
```

Die 4 Konditionszahlen sind

```
ans =
      1.5514e+004
      4.7661e+005
      1.4951e+007
      4.7537e+008
```

Einige Kommandos auf Teilen der Matrix haben wir schon kennengelernt. Nun wollen wir die Operationen erläutern, die für die Zeilen- oder/und Spaltenmanipulation bei der Lösung von LGS gebraucht werden.

```
matrix4
n = length(A)  % Dimension der quadratischen Matrix
AE = [ A a ]   % erweiterte Matrix
size(AE)
ai = A(1,:)    % ai=A(1:1,:)  1.Zeile  auswaehlen
bj = A(:,2)    % bj=A(:,2:2)  2.Spalte
A(:,1:2)       % Spalten 1..2 auswaehlen

% Austauschen von Zeile i und j
% Variante 1
i = 1; j = 2;
AH = A;
for k = 1:n
  h = AH(i,k); AH(i,k) = AH(j,k); AH(j,k) = h;
end
AH
% Variante 2
AH = A;
[ AH(1:i-1,:); AH(j,:); AH(i+1:j-1,:); AH(i,:); AH(j+1:n,:)]

% (Zeile i) * c
c = 2; i = 1;
AH = A;
AH(i,:) = c*AH(i,:);
AH

% (Zeile i) := (Zeile i)+(Zeile j)*c
% Anwendung zur Erzeugung einer Null an der Position (i,1)
AH = A;
c = -0.5; i = 2; j = 1;
AH = A;
AH(i,:) = AH(i,:)+c*AH(j,:);
AH

AH =
      1.0000    0.5000    0.3333
           0    0.0833    0.0833
      0.3333    0.2500    0.2000
```

3 Lösung linearer Gleichungssysteme (LGS)

3.1 Definition der Komponenten eines LGS

Betrachten wir LGS der Gestalt $Ax = b$.

```
A = hilb(3)        % Hilbertmatrix
c = sum(A)         % Spaltensummen von A
b = sum(A')'       % Zeilensummen von A als Spaltenvektor
x = [0 0 0]'
xexakt = [1 1 1]'  % exakte Loesung

r = b-A*xexakt     % Kontrolle des Residuums
AE =[A b]          % erweiterte Matrix
B = eye(3)         % Einheitsmatrix
for i = 1:3, for j = 1:3, AA(i,j) = i+j-2; end, end
AA(3,3) = 5;
AA

% symmetrische positiv definite streng diagonal dominante Matrix
A1 = [ 10  -1   2   0
       -1  11  -1   3
        2  -1  10  -1
        0   3  -1   8 ]

b1 = [  6  25 -11  15 ]'
```

Einige Ausgaben dazu im Worksheet, wobei insbesondere die numerische Berechnung des Residuums r für die exakte Lösung mit den Ungenauigkeiten in der letzten Mantissenstelle des Gleitkommaformats *double* zu beachten ist.

```
A =
     1.0000    0.5000    0.3333
     0.5000    0.3333    0.2500
     0.3333    0.2500    0.2000

b =
     1.8333
     1.0833
     0.7833

r =
     1.0e-015 *
          0
          0
         -0.1110
```

```
AE =
     1.0000    0.5000    0.3333    1.8333
     0.5000    0.3333    0.2500    1.0833
     0.3333    0.2500    0.2000    0.7833

AA =
     0     1     2
     1     2     3
     2     3     5
```

3.2 Numerische und symbolische direkte Lösung von LGS

Die verschiedenen Funktionen zu Lösung von LGS basieren meist auf der Gaußelimination. Dabei werden Pivotstrategien berücksichtigt, die zu Zeilenvertauschungen führen. Letzteres kann durch Ergebnisparameter wie Ausgabe der Permutationmatrix abgefragt werden. Für spezielle Koeffizientenmatrizen, wie z.B. symmetrische oder symmetrische positiv definite (spd), gibt es natürlich spezielle Algorithmen.
Rundungseffekte in der Gleitkommaarithmetik der numerische Rechnung in MATLAB führen anders als bei symbolischen Umformungen in MATLAB oder auch Maple zu abweichenden Zeilenvertauschungen und/oder Zerlegungen der Matrix.
Zunächst diskutieren wir einige Lösungsmöglichkeiten.

3.2.1 Row reduce Variante auf Gauß-Jordan-Form

Die built-in-Funktion `rref` angewendet nur auf die Koeffizientenmatrix kann zu ihrer Rangbestimmung genutzt werden. Ist die Koeffizientenmatrix regulär, so wird diese transformiert auf die Einheitsmatrix. Eventuell notwendige Zeilenvertauschungen sind aus der Darstellung jedoch nicht ersichtlich.

```
rref(AA)

ans =
     1     0     0
     0     1     0
     0     0     1
```

Anwendung für Lösung eines LGS

```
rref(AE)

ans =
     1.0000         0         0    1.0000
          0    1.0000         0    1.0000
          0         0    1.0000    1.0000

x = ans(1:3,4)
```

```
x =
     1.0000
     1.0000
     1.0000

rref([AA [8 14 23]'])     % mit Zeilenvertauschung

ans =
     1     0     0     1
     0     1     0     2
     0     0     1     3

ans(:,4)

ans =
     1
     2
     3
```

3.2.2 Symbolische Lösung

Dazu gibt es das Maple-Kommado `linsolve`.
Die Lösbarkeit des LGS wird geprüft. Mögliche Nachrichten, Warnungen bzw. Fehlermeldungen sind

Warning: Matrix is rank deficient; solution does not exist.
Warning: Matrix is rank deficient; solution is not unique.

```
x = linsolve(A,b)
numeric(x)

x =
  [281474976710655/281474976710656]
  [140737488355331/140737488355328]
  [140737488355325/140737488355328]

ans =
     1.00000000000000
     1.00000000000002
     0.99999999999998

x = linsolve(AA,[8 14 23]')

x =
  [1]
  [2]
  [3]
```

3.2.3 LU-Faktorisierung

Die LU-Faktorisierung ohne Zeilenvertauschung liefert $A = LU$ mit den entsprechen Dreiecksmatrizen L, $l_{ii} = 1$, und U. Zeilenvertauschungen werden entweder in der unteren Dreiecksmatrix L einbezogen oder als Permutationsmatrix in Form eines zusätzlichen Ergebnisparameters angegeben.

$[L,U] = lu(A)$	$A = LU$,	in L stehen die untere Dreiecksmatrix und Permutationsmatrix, U obere Dreiecksmatrix
$[L,U,P] = lu(A)$	$PA = LU$,	L, U untere bzw. obere Dreiecksmatrix, P Permutationsmatrix

$lu(A)$ alleine ergibt das Tableau $C\backslash B$, welches beim verketteten Gaußalgorithmus entsteht, so daß $A = -CB = LU$ ist (output from LINPACK'S ZGEFA routine).

Nehmen wir zunächst den einfachen Fall einer spd streng diagonal dominanten Matrix.

```
A1

A1 =
     10    -1     2     0
     -1    11    -1     3
      2    -1    10    -1
      0     3    -1     8

% Tableau wie beim verketteten Gaussalgorithmus
lu(A1)

ans =
     10.0000   -1.0000    2.0000         0
      0.1000   10.9000   -0.8000    3.0000
     -0.2000    0.0734    9.5413   -0.7798
           0   -0.2752    0.0817    7.1106

[L U P] = lu(A1)

L =
      1.0000         0         0         0
     -0.1000    1.0000         0         0
      0.2000   -0.0734    1.0000         0
           0    0.2752   -0.0817    1.0000

U =
     10.0000   -1.0000    2.0000         0
           0   10.9000   -0.8000    3.0000
           0         0    9.5413   -0.7798
           0         0         0    7.1106
```

```
P =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

[L U] = lu(A1)  % analog ohne Ausgabe von P
```

Die Lösung des LGS kann dann mittels Vorwärts- und Rückwärtsrechnung unter Verwendung des Array-Operators \ erhalten werden.

```
y = L\a1'
x = U\y
x = U\L\a1'
```

Das zweite Beispiel sei ein LGS mit der spd Hilbertmatrix $H(3)$.

```
A =
     1.0000    0.5000    0.3333
     0.5000    0.3333    0.2500
     0.3333    0.2500    0.2000
```

Die symbolische Rechnung und Zerlegung in **Maple** läuft ohne Zeilenvertauschung ab und ergibt die erwartete Zerlegung.

```
A = [ 1    0  0                 # Maple
      1/2  1  0
      1/3  1  1]*[1  1/2   1/3
                  0  1/12  1/12
                  0  0     1/180]
```

MATLAB hingegen führt im 2.Schritt eine Vertauschung von 2. und 3. Zeile durch.

```
format rat      % display rational approximations
lu(A)

ans =
      1           1/2          1/3
    -1/3          1/12         4/45
    -1/2          -1          -1/180

[L U] = lu(A)   % Zeilenvertauschung zu erkennen

L =
      1            0            0
     1/2           1            1
     1/3           1            0
```

```
U =
      1           1/2          1/3
      0           1/12         4/45
      0            0         -1/180

x = (U\L\b)'

x =
      1    1    1
```

Pivotstrategie: im 2.Schritt beim Vergleich von 1/12=0.0833333 in der 2.Zeile mit 1/12 darunter wird die 3.Zeile bevorzugt.
Zur Kontrolle noch einmal mit der Permutationsmatrix.

```
[L,U,P] = lu(A)   % P*A=L*U, U wie oben

L =
      1            0            0
     1/3           1            0
     1/2           1            1
P =
      1            0            0
      0            0            1
      0            1            0
```

Gehen wir zum symbolischen Rechnen über.

```
AS = sym(A)

AS =
  [  1, 1/2, 1/3]
  [1/2, 1/3, 1/4]
  [1/3, 1/4, 1/5]

AS = sym('[1,1/2,1/3;1/2,1/3,1/4;1/3,1/4,1/5]') % Definition mit Elementen

AS =
  [  1, 1/2, 1/3]
  [1/2, 1/3, 1/4]
  [1/3, 1/4, 1/5]

inverse(AS)   % Matrix symbolisch invertieren

ans =
  [  9,  -36,   30]
  [-36,  192, -180]
  [ 30, -180,  180]
```

Die Funktion *lu* ist nicht anwendbar auf die symbolische Matrix AS.
Fehlermeldung: *Error using ⇒ lu*
Matrix must be square.

Um die Zeilenvertauschung bei der Faktorisierung in MATLAB zu unterbinden, vergrößern wir das Element $A(2,2)$ etwas gemäß $A(2,2) = A(2,2) + 1e-16$. Wir erhalten dann ein mit Maple vergleichbares Ergebnis.

```
A =
      1            1/2          1/3
     1/2           1/3          1/4
     1/3           1/4          1/5

[L,U] = lu(A)    % P=I

L =
      1             0            0
     1/2            1            0
     1/3            1            1

U =
      1            1/2          1/3
      0            1/12         1/12
      0             0           1/180
```

Zum Abschluß noch die Faktorisierung der Matrix AA, wobei zu sehen ist, daß die Pivotstrategie mit Zeilenvertauschung sich jeweils das betragsgrößte Element der Spalte sucht.

```
[L,U,P] = lu(AA)  % P*A=L*U

L =
      1             0            0
      0             1            0
     1/2           1/2           1

U =
      2             3            5
      0             1            2
      0             0          -1/2

P =
      0             0            1
      1             0            0
      0             1            0
```

3.2.4 Cholesky-Zerlegung

Das ist die symmetrische Zerlegung $R^T R$, R obere rechte Dreiecksmatrix, der spd Matrix A. Natürlich ist eine Faktorisierung im Komplexen auch denkbar. Dabei entstehen Zeilen mit rein imaginären Elementen.
Weiterhin soll hier ein Hinweis auf das Darstellungsformat `format rat` erfolgen. Reelle Zahlen werden dabei durch Brüche (Zähler und Nenner ganzzahlig) mit einer Genauigkeit von 6-7 Mantissenstellen approximiert und repräsentiert.
Wir verwenden das Cholesky-Verfahren für die spd Hilbertmatrix

$$H(3) = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix},$$

was bei exakter Rechnung die obere Dreiecksmatrix

$$R = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{\sqrt{3}}{6} & \frac{\sqrt{3}}{6} \\ 0 & 0 & \frac{\sqrt{5}}{30} \end{pmatrix} \approx \begin{pmatrix} 1.00000000000000 & 0.50000000000000 & 0.33333333333333 \\ 0 & 0.28867513459481 & 0.28867513459481 \\ 0 & 0 & 0.07453559924999 \end{pmatrix}$$

liefert.

```
format long
R = chol(A)

R =
     1.00000000000000   0.50000000000000   0.33333333333333
                     0   0.28867513459481   0.28867513459481
                     0                   0   0.07453559924999

x = (R\(R'\b))'

x =
     1     1     1

format rat
R1 = chol(A)

R1 =
     1           1/2          1/3
     0         390/1351     390/1351
     0           0          317/4253

R1-R   % ist Nullmatrix im Format rat
L = chol(A)';            % L untere Dreiecksmatrix
```

Man bedenke, daß $R1$ nur die Bruchdarstellung von R ist. Würde man die approximierenden Brüche zur Definition einer neuen Matrix $R2$ verwenden, so wird der Darstellungsfehler von $\approx 1e{-}6$ sichtbar. Im kurzen Format `short` mit 4 Nachkommastellen der Mantisse bleiben diese Abweichungen oft verborgen.

```
R2 = [ 1   1/2      1/3
       0 390/1351 390/1351
       0    0     317/4253 ]

R2 =
     1.00000000000000   0.50000000000000   0.33333333333333
                     0   0.28867505551443   0.28867505551443
                     0                  0   0.07453562191394
```

Die Cholesky-Zerlegung `R = chol(AA)` ist wegen der fehlenden Voraussetzungen zur Matrix AA nicht durchführbar und bringt eine Fehlermeldung.

3.2.5 Invertieren von Matrizen und Lösung mittels Inverser

Wir zeigen mehrere Möglichkeiten zur Invertierung einer Matrix, so daß mit deren Ergebnis die Lösung des LGS dann durch eine Matrix-Vektor-Multiplikation erhalten wird.

```
X1 = inv(A)    % A = Hilbertmatrix H(3)

X1 =
     9.0000  -36.0000   30.0000
   -36.0000  192.0000 -180.0000
    30.0000 -180.0000  180.0000

rref([A eye(3)])

ans =
     1.0000         0         0    9.0000  -36.0000   30.0000
          0    1.0000         0  -36.0000  192.0000 -180.0000
          0         0    1.0000   30.0000 -180.0000  180.0000

X2 = ans(1:3,4:6)

X2 =
     9.0000  -36.0000   30.0000
   -36.0000  192.0000 -180.0000
    30.0000 -180.0000  180.0000

x = (inv(A)*b)'    % aufwendiger als x=A\b

x =
     1.0000    1.0000    1.0000
```

```
X4 = inverse(A)    % symbolisch

% Bruchdarstellung sehr lang!
X4 =
[   422550200076077968365443525 29/469500222306761221183170587 3,
      -5070602400912910850587371765 76/1408500666920283663549511761 9,
          1408500666920249886552306483 20/469500222306761221183170587 3 ]
[-5070602400912910850587371765 76/1408500666920283663549511761 9,
       8112963841460664116339235880960/42255020007608509906485352857,
         -2535301200456458802993406410752/1408500666920283663549511761 9 ]
[  1408500666920249886552306483 20/469500222306761221183170587 3,
      -2535301200456458802993406410752/1408500666920283663549511761 9,
          8451004001521544355310112604 16/469500222306761221183170587 3 ]

% Kontrolle zum Element an der Position (1,1)
9-422550200076077968365443525 29/469500222306761221183170587 3

ans =
     1.527666881884215e-013

X3 = inv(AA)

X3 =
     -1    -1     1
     -1     4    -2
      1    -2     1

AA*X3    % Kontrolle
```

3.2.6 Lösung mittels LU-Faktorisierung

Hier kommen die für MATLAB typischen Matrixoperatoren \ und / zur Anwendung.

$x = A\backslash b$ ist die Lösung von $Ax = b$,
$x = b/A$ ist die Lösung von $xA = b$.

In der Links-Division \ wird die quadratische Matrix mittels Gaußelimination faktorisiert und damit dann das LGS $Ax = b$ gelöst.
Für eine rechteckige Matrix A liegt die Householder-Orthogonalisierung zugrunde, und man sucht die Lösung im Sinne der Methode der kleinsten Quadrate (least squares sense). Rechts- und Links-Division sind ineinander überführbar.

$$b/A = (A'\backslash b')', \quad A\backslash b = (b'/A')'.$$

Die genannten Divisionen sind im Vergleich zu anderen Verfahren zu empfehlen, weil sie im allgemeinen genauer und schneller sind.

```
x = A\b

x =
     1.0000
     1.0000
     1.0000

X1 = A\eye(3)   % Inverse

X1 =
     9.0000  -36.0000   30.0000
   -36.0000  192.0000 -180.0000
    30.0000 -180.0000  180.0000
```

3.2.7 Anwendung des Kommandos *solve* für symbolisches Rechnen

Das Kommando `solve` wird schrittweise durch einige kleinere Demonstrationen eingeführt.

```
x = solve('x^2-1')  % Aufloesung nach x
x =
  [ 1]
  [-1]

[u,v,w] = solve('w-u=0','w*u=1','u+v=1','u,v,w')
u =
  [ 1]
  [-1]
v =
  [0]
  [2]
w =
  [ 1]
  [-1]

x = solve('A(1,1)*x+A(1,2)*y+A(1,3)*z-b(1)=0')   % Aufloesung nach x

x =
     -(A(1,2)*y+A(1,3)*z-b(1))/A(1,1)

[x,y] = solve('x+y=2','x-y')   % Aufloesung nach x,y

x =
     1
y =
     1
```

```
  [x,y] = solve('A(1,1)*x+A(1,2)*y=1','x+y=1','x,y')
  x =
       -(-1+A(1,2))/(A(1,1)-A(1,2))
  y =
       1/(A(1,1)-A(1,2))*(A(1,1)-1)
% folgendes etwas komplizierter
[x,y] = solve('A(1,1)*x+A(1,2)*y+A(1,3)*z-b(1)=0',
              'A(2,1)*x+A(2,2)*y+A(2,3)*z-b(2)=0','x,y')

x =
 -(-A(2,3)*z*A(1,2)+b(2)*A(1,2)+A(2,2)*A(1,3)*z-A(2,2)*b(1))/
                               (-A(1,2)*A(2,1)+A(1,1)*A(2,2))
y =
 -1/(-A(1,2)*A(2,1)+A(1,1)*A(2,2))*(-A(1,3)*z*A(2,1)+
               b(1)*A(2,1)+A(1,1)*A(2,3)*z-A(1,1)*b(2))

[x,y,z] = solve('A(1,1)*x+A(1,2)*y+A(1,3)*z-b(1)=0',
                'A(2,1)*x+A(2,2)*y+A(2,3)*z-b(2)=0',
                'A(3,1)*x+A(3,2)*y+A(3,3)*z-b(3)=0','x,y,z')

x =
 (A(2,2)*A(1,3)*b(3)-A(2,2)*b(1)*A(3,3)+A(2,3)*A(3,2)*b(1)-A(2,3)*b(3)*A(1,2)-
  b(2)*A(3,2)*A(1,3)+b(2)*A(3,3)*A(1,2))/(A(3,1)*A(2,2)*A(1,3)+A(3,2)*A(1,1)*A(2,3)-
  A(3,2)*A(2,1)*A(1,3)-A(3,1)*A(2,3)*A(1,2)-A(3,3)*A(1,1)*A(2,2)+A(3,3)*A(2,1)*A(1,2))
% analog y,z

% Defintion der Gleichungen
Eq1 = 'A(1,1)*x1+A(1,2)*x2+A(1,3)*x3-b(1)=0';
Eq2 = 'A(2,1)*x1+A(2,2)*x2+A(2,3)*x3-b(2)=0';
Eq3 = 'A(3,1)*x1+A(3,2)*x2+A(3,3)*x3-b(3)=0';

[x1,x2,x3] = solve(Eq1,Eq2,Eq3,'x1,x2,x3')

x1 =
 (A(2,2)*A(1,3)*b(3)-A(2,2)*b(1)*A(3,3)+A(2,3)*A(3,2)*b(1)-A(2,3)*b(3)*A(1,2)-
  b(2)*A(3,2)*A(1,3)+b(2)*A(3,3)*A(1,2))/(A(3,1)*A(2,2)*A(1,3)+A(3,2)*A(1,1)*A(2,3)-
  A(3,2)*A(2,1)*A(1,3)-A(3,1)*A(2,3)*A(1,2)-A(3,3)*A(1,1)*A(2,2)+A(3,3)*A(2,1)*A(1,2))

x2 =
 (-A(2,1)*A(1,3)*b(3)+A(2,1)*b(1)*A(3,3)-b(1)*A(3,1)*A(2,3)+A(1,1)*A(2,3)*b(3)+
   A(1,3)*A(3,1)*b(2)-A(1,1)*b(2)*A(3,3))/(A(3,1)*A(2,2)*A(1,3)+A(3,2)*A(1,1)*A(2,3)-
   A(3,2)*A(2,1)*A(1,3)-A(3,1)*A(2,3)*A(1,2)-A(3,3)*A(1,1)*A(2,2)+A(3,3)*A(2,1)*A(1,2))

x3 =
 -1/(A(3,1)*A(2,2)*A(1,3)+A(3,2)*A(1,1)*A(2,3)-A(3,2)*A(2,1)*A(1,3)-A(3,1)*A(2,3)*A(1,2)-
  A(3,3)*A(1,1)*A(2,2)+A(3,3)*A(2,1)*A(1,2))*(-A(3,1)*A(2,2)*b(1)-A(3,2)*A(1,1)*b(2)+
  A(3,2)*A(2,1)*b(1)+A(3,1)*b(2)*A(1,2)+b(3)*A(1,1)*A(2,2)-b(3)*A(2,1)*A(1,2))
```

3.2.8 Funktion analog zum Kommando *linsolve*

Der Funktionskopf des entsprechenden m-Files *gaussel.m* ist

`function [B,r] = gaussel(A).`

Das Eliminationsverfahren erzeugt eine obere Dreiecksmatrix B (nach ev. Zeilenvertauschungen) und den Rang der Matrix. Die einfache Spaltenpivotisierung wird gemäß $A(r,r) \neq 0$ durchgeführt.
Für eine reguläre quadratische Matrix A kann die Funktion dann zur Lösung des LGS mit oberer Dreiecksmatrix genutzt werden. Für eine beliebige (n,m)-Rechteckmatrix bedeutet es i.a. nur eine Transformation auf die obere Dreiecksmatrix mit der Bestimmung des Rangs $r \leq \min(n,m)$.

```
% gaussel.m
%
function [B,r] = gaussel(A)
% gaussel returns the transformed rectangular matrix
% as upper triangular matrix B (after possible row exchanges,
% pivot strategy with A(r,r)<>0 ) with the rank r<=min(n,m).
% Use for solving Ax=b or transformation of A.
%
[n m] = size(A);  % row and column number of A
B = A;
r = 1;  % row index
c = 1;  % column index
while (c<=m) & (r<=n)
  i = r;
  % find pivot element in column c
  while  (i<=n)   % nicht "for i = r:n"  nehmen, da nach
                  % Laufanweisung stets i<=n ist (also i~=n+1),
                  % anders als in Maple
    if B(i,c) ~= 0, break, end;
    i = i+1;
  end;
  if (i<=n)       % pivot element exists
    if (i~=r)     % row exchange
      for j = c:m
        t = B(i,j); B(i,j) = B(r,j); B(r,j) = t;
      end;
    end;
    % transform rest table
    for i = r+1:n
      if B(i,c) ~= 0
        t = B(i,c)/B(r,c);
        for j = c+1:m, B(i,j) = B(i,j)-t*B(r,j); end;
        B(i,c) = 0;
```

```
      end;
    end;
    r = r+1;
  end;
  c = c+1;
end;
r = r-1;
% end of function gaussel
```

Einige Aufrufe der Funktion.

```
  gaussel(A)

  ans =
       1.0000    0.5000    0.3333
            0    0.0833    0.0833
            0         0    0.0056

  [B,r] = gaussel(A)

  B =
       1.0000    0.5000    0.3333
            0    0.0833    0.0833
            0         0    0.0056

  r =
       3
```

Transformation der erweiterten Matrix mit Lösung des Dreieckssystems.

```
  [n m] = size(AE)
  [B,r] = gaussel(AE)

  B =
       1.0000    0.5000    0.3333    1.8333
            0    0.0833    0.0833    0.1667
            0         0    0.0056    0.0056
  r =
       3

  if r==n
    for i = n:-1:1
      h = B(i,n+1);
      for j = i+1:n
        h = h - x(j)*B(i,j);
      end;
      x(i) = h/B(i,i);
```

```
  end;
  disp('Loesung des Gleichungssystems')
  x
else
  sprintf('Abbruch wegen Rang r = %2i < %2i = n',r,n)
end;
```

Weitere Beispiele für Matrixtransformationen.

```
[B,r] = gaussel(AA)  % mit Zeilenvertauschung

B =
     1     2     3
     0     1     2
     0     0     1
r =
     3

AB = [ 0  1  1  1
       0  2  3  4
       0  0  6  5 ];
[B,r] = gaussel(AB)

B =
     0     1     1     1
     0     0     1     2
     0     0     0    -7
r =
     3

AB = [ 0  0  1  1  1
       0  0  2  3  4
       0  0  0  6  5 ]
[B,r] = gaussel(AB)

B =
     0     0     1     1     1
     0     0     0     1     2
     0     0     0     0    -7
r =
     3

% Transformation von Rechteckmatrizen
gausses([ A; b'; 2*b' ])
gausses([ A b 2*b ])
```

3.3 Fehlerabschätzungen

Es gibt eine Fülle von Varianten der Genauigkeitsbewertung von Ergebnissen der numerischen Rechnung. Sie beinhalten Abschätzungen für den Residuenvektor (Bildvektordifferenz) bzw. Forderungen an den absoluten oder relativen Fehler des Lösungsvektors. Letzteres passiert im Rahmen der sogenannten Rückwärtsanalyse (backward analysis), wo $x + \Delta x$ als als exakte Lösung des leicht gestörten Systems

$$(A + \Delta A)\, x' \;=\; b + \Delta b$$

betrachtet wird.
(1) Die Größe des *Residuenvektors* $r = A\tilde{x} - b$ der Näherung $\tilde{x}$ führt über die Gleichung $\Delta x = \tilde{x} - x = A^{-1}r$ (Δx wird auch Urbildfehler genannt) und Normabschätzungen auf die Beziehung

$$\|\Delta x\| \le \|A^{-1}\|\|r\| \le \frac{\|C\|}{1 - \|CA - I\|}\|r\|,$$

wobei C genäherte Inverse von A mit $\|CA - I\| < 1$ ist und $S = CA - I$ die zu C gehörige Störmatrix (Restmatrix) sind.
Dabei tritt die absolute Konditionszahl $acond(A) = \|A^{-1}\|$ auf.
Ist die Störmatrixnorm $\|S\| > 1$, so ist die Berechnung von C offenbar mit so großen Fehlern behaftet, daß diese mit numerisch stabileren Algorithmen und/oder erhöhter arithmetischer Genauigkeit notwendig ist.

(2) Mit obiger Beziehung und $\|b\| \le \|A\|\|x\|$ erhält man

$$\frac{\|\Delta x\|}{\|x\|} \le \|A\|\|A^{-1}\|\frac{\|r\|}{\|b\|} = cond(A)\frac{\|r\|}{\|b\|}.$$

Beides sind Abschätzungen für den absoluten bzw. relativen Fehler. Sie bedeuten, daß neben einem kleinen Residuenvektor r die Elemente der inversen Matrix bzw. die Kondition der Matrix ausschlaggebend für den Fehler der Näherung $\tilde{x}$ sind. Kleines Residuum heißt also nicht automatisch kleiner Lösungsfehler.

Darüber hinaus treten die Störmatrix S und die Näherungsinverse C als erzeugende Matrix in der üblichen Nachiteration (das ist eine Fixpunktiteration)

$$x^{(m+1)} = x^{(m)} - C(Ax^{(m)} - b) = -Sx^{(m)} + Cb$$

für die Fehlerverbesserung auf, wo die Störmatrixnorm im wesentlichen die Kontraktionskonstante für die Konvergenz darstellt.
Will man die Nachiteration auf die Verbesserung der Lösung eines direkten Verfahrens anwenden, so ist zu beachten, daß dies im allgemeinen nur Sinn macht mit Erhöhung der Genauigkeit der Gleitkommaarithmetik.

3.3.1 Residuum bei schlecht konditionierten LGS

Äquilibrierte Matrizen mit schlechter Kondition zeichnen sich dadurch aus, das ihre Inverse betragsgroße Elemente hat oder die Determinante einer von beiden fast Null ist.
Bei der Berechnung des Residuums spielt dann in MATLAB noch die Reihenfolge der Auswertung der Formel eine Rolle.

```
format long e
A = [ 1 -0.01; -0.01 0 ];
b = [ 1.1  -2.1 ]';
xexakt = [ 210  20890 ]';

inv(A)

ans =
     -2.081668171172168e-017   -1.000000000000000e+002
     -1.000000000000000e+002   -1.000000000000000e+004

inv(A)*A    % Kontrolle

ans =
     1.000000000000000e+000    2.081668171172168e-019
     2.081668171172168e-015    1.000000000000000e+000

xcomp = A\b

xcomp =
        210
      20890

r = b-A*xcomp;

r =
     4.440892098500626e-015
     0

% 2. Auswertungfolge
-A(1,1)*xcomp(1)-A(1,2)*xcomp(2)+b(1)

ans =
     5.773159728050813e-015

-A(2,1)*xcomp(1)-A(2,2)*xcomp(2)+b(2)

ans =
     0

% 3. Auswertungfolge
b(1)-A(1,1)*xcomp(1)-A(1,2)*xcomp(2)

ans =
     0
```

```
b(2)-A(2,1)*xcomp(1)-A(2,2)*xcomp(2)

ans =
     0
```

Nun vergleichen wir die Computerlösung mit einem weiteren Vektor, der eventuell als Kandidat für die Lösung in Frage kommen könnte.

```
format short
A = [ 5       10
      5.00001 10 ];   % schlecht konditioniert
b = [ 15 15.00001]';
xacc = [ 1 1 ]';      % exakte Loesung

% Computerloesung
x = A\b

x =
     1.0000
     1.0000

r = b-A*x

r =
     0
     0

% zu untersuchender Kandidat fuer Loesung
xest=[3 0]';
r = b - A*xest   % kleines Residuum

r =
    1.0e-004 *
        0
       -0.2000

e = xacc - xest  % grosser Abstand der Vektoren

e =
     -2
      1
```

Wenn die Konditionszahl $cond(A)$ nahe der 1 ist, dann ist der relative Fehler von x ungefähr von der gleichen Größe wie das relative Residuum. Ansonsten gilt die Beziehung

$$\frac{r_1}{cond(A)} \leq e_1 \leq r_1 \; cond(A), \quad r_1 = \frac{\|r\|}{\|b\|}, \; e_1 = \frac{\|\Delta x\|}{\|x\|}.$$

```
Ainv = inv(A)

Ainv =
  1.0e+005 *
     -1.0000    1.0000
      0.5000   -0.5000

% Konditionszahlen
c1 = norm(A,1)*norm(Ainv,1)      % Spaltensummennorm

c1 =
     3.0000e+006

c2 = norm(A,2)*norm(Ainv,2)      % Spektralnorm

c2 =
     2.5000e+006

cinf = norm(A,inf)*norm(Ainv,inf) % Zeilensummennorm

cinf =
     3.0000e+006

% relative Fehler
r1 = norm(r)/norm(b)     %  Euklidische Norm

r1 =
     9.4281e-007

e1 = norm(e)/norm(xacc)

e1 =
     1.5811

s1 = sprintf('r1/cond(A) = %6.4e',r1/cond(A));
s2 = sprintf(' <= e1 = %6.4f',e1);
s3 = sprintf(' <= %6.4f = r1*cond(A)',r1*cond(A));
s = [s1 s2 s3];
disp(s)

r1/cond(A) = 3.7712e-013 <= e1 = 1.5811 <= 2.3570 = r1*cond(A)
```

3.4 Komplexität und Geschwindigkeitstest

Zur Einschätzung der Effizienz von Algorithmen bietet MATLAB mehrere Möglichkeiten. Zwei Varianten sind die Bestimmung der Anzahl der durchgeführten Gleitpunktoperationen mit `flops` (floating point operations) sowie die Zeitmessungen mit `clock, etime` und `tic, toc`.
Das Kommando `flops(0)` (nicht `flops = 0`) setzt den Zähler auf Null zurück. So kann die Eingabe von `flops(0)` unmittelbar vor dem Beginn des Algorithmus und der Aufruf `flops` gleich nach seiner Beendigung die *flops* ermitteln.

```
n = 20;
A = rand(n);     % Zufallsmatrix
b = rand(n,1);   % Zufallsspaltenvektor
% Anzahl der Operationen flops
flops(0);
A\b;
flops

ans =
     8034
```

Die MATLAB Funktion `clock` gibt die aktuelle Zeit mit der Genauigkeit auf eine Hundertstel Sekunde an. Mit zwei solchen Zeiten kann `etime` die abgelaufene Zeit (Zeitdifferenz) in Sekunden bestimmen.
Seit der Version 4.0 gibt es die bequemere Variante einer Stoppuhr mit `tic, toc`.

```
% Zeitmessungen in Sekunden mit clock/etime bzw. tic/toc
t0 = clock;
A\b;
t = etime(clock,t0)

t =
     0.0600

tic
A\b;
toc

elapsed_time =
     0.0600

tic
A\b;
t = toc

t =
     0.0600
```

Man beachte, daß bei timesharing Arbeit am Rechner das Ergebnis der Zeitmessung natürlich verfälscht sein kann. Rechnungen an einem separaten 486er PC 66MHz zeigen, daß dies bei Arbeit im Netz mit dem PC (hier Pentium II 350MHz) bis zu ca 10% mehr Rechenzeit ausmacht.

3.4.1 Komplexität von Algorithmen zur Lösung von LGS

Wir vergleichen hier die Operationszahlen und Rechenzeiten von 5 Algorithmen. Die Rechnungen wurden auf einem 486er Einzel-PC 66MHz sowie einem PC im Netz mit Pentium II Prozessor 350MHz durchgeführt.

$Ax = b$, AB erweiterte Koeffizientenmatrix

$[L, U] = lu(A)$; $U\backslash(L\backslash b)$

$AT = gaussel(AB)$; x mit Rückwärtseinsetzen

$A\backslash b$

$rref(AB)$

$inv(A) * b$

Die verschiedenen Ergebnisse werden so abgespeichert, um sie u.a. als vergleichendes Balkendiagramm darzustellen.

```
title('SPEEDTEST for SOLUTION of Ax=b for n = 20x20 .. 400x400 ')
clear n t s sn ns tn nt q tt ss
echo off
h = 1;     % Skalierung des Balkenabstandes
na = 20;
nsw = 20;
ne = 200;
for n = na:nsw:ne
  A = rand(n)+n*eye(n);
  b = rand(n,1);
  AB = [A b];
  %
  t0 = clock;
  flops(0);
  [L U] = lu(A);
  U\(L\b);
  t(n) = etime(clock,t0);
  s(n) = flops;
  q(n) = 0;
  if (t(n)~=0), q(n) = s(n)./t(n); end;
  %
  t0 = clock;
  flops(0);
  AT = gaussel(AB);
  for i = n:-1:1
```

```
    hh = AT(i,n+1);
    for j = i+1:n
      hh = hh - x(j)*AT(i,j);
    end;
    x(i) = hh/AT(i,i);
  end;
  t(n+2*h) = etime(clock,t0);
  s(n+2*h) = flops;
  q(n+2*h) = 0;
  if (t(n+2*h)~=0), q(n+2*h) = s(n+2*h)./t(n+2*h); end;
  %
  t0 = clock;
  flops(0);
  A\b;
  t(n+4*h) = etime(clock,t0);
  s(n+4*h) = flops;
  q(n+4*h) = 0;
  if (t(n+4*h)~=0), q(n+4*h) = s(n+4*h)./t(n+4*h); end;
  %
  t0 = clock;
  flops(0);
  rref(AB);
  t(n+6*h) = etime(clock,t0);
  s(n+6*h) = flops;
  q(n+6*h) = 0;
  if (t(n+6*h)~=0), q(n+6*h) = s(n+6*h)./t(n+6*h); end;
  %
  t0 = clock;
  flops(0);
  inv(A)*b;
  t(n+8*h) = etime(clock,t0);
  s(n+8*h) = flops;
  q(n+8*h) = 0;
  if (t(n+8*h)~=0), q(n+8*h) = s(n+8*h)./t(n+8*h); end;
  %
%  t0=clock;
%  flops(0);
%  Maple-Prozedur, symbolisch
%  linsolve(A,b);   % Warnung, string zu lange in Maple
%  t(n+10*h) = etime(clock,t0);
%  s(n+10*h) = flops;
%  q(n+10*h) = 0;
%  if (t(n+10*h)~=0), q(n+10*h) = s(n+10*h)./t(n+10*h); end;
  end
```

```
[ns,sn] = bar(s);
plot(ns,sn,'g')
xlabel('dimension n')
ylabel('flops')
print graph31.ps -dps
pause
[nt,tn] = bar(t);
plot(nt,tn)
xlabel('dimension n')
ylabel('elapsed time in sec')
print graph32.ps -dps
pause
bar(q)
xlabel('dimension n')
ylabel('quotient of flops / time(sec) ')
print graph33.ps -dps
pause
clf

disp(' ')
disp(tt)
pause
disp('flops')
disp('           n          lu     gaussel         A\b        rref         inv')
ss = zeros(round((ne-na)/nsw)+1,6);
for j=na:nsw:ne
  ss(round((j-na)/nsw)+1,1) = j;
  for k=0:2:8
    ss(round((j-na)/nsw)+1,round(k/2)+2) = s(j+k*h);
  end
end;
disp(ss)

disp('times')
disp('         n        lu   gaussel       A\b      rref       inv')
tt = zeros(round((ne-na)/nsw)+1,6);
for j=na:nsw:ne
  tt(round((j-na)/nsw)+1,1) = j;
  for k=0:2:8
    tt(round((j-na)/nsw)+1,round(k/2)+2) = t(j+k*h);
  end
end;
```

Bei der Anzahl der Gleitpunktoperationen werden alle arithmetischen Operationen extra gezählt. Die sonst übliche Angabe des Aufwandes für die Gaußelimination mit $\mathcal{O}(\frac{n^3}{3} + n^2 - \frac{n}{3})$ im Operationsmix $(+, *)$ muß also verdoppelt werden.

Somit ergibt sich die Größenordnung $\mathcal{K} = \mathcal{O}(\frac{2}{3}n^3)$ für die Verfahren *lu*, *gaussel* und $A\backslash b$. Der Zugang über die inverse Matrix hat die Komplexität $\mathcal{O}(\frac{n^3}{3} + n \cdot n^2 - \frac{n}{3})$ (n rechte Seiten), was hier auf die Größenordnung $\mathcal{O}(2n^3)$ führt.

Ausgewählte Ergebnisse für 486er PC.

```
title('SPEEDTEST for SOLUTION of Ax=b for n = 20x20 (20x20)) 200x200')
flops
```

n	lu	gaussel	A\b	rref	inv
20	6028	6221	8020	27747	18510
40	45238	46021	53466	132595	138046
60	149648	151421	168298	357901	454568
80	351258	354421	384610	750222	1064170
100	682068	687021	734244	1351968	2062694
120	1174078		1249534		3546474
140	1859288		1962140		5611170
160	2769698		2904036		8352756
180	3937308		4107254		11867264
200	5394118		5603902		16250802

```
times
```

n	lu	gaussel	A\b	rref	inv
20.0000	0	0.9900	0	5.7700	0
40.0000	0	6.4800	0	20.8700	0.0600
60.0000	0.0600	20.4800	0.1100	44.5500	0.1600
80.0000	0.1700	48.1100	0.2200	77.6100	0.4400
100.0000	0.3300	89.5300	0.3300	118.5300	0.8200
120.0000	0.4900		0.4900		1.3700
140.0000	0.7700		0.7100		2.0800
160.0000	1.0500		1.1500		3.0300
180.0000	1.7000		1.6400		4.2900
200.0000	2.3600		2.2500		6.1500

Die Kommandos *lu* und $A\backslash b$ zeigen sowohl für *flops* als auch in der Rechenzeit $times = T$ ähnliches Verhalten. Entsprechend seiner Komplexität ist *inv* dreimal schlechter. Daß *gaussel* trotz gleicher *flops* wie *lu* in der Rechenzeit deutlich schlechter ausfällt, liegt also nicht hauptsächlich am timesharing Modus des Rechners. Es ist zu vermuten, daß auch die Struktur der einfachen Laufanweisungen in der Funktion *gaussel* zu Rechenzeitverlusten führt gegenüber solchen, die eventuell in den anderen Kommandos Teilvektoren und Untermatrizen nutzt und eine schnelle Adressenmanipulation beim Zugriff auf Feldkomponenten realisiert.
Auch *rref* wird im Vergleich mit *gaussel* trotz größerer *flops* bezüglich der Rechenzeit immer günstiger, kann aber bei weitem nicht mit *lu* konkurrieren.
Die Gaußelimination ohne Pivotstrategie (Funktion *gausseo*) zeigt ähnliche Komplexität wie *gaussel*. Das unterstreicht, daß die Auswertung von Bedingungen/Tests größenordnungsmäßig die Komplexität nicht beeinflußt.

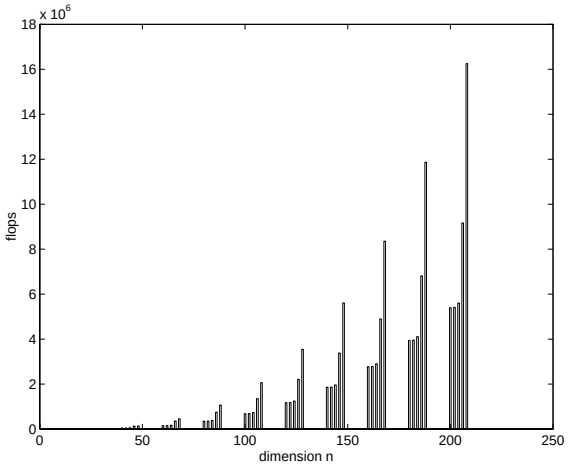
Ergebnistableau für PC Pentium II: *flops*.

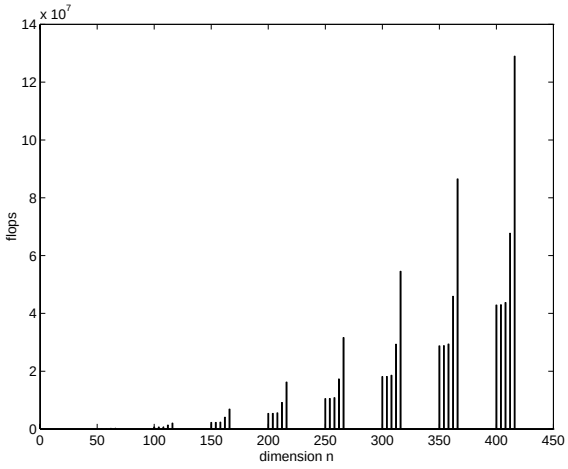
n	lu	$gaussel$	$A\backslash b$	$rref$	inv
20	6028	6221	8028	27435	18518
40	45238	46021	53482	132595	138062
60	149648	151421	168380	357758	454650
80	351258	354421	384674	748467	1064234
100	682068	687021	734390	1349225	2062840
120	1174078	1181221	1249514	2214142	3546454
140	1859288	1869021	1962066	3378712	5611096
160	2769698	2782421	2903782	4896733	8352502
180	3937308	3953421	4107164	6812084	11867174
200	5394118	5414021	5604018	9170035	16250918
250	10511393	10542521	10840204	17335976	31642579
300	18136168	18181021	18609462	29346143	54564812
350	28768443	28829521	29412540	45912719	86518365
400	42908218	42988021	43750234	67730467	129004034
$\mathcal{K}$	$\frac{2}{3}n^3$				$2n^3$

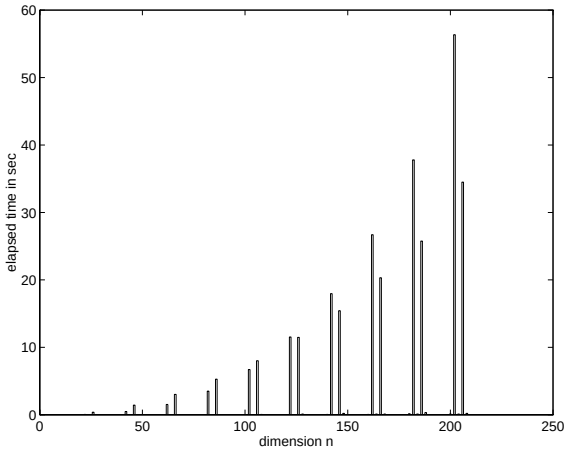
Ergebnistableau für PC Pentium II: *times in sec*.

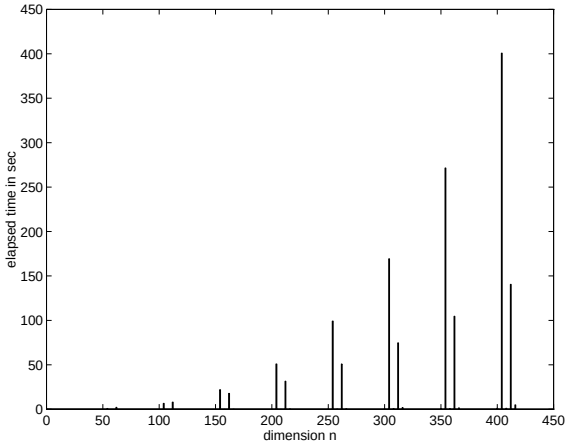
n	lu	$gaussel$	$A\backslash b$	$rref$	inv
20	0	0.05	0	0.39	0
40	0	0.49	0	1.43	0
60	0	1.54	0	3.02	0
80	0	3.51	0	5.28	0
100	0	6.70	0	8.02	0.05
120	0	11.53	0	11.48	0.11
140	0.06	17.96	0	15.43	0.22
160	0.06	26.69	0.11	20.32	0.22
180	0.11	37.79	0.11	25.76	0.33
200	0.11	56.35	0.11	34.49	0.44
250	0.21	99.26	0.32	50.92	0.49
300	0.38	169.34	0.66	74.70	1.26
350	0.71	271.50	0.66	104.63	1.49
400	1.10	400.79	1.04	140.56	3.89
Vergl.	$\frac{1}{3}T$	$100T$	$\frac{1}{3}T$	$35T$	T

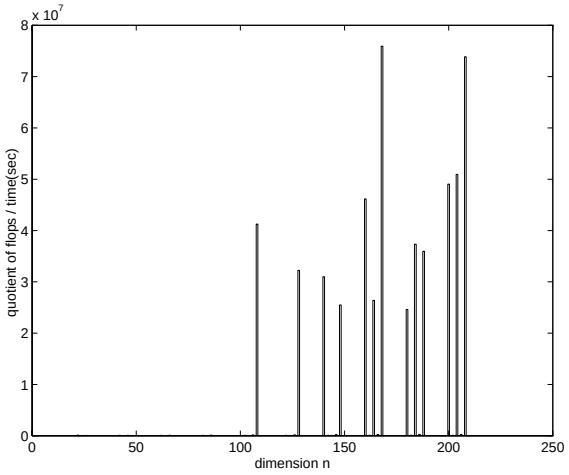
Einige graphische Auswertungen zur Komplexität bei $Ax = b$.
$flops$, $elapsed\ time\ in\ sec$ und $quotient\ of\ flops/time(sec)$ in Abhängigkeit von n.
Balken v.l.n.r.: lu, $gaussel$, $A\backslash b$, $rref$, inv.

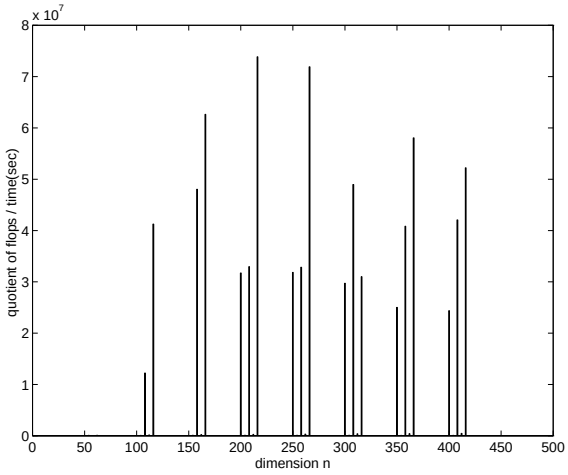












3.4.2 Komplexität von Algorithmen zur Bestimmung der Inversen

Wir vergleichen hier die Operationszahlen und Rechenzeiten von 4 Algorithmen. Die Rechnungen wurden auf einem PC mit Pentium II Prozessor 350MHz durchgeführt.

A^{-1}, AE um Einheitsmatrix I erweiterte Koeffizientenmatrix

$inv(A)$
$A\backslash I$
$[L,U] = lu(A);\ U\backslash(L\backslash I)$
$rref(AE)$

```
title('SPEEDTEST for INVERSION of MATRICES n = 10x10 .. 400x400 ')
clear n t s sn ns tn nt q tt ss
echo off
h = 1;     % Skalierung des Balkenabstandes
na = 10;
nsw = 10;
ne = 100;
for n=na:ns:ne
  A = rand(n)+n*eye(n);
  I = eye(n);
  AE = [A I];
  %
  t0 = clock;
  flops(0);
  inv(A);
  t(n) = etime(clock,t0);
  s(n) = flops;
  q(n) = 0;
  if (t(n)~=0), q(n) = s(n)./t(n); end;
  %
  t0 = clock;
  flops(0);
  A\I;
  t(n+2*h) = etime(clock,t0);
  s(n+2*h) = flops;
  q(n+2*h) = 0;
  if (t(n+2*h)~=0), q(n+2*h) = s(n+2*h)./t(n+2*h); end;
  %
  t0 = clock;
  flops(0);
  [L U] = lu(A);
  U\(L\I);
  t(n+4*h) = etime(clock,t0);
  s(n+4*h) = flops;
```

```
  q(n+4*h) = 0;
  if (t(n+4*h)~=0), q(n+4*h) = s(n+4*h)./t(n+4*h); end;
  %
  t0 = clock;
  flops(0);
  rref(AE);
  t(n+6*h) = etime(clock,t0);
  s(n+6*h) = flops;
  q(n+6*h) = 0;
  if (t(n+6*h)~=0), q(n+6*h) = s(n+6*h)./t(n+6*h); end;
  %
 end
[ns,sn] = bar(s);
plot(ns,sn,'g')
xlabel('dimension n')
ylabel('flops')
print graph34.ps -dps
pause
[nt,tn] = bar(t);
plot(nt,tn)
xlabel('dimension n')
ylabel('elapsed time in sec')
print graph35.ps -dps
pause
bar(q)
xlabel('dimension n')
ylabel('quotient of flops / time(sec) ')
print graph36.ps -dps
pause
clf

disp(' ')
disp('flops')
disp('        n        inv         A\I          lu        rref')
ss = zeros(round((ne-na)/nsw)+1,4);
for j=na:nsw:ne
  ss(round((j-na)/nsw)+1,1) = j;
  for k=0:2:6
    ss(round((j-na)/nsw)+1),round(k/2)+2) = s(j+k*h);
  end
end;
disp(ss)

disp('times')
disp('       n       inv       A\I        lu      rref')
```

```
tt = zeros(round((ne-na)/nsw)+1,5);
for j=na:nsw:ne
  tt(round((j-na)/nsw)+1,1) = j;
  for k=0:2:6
    tt(round((j-na)/nsw)+1,round(k/2)+2) = t(j+k*h);
  end
end;
disp(tt)
```

Ergebnistableau für PC Pentium II: *flops*.

n	inv	$A\backslash I$	lu	$rref$
20	17708	22840	22750	47497
40	134814	176676	176280	277181
60	447356	589548	588610	826784
80	1051446	1389568	1387740	1840280
100	2042818	2704470	2701670	3457743
150	6846281	9085008	9078745	11084328
200	16171200	21484502	21473320	25605056
250	31517269	41902646	41885395	49264454
300	54384734	72339686	72314970	84330436
350	86273097	114795124	114762045	133003654
400	128683576	171270178	171226620	197482589
$\mathcal{K}$	$2n^3$	$\frac{8}{3}n^3$		

Ergebnistableau für PC Pentium II: *times in sec*.

n	inv	$A\backslash I$	lu	$rref$
20	0	0	0	0.55
40	0	0	0	2.04
60	0	0	0	4.72
80	0.06	0	0.05	8.63
100	0.06	0.06	0.11	13.89
150	0.22	0.22	0.27	31.53
200	0.44	0.61	0.87	59.65
250	0.68	1.21	1.48	100.02
300	1.26	2.08	3.40	154.40
350	1.49	2.25	4.01	225.14
400	3.89	6.32	9.39	312.91
Vergl.	T	$T..\frac{3}{2}T$	$T..2T$	

Wir hatten schon die Komplexität für die Bestimmung der Lösung eines LGS mittels $inv(A) * b$ untersucht. Die alleinige Invertierung der Matrix unterscheidet sich in den $flops$ nicht entscheidend. So beträgt die Größenordnung der Komplexität für die inverse Matrix auch $\mathcal{K} = \mathcal{O}(2n^3)$. Auch die Rechenzeiten T sind ungefähr die gleichen.
Die Kommandos $A\backslash I$ und lu zeigen sowohl für $flops$ als auch in der Rechenzeit ähnliches Verhalten, sind aber nicht so gut wie inv. Ihre Komplexität ist $\mathcal{O}(\frac{8}{3}n^3)$.
$rref$ fällt im Vergleich als sichtbar schlechteste Variante auf.

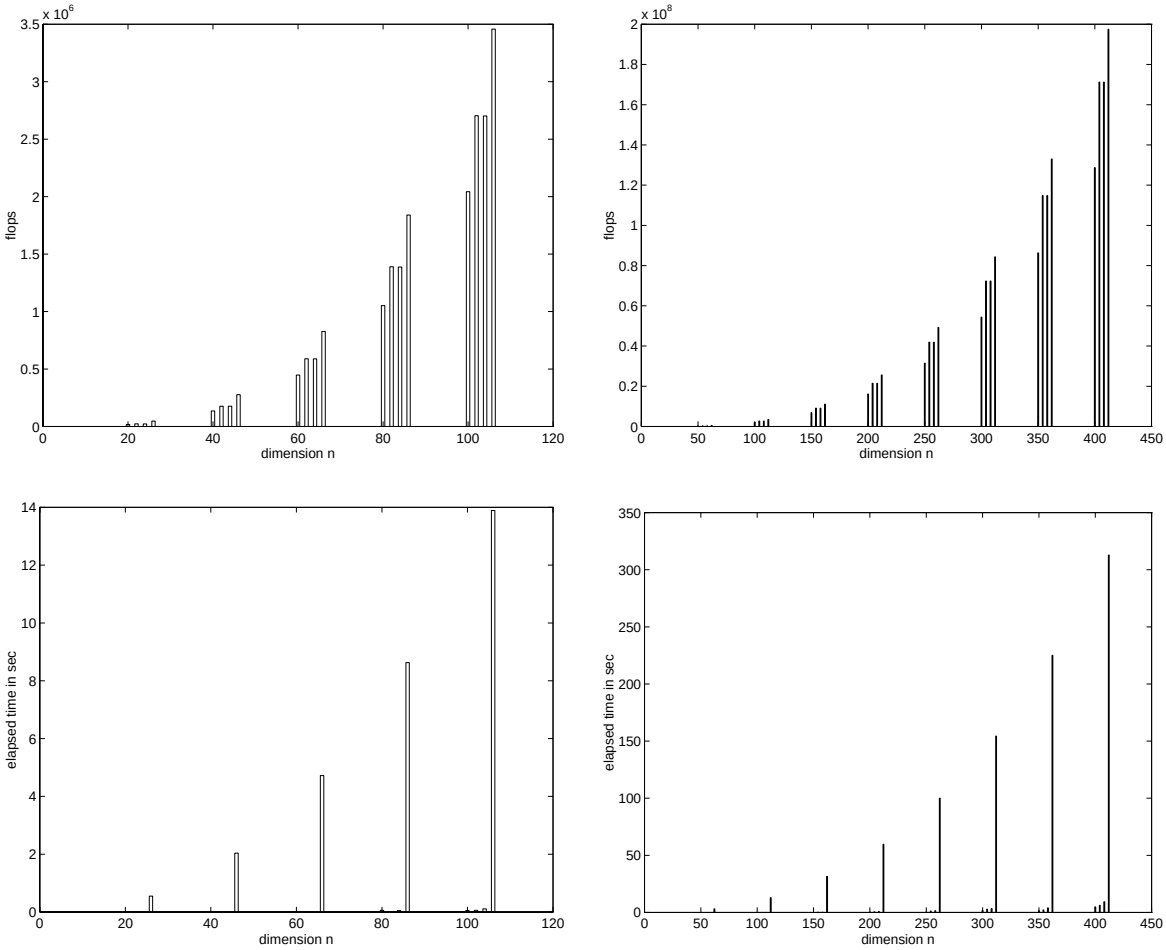
Die Kommandos $A\backslash I$ und lu geraten aber bezüglich der Rechenzeit im Vergleich mit inv mit wachsendem n immer mehr ins Hintertreffen. Das Verhältnis anders als bei den $flops$ verschlechtert sich jedoch. $rref$ braucht die längste Rechenzeit.

$A\backslash I$ braucht ca 4 Mal mehr $flops$ und $times$ als $A\backslash b$. Ähnlich bei lu, wobei die Rechenzeitrelation von $lu(A^{-1}) : lu(Ax = b)$ noch zunehmend schlechter wird.
Wird die inverse Matrix explizit gebraucht, so ist die built-in-Funktion inv für die Berechnung am besten geeignet.

Einige graphische Auswertungen zur Komplexität bei A^{-1}.

$flops$ und $elapsed\ time\ in\ sec$ in Abhängigkeit von n.

Balken v.l.n.r.: inv, $A\backslash I$, lu, $rref$.



3.5 Pivotstrategien für LGS

Bei der numerischen Lösung von LGS mittels Eliminationsverfahrens spielen sowohl die ***Skalierung*** als auch die ***Pivotstrategie*** eine wichtige Rolle. Beide zusammen bilden den wesentlichen Inhalt der Stabilität des Verfahrens.
Skalierung heißt eine Koeffizientenmatrix mit Elementen von ausgeglichener Größenordnung (Elementeabgleich, Äquilibrierung, Normalisierung) bereitzustellen. Sie ist eine Form der Verbesserung der Kondition der Matrix.
Pivotstrategien machen nur Sinn für solche Matrizen. Ohne die Forderung nach Gleichgewicht könnte man jede Zeile durch Multiplikation mit einem hinreichend großen Faktor zur Pivotzeile machen, falls der Kandidat für das Pivotelement nur verschieden von Null ist.
Die Kombination beider führt zu großer numerischer Stabilität bei relativ geringem zusätzlichen Aufwand. Dabei beschränkt man sich meist auf *Spaltenpivotisierung mit Zeilenvertauschung*, wählt aber das Pivotelement relativ zu einer Norm der entsprechenden Zeile der Ausgangsmatrix oder des Teiltableaus, ohne jedoch die Multiplikation mit den Skalierungsfaktoren tatsächlich durchzuführen (implizite Skalierung).

Zunächst wollen wir in Ergänzung zum m-File `gaussel.m` noch die beiden MATLAB Funktionen vor für die Varianten ohne und mit verbesserter Spaltenpivotisierung angeben.

```
% gausseo.m
function [B,k] = gausseo(A)
% gausseo returns the transformed rectangular matrix
% as upper triangular matrix B (without row exchanges,
% without pivot strategy), so far if it is possible.
% k is the number of performed stages. If k=min(n,m),
% then the minor A(1:k,1:k) is regular.
% Use for solving Ax=b or transformation of A.
%
[n m] = size(A); % row and column number of A
B = A;
k = 0;
r = 1;  % row index
c = 1;  % column index
while (c<=m) & (r<=n)
  if (B(r,r)==0)
    c=m+1; r=n+1;
  else
    % transform rest table
    k = k+1;
    for i = r+1:n
      if B(i,c) ~= 0
        t = B(i,c)/B(r,c);
        for j = c+1:m, B(i,j) = B(i,j)-t*B(r,j); end;
        B(i,c) = 0;
```

```
      end;
    end;
    r = r+1;
    c = c+1;
  end;
end;
% end of function gausseo
-------------------------------------------------------

% gausses.m
function [B,r] = gausses(A)
% gausses returns the transformed rectangular matrix
% as upper triangular matrix B (after possible row exchanges,
% pivot strategy with A(r,r)<>0 with largest value
% in the column c) with the rank r<=min(n,m).
% Use for solving Ax=b or transformation of A.
%
[n m] = size(A); % row and column number of A
B = A;
r = 1;  % row index
c = 1;  % column index
while (c<=m) & (r<=n)
  k = r;
  h = abs(B(r,c));
  i = r+1;
  % find largest pivot element in column c
  while  (i<=n)
    if (abs(B(i,c))>h)
      h = abs(B(i,c)); k = i;
    end;
    i = i+1;
  end;
  if (h>0)    % pivot element exists
    if (k~=r) % row exchange
      for j = c:m
        t = B(k,j); B(k,j) = B(r,j); B(r,j) = t;
      end;
    end;
    % transform rest table
    for i = r+1:n
      if B(i,c) ~= 0
        t = B(i,c)/B(r,c);
        for j = c+1:m, B(i,j) = B(i,j)-t*B(r,j); end;
        B(i,c) = 0;
      end;
```

```
    end;
    r = r+1;
  end;
  c = c+1;
end;
r = r-1;
% end of function gausses
```

Transformation von Rechteckmatrizen mittels verbesserter Pivotstrategie unter Verwendung von gausses. Hier ist die Zeilenvertauschung deutlich zu erkennen.

```
A = [ 1     1     0     1
      2     1    -1     1
     -1     2     3    -1
      3    -1    -1     2 ];
a = [ 2     1     4    -3 ]';

gausses([ A; a'; 2*a' ])   % (6,4)-Matrix
% im 1.Schritt wird 6.Zeile 2*a' "hochgetauscht"

ans =
    4.0000    2.0000    8.0000   -6.0000
         0    2.5000    5.0000   -2.5000
         0         0   -5.0000    4.0000
         0         0         0    2.4000
         0         0         0         0
         0         0         0         0

gausses([ A  a  2*a ])   % (4,6)-Matrix
% im 1.Schritt wird 4.Zeile  "hochgetauscht"

ans =
    3.0000   -1.0000   -1.0000    2.0000   -3.0000   -6.0000
         0    1.6667    2.6667   -0.3333    3.0000    6.0000
         0         0   -3.0000    0.0000    0.0000    0.0000
         0         0         0    0.6000    0.6000    1.2000
```

Implizite Skalierung = relative Pivotwahl mit Zeilenbetragssummen
(relative Kolonnenmaximumstrategie)

Die Matrixzerlegung $PA = LU$ mit impliziter Skalierung notieren wir als Turbo Pascal Routine.
Diese Methode findet man in den meisten Routinen zur LU-Zerlegung der großen Softwarepakete. Für die Ausgangsmatrix und die Zahlenwerte der aufeinanderfolgenden Schemata kann ein und dasselbe Feld benutzt werden. Nach beendeter Zerlegung werden somit $a_{ij} = l_{ij},\ i > j,\ l_{ii} = 1,$ und $a_{ij} = u_{ij},\ i \leq j,$ bedeuten.

Die Information über vorgenommene Zeilenvertauschungen wird im Vektor ip aufgebaut. Daraus kann die Permutationsmatrix P abgeleitet werden. Die Toleranzen dienen zu Testen auf Singularität der Matrix.

```
procedure Zerlegung_Gauss_rel_Pivot2(n:integer; var a:matrix;
  eps,detmax:float; var det:float; var ip:vektor1;
  var t,stufe:integer);
 { Gauss-Zerlegung von  A'=P*A(n,n)=L*U  auf dem Platz
   bei relativer Spaltenpivotisierung und Zeilenvertauschung,
   sowie impliziter Skalierung
   ( -> Permutationsmatrix P auf der Basis des Permutationsvektors ip)

   det(A) Determinante
   eps    Toleranz fuer Test auf Singularitaet |a[i,i]|<..,sum |a[i,j]|<..

   detmax Toleranz fuer Test auf |det(A)|>...
   ip     Permutationsvektor der Zeilenvertauschung
   t      Indikator : 0..default
                      1..Abbruch mit eps
                      2..Abbruch mit detmax
   stufe  Stufe (m,m) bei vorzeitigem Abbruch
   Lit.: H.R.Schwarz. Numerische Mathematik. B.G.Teubner Stuttgart 1988.}

var i,j,l,k:integer;
    q,s,max:float;
begin
 det:=1;
 stufe:=0; t:=0;
 for i:=1 to n do ip[i]:=i;        { Permutationsvektor --> P }
 for k:=1 to n do                  { Schritte k=1,2,...,n }
  begin
   { relative Pivotwahl mit Kolonnenmaximumstrategie }
   max:=0;
   for i:=k to n do
     begin
       s:=0;
       for j:=k to n do s:=s+abs(a[i,j]);
       if s<eps then    { Matrix A bzw. Teiltableau hat "Fast-Null-Zeile" }
         begin stufe:=i; det:=0; t:=1; EXIT; end;
       q:=abs(a[i,k])/s;
       if q>max then begin max:=q; l:=i; end;
     end;
   { Test auf Singularitaet (Nullspalte) und grosses det(A) }
   if max<eps then begin stufe:=k; det:=0; t:=1; EXIT; end;
   max:=a[l,k];
   det:=det*max;
   if abs(det)>detmax then begin stufe:=k; t:=2; EXIT; end;

   { Zeilenvertauschung }
   if l>k then
     begin
       det:=-det;
       for j:=1 to n do
         begin s:=a[k,j]; a[k,j]:=a[l,j]; a[l,j]:=s; end;
         j:=ip[k]; ip[k]:=ip[l]; ip[l]:=j;
       end;

   { Bestimmung der Elemente des Resttableaus }
   for i:=k+1 to n do   { Zeilen }
     begin
       s:=a[i,k]/max;
       a[i,k]:=s;
       for j:=k+1 to n do a[i,j]:=a[i,j]-s*a[k,j];  { Spalten }
     end;
  end;
end;
```

Der Algorithmus ist eigentlich nach $n-1$ Schritten schon abgeschlossen. Der letzte Schritt ($k = n$) dient nur zur Kontrolle des Koeffizienten $a_{nn} = u_{nn}$ und der Determinantenberechnung.
Ein weiterer Programmiertrick ist, daß man die relative Pivotwahl generell mit einem Vektor von Zeilennormen durchführt, der einmal vorab ermittelt wird. Eine solche Variante findet man in [8]. Dort werden die Zeilennormen $s_i = \sqrt{\sum_{j=1}^{n} a_{ij}^2}$ berechnet und ein sogenannter *Buchhaltervektor* *ph* mit den inversen Normen s_i^{-1} vorbesetzt. Bei Zeilentausch werden auch seine Komponenten entsprechend vertauscht. Der Aufwand verringert sich damit, ohne aber die Güte des Verfahrens entscheidend zu mindern.

```
procedure Zerlegung_Gauss_rel_Pivot1(n:integer; var a:matrix;
  eps,detmax:float; var det:float; var ip:vektor1;
  var t,stufe:integer);
 { Gauss-Zerlegung von  A'=P*A(n,n)=L*U  auf dem Platz
   bei relativer Spaltenpivotisierung und Zeilenvertauschung
   sowie impliziter Skalierung, Buchhaltervektor
   ( -> Permutationsmatrix P auf der Basis des Permutationsvektors ip)

   det(A) Determinante

   eps    Toleranz fuer Test auf Singularitaet |a[i,i]|<..,sum a[i,j]^2<..

   detmax Toleranz fuer Test auf |det(A)|>...
   ip     Permutationsvektor der Zeilenvertauschung
   t      Indikator : 0..default
                      1..Abbruch mit eps
                      2..Abbruch mit detmax
   stufe  Stufe (m,m) bei vorzeitigem Abbruch
   Lit.: N.Koeckler. Numerische Algorithmen in Softwaresystemen.
                     B.G.Teubner Stuttgart 1990.                  }

var i,j,l,k:integer;
    s,max,max1:float;
    ph:vektor;
begin
 det:=1;
 stufe:=0;
 t:=0;
 for i:=1 to n do
  begin
   ip[i]:=i;        { Permutationsvektor --> P }
   s:=0;
   for j:=1 to n do s:=s+sqr(a[i,j]);
   if s<eps then    { Matrix A hat "Fast-Null-Zeile" }
     begin stufe:=i; det:=0; t:=1; EXIT end;
   ph[i]:=1/sqrt(s);{ Vektor der Kehrwerte der Euklidischen Norm
                      der n Zeilenvektoren von A,
                      Buchhaltervektor                          }
  end;

 for k:=1 to n do   { Schritte k=1,2,...,n }
  begin
   { relative Pivotwahl mit Buchhaltervektor }
   max:=0;
   max1:=0;
   l:=k;

   for i:=k to n do
    begin
     s:=a[i,k];
     if abs(s*ph[i])>abs(max1) then
```

```
      begin max:=s; max1:=s*ph[i]; l:=i; end
    end;

   { Test auf Singularitaet und grosses det(A) }
   det:=det*max;
   if abs(max)<eps then begin stufe:=k; det:=0; t:=1; EXIT; end;
   if abs(det)>detmax then begin stufe:=k; t:=2; EXIT; end;

   { Zeilenvertauschung }
   if l>k then
    begin
     det:=-det;
     for j:=1 to n do
      begin s:=a[k,j]; a[k,j]:=a[l,j]; a[l,j]:=s; end;
     ph[l]:=ph[k];
     j:=ip[k]; ip[k]:=ip[l]; ip[l]:=j;
    end;

   { Bestimmung der Elemente des Resttableaus }
   for i:=k+1 to n do   { Zeilen }
    begin
     s:=a[i,k]/max;
     a[i,k]:=s;
     for j:=k+1 to n do a[i,j]:=a[i,j]-s*a[k,j];  { Spalten }
    end;
  end;
end;
```

3.6 Iterative Lösung von LGS

Iterative Verfahren für

$$Ax = b, \quad A(n,n), \quad a_{ii} \neq 0,\ i = 1,...,n, \quad x \text{ exakte Lösung},$$

beruhen auf Zerlegungen (Splitting) von A und Umformungen der Gestalt

$$\begin{aligned} A &= N - P, \quad \det N \neq 0 \\ Nx - Px &= b \\ Nx &= Px + b \\ x &= N^{-1}Px + N^{-1}b. \end{aligned}$$

Die allgemeine Iterationsverfahren (AIV, iterierfähige Form, Fixpunktform) ist

$$x^{(m+1)} = Hx^{(m)} + c, \quad m = 0,1,...; \quad H = N^{-1}P \text{ Iterationsmatrix}, \quad c = N^{-1}b.$$

Das AIV konvergiert für jede beliebige Startnäherung (Startvektor) $x^{(0)}$ in der Norm $\|\cdot\|$, falls mit der induzierten Matrixnorm $\|H\| < 1$ gilt. Dann ist $\lim\limits_{m\to\infty} x^{(m)} = x$.

$\|H\| < 1$ ist also eine hinreichende Bedingung.

Mit dem Spektralradius hat man eine hinreichende und notwendige Konvergenzbedingung: $\rho(H) < 1$.

3.6.1 Basisverfahren

Grundlage ist eine spezielle Zerlegung von A.

$$A = D - E - F = D(I - L - U)$$

I = Einheitsmatrix
$D = diag(A) = diag(a_{11}, a_{22}, ..., a_{nn})$
$L = (l_{ij})$ linke untere Dreiecksmatrix (Diagonale=0), $E = DL$
$U = (u_{ij})$ rechte obere Dreiecksmatrix (Diagonale=0), $F = DU$

Problemparameter

H	Iterationsmatrix
W	Vorkonditionierungsmatrix
$x^{(0)}$	Startvektor
$x^{(m+1)}$	Lösung oder letzter Vektor
$r^{(m+1)}$	$= b - Ax^{(m+1)}$, Restfehler der $(m+1)$-ten Iterierten, Residuum
$iter_{max}$	maximale Iterationsanzahl
ε	Test auf Abbruch der Iteration $\parallel x^{(m+1)} - x^{(m)} \parallel < \varepsilon,\ \parallel Ax^{(m+1)} - b \parallel < \varepsilon$
m	benötigte Iterationsanzahl
x_{abw}	$= \parallel x^{(m)} - x^{(m-1)} \parallel$

(1) Gesamtschrittverfahren (GSV, JACOBI)

Voraussetzung: $a_{ii} \neq 0, \quad i = 1(1)n$ (ggf. Zeilenvertauschungen vornehmen)

Zerlegung der Koeffizientenmatrix $A = N - P = D - (E + F)$

$$N = D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}, \quad P = E + F = \begin{pmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & 0 & \cdots & -a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ -a_{n1} & -a_{n2} & \cdots & 0 \end{pmatrix}$$

Komponentenweise Darstellung des GSV

$$x_i^{(m+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1,\ j \neq i}^{n} a_{ij} x_j^{(m)} \right), \quad i = 1(1)n, \quad m = 0, 1, 2, ...$$

Matrix-Vektor-Form des GSV

$$\begin{aligned} x^{(m+1)} &= D^{-1}(b + (E + F)x^{(m)}) \\ &= D^{-1}(D - A)x^{(m)} + D^{-1}b \\ &= (I - D^{-1}A)x^{(m)} + D^{-1}b \\ &= x^{(m)} - D^{-1}(Ax^{(m)} - b), \quad r^{(m)} = b - Ax^{(m)} \quad \text{Residuum} \\ &= x^{(m)} + D^{-1}r^{(m)} \end{aligned}$$

$$\begin{aligned} H = J &= I - D^{-1}A \quad \text{Iterationsmatrix} \\ W &= D \quad\quad\quad\; \text{Wichtung, Skalierungsmatrix} \\ & D^{-1}r^{(m)} \quad \text{gewichtetes Residuum,} \\ & \quad\quad\quad\quad\; \text{Verbesserung (update) für } x^{(m)} \end{aligned}$$

(2) Einzelschrittverfahren (ESV, GAUSS-SEIDEL)

Voraussetzung: $a_{ii} \neq 0, \quad i = 1(1)n$ (ggf. Zeilenvertauschungen vornehmen)

Zerlegung der Koeffizientenmatrix $A = N - P = (D - E) - F$

$$N = D - E = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}, \quad P = F = \begin{pmatrix} 0 & -a_{12} & -a_{13} & \cdots & -a_{1n} \\ 0 & 0 & -a_{23} & \cdots & -a_{2n} \\ 0 & 0 & 0 & \cdots & -a_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

Komponentenweise Darstellung des ESV

$$x_i^{(m+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(m)} \right), \quad i = 1(1)n, \quad m = 0, 1, 2, ...$$

Matrix-Vektor-Form des ESV

$$\begin{aligned} x^{(m+1)} &= D^{-1}(b + Ex^{(m+1)} + Fx^{(m)}) \\ Dx^{(m+1)} - Ex^{(m+1)} &= Fx^{(m)} + b \\ (D - E)x^{(m+1)} &= Fx^{(m)} + b \\ x^{(m+1)} &= (D - E)^{-1}[(D - E - A)x^{(m)} + b] \\ x^{(m+1)} &= (I - (D - E)^{-1}A)x^{(m)} + (D - E)^{-1}b \\ x^{(m+1)} &= \{I - [D(I - L)]^{-1}A\}x^{(m)} + [D(I - L)]^{-1}b \\ x^{(m+1)} &= x^{(m)} - [D(I - L)]^{-1}(Ax^{(m)} - b) \\ x^{(m+1)} &= x^{(m)} + (D - E)^{-1}r^{(m)}, \quad r^{(m)} = b - Ax^{(m)} \quad \text{Residuum} \\ H = H_1 &= I - [D(I - L)]^{-1}A = I - (D - E)^{-1}A \quad \text{Iterationsm.} \\ W &= D(I - L) = D - E \quad \text{Wichtung, Skalierungsmatrix} \\ & (D - E)^{-1}r^{(m)} \quad \text{gewichtetes Residuum,} \\ & \quad\quad\quad\quad\quad\; \text{Verbesserung (update) für } x^{(m)} \end{aligned}$$

(3) Fehleranalyse und Abschätzungen

Iterationsfehler $e^{(m)} = x^{(m)} - x$

Wegen

$$e^{(m+1)} = x^{(m+1)} - x = Hx^{(m)} + c - (Hx + c) = H(x^{(m)} - x) = He^{(m)}$$

sind der zentrale Punkt für Konvergenz- und Fehlerbetrachtungen die Norm bzw. Eigenwerte der Iterationsmatrix H.

Aufgrund der Beziehung

$$x^{(m+1)} - x^{(m)} = H(x^{(m)} - x^{(m-1)})$$

notieren wir die allgemeine Kontraktionsbedingung bei Iterationsverfahren

$$\|x^{(m+1)} - x^{(m)}\| \leq \lambda \, \|x^{(m)} - x^{(m-1)}\| \quad \text{mit} \quad 0 < \lambda < 1.$$

Des weiteren gelten

a-posteriori–Fehlerschätzung

$$\|x^{(m)} - x\| \leq \frac{\lambda}{1-\lambda} \, \|x^{(m)} - x^{(m-1)}\|,$$

a-priori–Fehlerschätzung

$$\|x^{(m)} - x\| \leq \frac{\lambda^m}{1-\lambda} \, \|x^{(1)} - x^{(0)}\|.$$

MATLAB-Funktionen für GSV und ESV.

```
% GSV.M
% Gesamtschrittverfahren, Jacobi-V. fuer LGS

function x = gsv(A,a,x0,TOL,MaxIt,NK)
% Eingangsparameter
%  Ax=a,    A=A(n,n)
%  x0       Startvektor
%  TOL      Toleranz
%  MaxIt    maximale Iterationsanzahl
%  NK       Normvariante (1,2,inf)
% Ergebnisse
%  x        Loesungsvektor oder letzter Vektor
%  i        Benoetigete Iterationen
%  r=A*x-a  Residuum und Norm  ||r||=norm(A*x-a,NK)
%  Diverse Informationen ueber Iterationsverlauf

D  = diag(diag(A));
D1 = diag(1./diag(A));
N  = D - A;
B  = D1*N;
b  = D1*a;
xn = x0;
```

```
for i=1:MaxIt
  xnp1 = B*xn + b;
  if (norm(xnp1-xn,NK)<TOL)
    disp(' ');
    disp(' Geforderte Genauigkeit erreicht!');
    disp(' Benoetigte Iterationen:');
    disp(i);
    break;
  end;
  xn = xnp1;
  if (i==MaxIt)
    disp(' ');
    disp('Maximale Iterationsanzahl erreicht!');
  end;
end;

x = xnp1;

disp(' ');
disp('Residuum r=A*x-a = '); disp(A*x-a);
disp('||r|| = '); disp(norm(A*x-a,NK));
% Ende Funktion gsv
```

```
% ESV.M
% Einzelschrittverfahren, Gauss-Seidel-V. fuer LGS

function x = esv(A,a,x0,TOL,MaxIt,NK)
% Eingangsparameter
%  Ax=a,    A=A(n,n)
%  x0       Startvektor
%  TOL      Toleranz
%  MaxIt    maximale Iterationsanzahl
%  NK       Normvariante (1,2,inf)
% Ergebnisse
%  x        Loesungsvektor oder letzter Vektor
%  i        Benoetigete Iterationen
%  r=A*x-a  Residuum und Norm  ||r||=norm(A*x-a,NK)
%  Diverse Informationen ueber Iterationsverlauf

xn   = x0;
xnp1 = x0;
n = max(size(xn));
```

```
for i=1:MaxIt
  for j=1:n
    s = 0;
    for k=1:j-1
      s = s - A(j,k)*xnp1(k);
    end;
    for k=j+1:n
      s = s - A(j,k)*xn(k);
    end;
    xnp1(j) = (s + a(j))/A(j,j);
  end;
  if (norm(xnp1-xn,NK)<TOL)
    disp(' Geforderte Genauigkeit erreicht!');
    disp(' Benoetigte Iterationen:');
    disp(i);
    break;
  end;
  xn = xnp1;
  if (i==MaxIt)
    disp(' ');
    disp('Maximale Iterationsanzahl erreicht!');
  end;
end;

x = xnp1;

disp(' ');
disp('Residuum r=A*x-a = '); disp(A*x-a);
disp('||r|| = '); disp(norm(A*x-a,NK));
% Ende Funktion esv
```

3.6.2 Das GSV mit streng diagonal dominanter Koeffizientenmatrix

```
  A = [ 12 -2  3
        -1  8 -2
        -1  3 12 ];
  b = [ 18 -32 6 ]';
  [n m] = size(AB);
  AB = [A b];
  xexakt = A\b

  xexakt =
       0.5450
      -3.5723
       1.4385
```

Einige Iterationen des GSV.

```
D = diag(diag(A));
c = inv(D)*b
H = eye(n)-inv(D)*A    % Iterationsmatrix

H =
          0    0.1667   -0.2500
     0.1250         0    0.2500
     0.0833   -0.2500         0

x0 = zeros(n,1);
xk = x0;
ndiff = 1;
k = 0;
while ndiff>1e-3
  x = xk
  xk = H*x+c
  ndiff = norm(xk-x,2)
  k = k+1
end
```

Ergebnistableau: x^* exakte Lösung.

k	$ndiff$	$x^{(k)}$		
0	1	0	0	0
1	4.3012	1.5	-4	0.5
2	1.4107	0.7083	-3.6875	1.6250
3	0.3264	0.4792	-3.5052	1.4809
4	0.1130	0.5456	-3.5699	1.4162
5	0.0237	0.5510	-3.5777	1.4379
6	0.0094	0.5442	-3.5716	1.4403
7	0.0021	0.5446	-3.5719	1.4383
8	0.0007	0.5451	-3.5724	1.4384
x^*		0.5450	-3.5723	1.4385

Fehlerabschätzungen für $\|x^* - x^{(8)}\|_\infty = 0.0002$

```
x1 = H*x0+c;
lambda = norm(H,inf)

lambda =
     0.4167
```

```
% a priori - Abschaetzung, etwas grob
w = lambda^k/(1-lambda)*norm(x1-x0,inf);
sprintf('||xk-x*|| <= %6.4f ',w)

ans =
  ||xk-x*|| <= 0.0062

% a posteriori-Abschaetzung, relativ genau
w = lambda/(1-lambda)*norm(xk-x,inf)
sprintf('||xk-x*|| <= %6.4f ',w)

ans =
  ||xk-x*|| <= 0.0003
```

3.6.3 GSV und ESV als Funktionen

Test am obigen Beispiel mit Genauigkeiten $TOL = 1e{-}4,\ 1e{-}6,\ 1e{-}8$ und Bestimmung der Anzahl der notwendigen Iterationen in der Routine.

```
MaxIt = 100;
TOL = 1e-4;
xg = gsv(A,a,x0,TOL,MaxIt,inf)

 Geforderte Genauigkeit erreicht!
 Benoetigte Iterationen:
      10
 Residuum r=A*x-a =
      1.0e-003 *
       -0.1416
        0.0843
        0.1055
 ||r|| =
      1.4160e-004
 xg =
        0.5450
       -3.5722
        1.4385

echo on
xe = esv(A,a,x0,TOL,MaxIt,inf)

 Geforderte Genauigkeit erreicht!
 Benoetigte Iterationen:
      6
```

```
Residuum r=A*x-a =
     1.0e-004 *
      -0.7487
       0.2184
       0
||r|| =
     7.4866e-005
xe =
       0.5450
      -3.5723
       1.4385
```

Ergebnistabelle mit Anzahl der Iterationen k und Residuumsfehler $\|r\|_\infty$ für verwendete Toleranzen. Das ESV braucht ungefähr halb soviel Iterationen wie das GSV.

Tol.	1e-4		1e-6		1e-8	
Verf.	k	$\|r\|_\infty$	k	$\|r\|_\infty$	k	$\|r\|_\infty$
GSV	10	1.4160e-004	14	8.2852e-007	17	2.2969e-008
ESV	6	7.4866e-005	8	6.3850e-007	10	5.4410e-009

3.6.4 Iterationsmatrizen des GSV und ESV

Test der Iterationsverfahren an einem Beispiel, wo Zeilenvertauschungen erforderlich sind.

```
A = [ 2   3  20  -1
      3   2   1  20
      1  10  -1   2
     10  -1   2  -3 ];
b = [ -10  15   5   0 ]';
xexakt = A\b

xexakt =
       0.3447
       0.2719
      -0.5403
       0.6981
```

Zeilentausch mit Permutationsvektor p oder -matrix P.

```
p = [ 4 3 1 2 ];
P = [ 0 0 0 1
      0 0 1 0
      1 0 0 0
      0 1 0 0 ];
```

```
% Zeilenvertauschung fuehrt auf diagonal dominante Matrix
AS = P*A;
n = size(AS);
bs = P*b;
```

Iterationsmatrizen und Normen.

```
% GSV
D = diag(diag(AS));
H = eye(n)-inv(D)*AS+eye(n)

H =
             0    0.1000   -0.2000    0.3000
       -0.1000         0    0.1000   -0.2000
       -0.1000   -0.1500         0    0.0500
       -0.1500   -0.1000   -0.0500         0

n1g = norm(H,inf);
sprintf('Zeilensummennorm = %6.4f ',n1g)

ans =
 Zeilensummennorm = 0.6000

n2g = norm(H,1);
sprintf('Spaltensummennorm = %6.4f ',n2g)

ans =
 Spaltensummennorm = 0.5500

% ESV
C = tril(AS);
H1 = eye(n)-inv(C)*AS

H1 =
             0    0.1000   -0.2000    0.3000
        0.0000   -0.0100    0.1200   -0.2300
        0.0000   -0.0085    0.0020    0.0545
        0.0000   -0.0136    0.0179   -0.0247

n1e = norm(H1,inf);
sprintf('Zeilensummennorm = %6.4f ',n1e)

ans =
 Zeilensummennorm = 0.6000
```

```
n2e = norm(H1,1);
sprintf('Spaltensummennorm = %6.4f ',n2e)

ans =
 Spaltensummennorm = 0.6092
```

Anwendung der a-priori-Abschätzung zur Bestimmung der Iterationsanzahl bei Toleranz $1e{-}4$ und Zeilensummennorm. Die Anzahl der wirklich gebrauchten Iterationen dafür ist meist geringer.

```
x0 = zeros(n,1);    % Startvektor
tol = 1e-4;
% GSV
x1 = H*x0+inv(D)*bs;
lambda = n1g;
normg = norm(x1-x0,inf)

normg =
     0.7500

kgsv = ceil(log(tol*(1-lambda)/normg)/log(lambda))

kgsv =
     20
```

4 Anhang

Anhang A

Ausgabe von MATLAB Plots

Einige Bemerkungen zum Druck von Plots in *m*-Files auf Drucker. Dazu dient das Kommando *print*.
Dabei ist neben dem Dateinamen die Geräteoption Encapsulated PostScript (EPS) oder PostScript (PS) anzugeben. Beide Varianten sind funktionieren jedoch anders, wenn der Dateiname fehlt.

- Druck von PS File

print filename.ps −dps	Ausgabe des Plots als File
print −dps	automatischer Druck des Files auf Drucker

  Dateikopf und -ende der entsprechenden Postscript-Datei (*ps*-Format) sind

```
%!PS-Adobe-2.0
%%Creator: MATLAB, The Mathworks, Inc.
%%Title: grdem31.ps
%%CreationDate: 07/12/99  15:27:00
%%DocumentNeededFonts: Helvetica
%%DocumentProcessColors: Cyan Magenta Yellow Black
%%Pages: (atend)
%%BoundingBox: (atend)
%%EndComments

%%BeginProlog
% MathWorks dictionary
/MathWorks 150 dict begin
 ..............................

gr
end
eplot
%%EndObject graph 1

epage
end
showpage

%%Trailer
%%BoundingBox:    75   209   545   589
%%Pages: 001
%%EOF
```

- Druck von EPS File

  print filename.eps −deps Ausgabe des Plots als File

  print −deps kein automatischer Druck des Files

  Dateikopf und -ende der entsprechenden Postscript-Datei (*eps*-Format) sind

```
%!PS-Adobe-2.0 EPSF-1.2
%%Creator: MATLAB, The Mathworks, Inc.
%%Title: grdem32.eps
%%CreationDate: 07/12/99  15:27:01
%%DocumentNeededFonts: Helvetica
%%DocumentProcessColors: Cyan Magenta Yellow Black
%%Pages: 1
%%BoundingBox:    75   209   548   589
%%EndComments

%%BeginProlog
% MathWorks dictionary
/MathWorks 150 dict begin
.........................

gr
end
eplot
%%EndObject graph 1
epage
end
showpage
%%Trailer
%%EOF
```

  Ohne Angabe des Dateinamens erscheint eine Fehlermeldung.

  Encapsulated PostScript files can not be sent to printer.
  File saved to disk under name 'figure1.eps'.

Darüber hinaus ist in der Datei ersichtlich, daß die Graphik sich in einer Box mit den Grenzen (75,209,548(oder 545),589), also der Dimension $473 \times 380pt = 167 \times 134mm$ befindet. Diese Graphiken können z.B. dann in einem TEX oder LATEX Dokument einbezogen und manipuliert werden. Wir notieren Beispielvarianten mit dem Stil *psfig.sty*.

```
% im Original
\psfig{figure=grdem31.ps}
% normale Ansicht auf halbe Breite verkleinert
\psfig{figure=grdem32.eps,width=8cm}
```

Anhang B

Zusammenstellung von Adressen

1. World Wide Web (WWW) mit MATLAB Seiten

Die TEX Quelle sowie das PostScript file `primer35.ps` der 2.Edition des MATLAB Primers steht stehen mittels *ftp* auf `ftp.math.ufl.edu` im Verzeichnis `pub/matlab` zur Verfügung.

Die MathWorks Inc. und MathTools Ltd. entwickeln und vertreiben die Computersoftware MATLAB und andere Komponenten und sind natürlich mit ihrem ganzen Angebot auch im Internet zu finden.

http://www.mathworks.com/	The MathWorks Inc. home (auch Simulink)
http://www.mathworks.com/products/matlab/	MATLAB 5.3
http://www.mathworks.com/products/matlab/	MATLAB web server 1.0
http://www.mathworks.com/support/books/	MATLAB based books
http://www.mathtools.com/	MATLAB Toolboxen von MathTools Ltd. (auch MATCOM, MIDEVA)
http://krum.rz.uni-mannheim.de/cafgbench.html	Computer Algebra Benchmarks (RZ/Uni Mannheim)

2. Verzeichnisse mit MATLAB *m*-Files für Skripte und Funktionen

Zu den Kapiteln 1-3 des Skripts liegen die entsprechenden Files (meist *m*-Files) und diverse Daten- und Ergebnisfiles vor.

`*.m, *.ps, *.eps, *.mat`

Die Dateien sind zu finden im Novell-Netz PIVOT des Instituts für Mathematik bzw. auf der persönlichen Homepage im Internet.

\\PIVOT\SHARE Q:\NEUNDORF\STUD_M93\MATLAB1

Homepage Navigator → Publications → Computeralgebra → MATLAB1

e-mail: ***neundorf@mathematik.tu-ilmenau.de***

Homepage: ***http://imath.mathematik.tu-ilmenau.de/~neundorf/index_de.html***

Literatur

[1] Sigmon, K.: *MATLAB Primer, Second Edition.*
Department of Mathematics, University of Florida USA 1992.

[2] Sigmon, K.: *MATLAB Primer 5e.* CRC Press 1998.

[3] *MATLAB User's Guide and Reference Guide.*

[4] *MATLAB Release Notes 4.1. For Unix Workstations.*
The MathWorks Inc. Natick, Massachusetts USA 1993.

[5] *The Student Edition of MATLAB. Version 4. User's Guide.*
The MathWorks Inc. Prentice Hall, Englewood Cliffs New Jersey 1995.

[6] *The Student Edition of MATLAB 5.* Prentice Hall.

[7] Redfern, D.; Campbell, C.: *The MATLAB 5 Handbook.* Springer-Verlag 1998.

[8] Köckler, N.: *Numerische Algorithmen in Softwaressystemen: unter besonderer Berücksichtigung der NAG-Bibliothek.* B.G. Teubner Stuttgart 1990.

[9] Neundorf, W.: *Manipulation von Matrizen I - Grundlagen, Norm, Kondition und Skalierung.* Preprint M 16/96 IfMath der TU Ilmenau, November 1996.

[10] Mathews, J.H.; Fink, K.D.: *Numerical Methods using MATLAB.* Prentice Hall London 1999.

[11] Chen, K.; Giblin, P.J.; Irving, A.: *Mathematical explorations with MATLAB.* Cambridge University Press 1999.

[12] Mohr, R.: *Numerische Methoden in der Technik: eine Lehrbuch mit MATLAB-Routinen.* Vieweg Braunschweig 1998.

[13] Golubitsky, M.; Dellnitz, M.: *Linear algebra and differential equations using MATLAB.* Brooks/Cole Pub. Co Pacific Grove 1999.

Anschrift:

Dr. Werner Neundorf
Technische Universität Ilmenau, Institut für Mathematik
PF 10 0565
D - 98684 Ilmenau

e-mail : neundorf@mathematik.tu-ilmenau.de