MATLAB

Eine Einführung

Getting Started with MATLAB

Übersetzt und gestaltet von
Robert Wilke und **Ulf Wittl**
im Rahmen des Meile2000-Projekts

Ergänzt durch Übungen von
Stephanie Schüpferling

Fachbereich Informatik und Mathematik
FH Regensburg
Prof. Dr. H.-J. Wagner und Prof. Dr. H.-W. Goelden

Version 2.0

MATLAB-Version 5.3

Hinweise

Alle Beispiele wurden mit MATLAB Version 5.3 getestet. Wenn auf Ihrem Computer MATLAB installiert ist, können Sie die im Text markierten Programme ablaufen lassen, ohne diese zuvor abzutippen. Es genügt, wenn Sie einfach auf ein markiertes Programm klicken. Dadurch öffnet sich ein MATLAB-Befehlsfenster, in dem die Befehlsfolge abläuft.

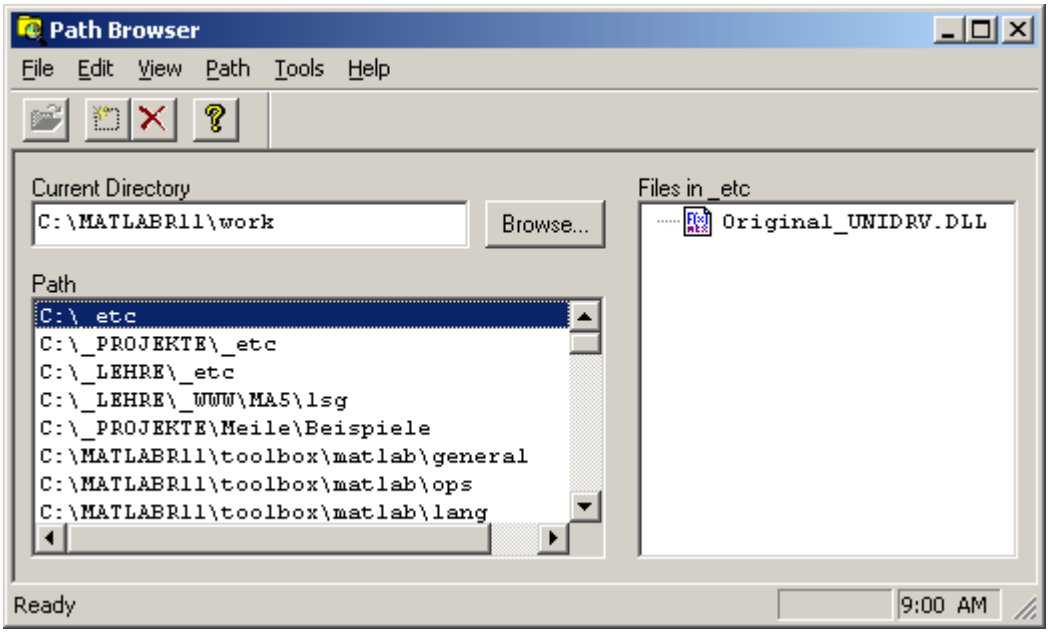
Durch Eingabe einer beliebigen Taste wird das MATLAB-Befehlsfenster wieder geschlossen. Wenn Sie es aber ausnahmsweise offen lassen wollen, um vielleicht noch etwas auszuprobieren, dann tippen Sie einfach [Strg][C] und es bleibt erhalten.

Voraussetzung hierfür sind allerdings 2 Installationsmaßnahmen, die im folgenden beschrieben werden.

Installation

Um die vorbereiteten Verknüpfungen zu MATLAB über Programm unter Windows funktionsfähig zu machen, sind folgende beiden Schritte erforderlich:

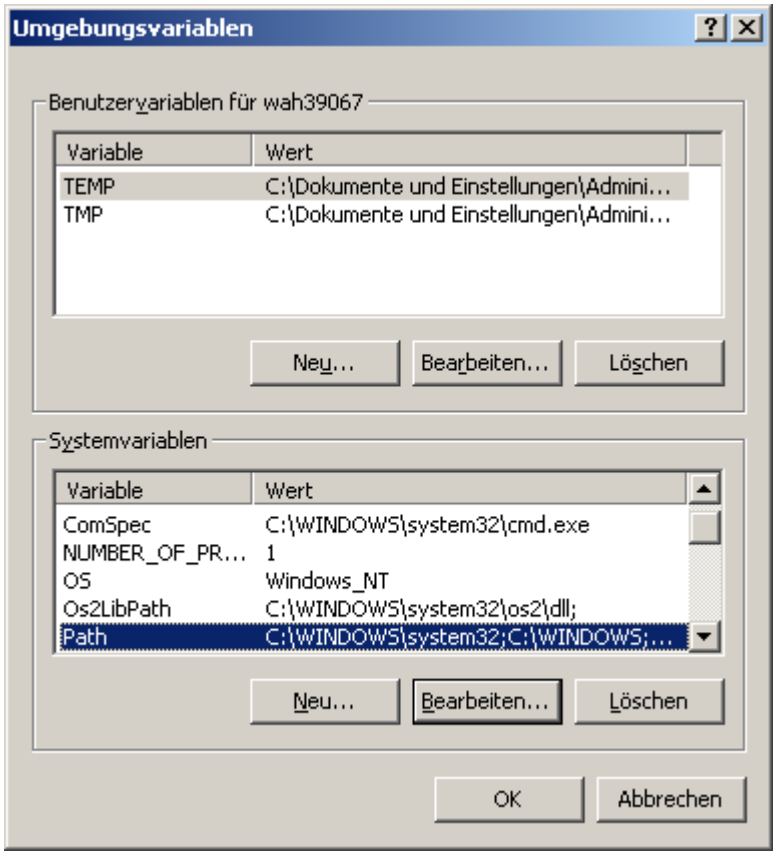
1. In MATLAB muss ein Pfadeintrag auf den Ordner verweisen, in dem die Programm-Dateien stehen. Dazu öffnet man in MATLAB den Path-Browser (s.u.) über **File / Set Path...**



   Hierin ist unter **Path / Add to path** der Pfad für die Programm-Dateien einzutragen. Anschließend ist diese Pfadeinstellung mit **File / Save path** zu speichern.

2. Der System-Pfad muss den Ordner enthalten, in dem die Datei **matlab.exe** steht, also etwa **C:\Programme\Matlab\bin**.

   In Windows2000 geschieht dies z.B. über **Start / Einstellungen / Systemsteuerung / System**. Im Fenster **Systemeigenschaften** können dann auf der Registerkarte **Erweitert** die **Umgebungsvariablen** eingestellt werden. Dazu erscheint folgendes Fenster, in dessen unterer Zeile der Pfadeintrag markiert wurde.



   Nach Klick auf Bearbeiten kann man ihn vollständig lesen. Sofern noch kein Verweis auf MATLAB vorliegt, wird der vorhandene Pfad-Eintrag ergänzt durch den Pfad zur Ihrer Datei **matlab.exe**, also z.B. durch **;C:\Programme\Matlab\bin**

Einige der Übungsaufgaben sind mit einem Pfeil (z.B. 3.►) versehen. Diese Aufgaben können auch mit Papier und Bleistift gelöst werden, z.B. als Prüfungsvorbereitung für Lineare Algebra.

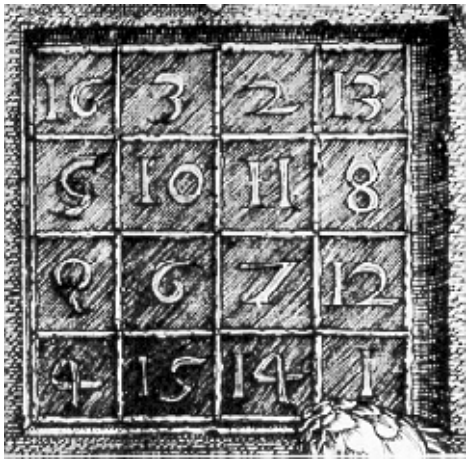
Bei Aufgaben mit einem Stern (z.B. 4.*) ist der Aufwand zur Lösung etwas höher.

Inhalt

1. Matrizen und Magische Quadrate

MATLAB ist am besten zu erlernen, wenn man lernt mit Matrizen umzugehen. Dies zeigt Ihnen folgender Abschnitt. In MATLAB sind Matrizen rechteckige Zahlen-Felder. 1x1-Matrizen haben oft eine besondere Bedeutung, sie heißen auch Skalare. Matrizen mit nur einer Zeile oder Spalte heißen Vektoren. MATLAB hat verschiedene Möglichkeiten, numerische und nicht-numerische Daten zu speichern. Am Anfang ist es aber am Besten sich alles in Matrixform vorzustellen. Die Rechenoperationen sind in MATLAB so natürlich wie möglich gestaltet. Wo andere Programmier-sprachen mit einzelnen Zahlen rechnen, ermöglicht MATLAB Ihnen, mit ganzen Matrizen effizient und einfach zu arbeiten.

Ein gutes Beispiel für eine Matrix erscheint im Renaissance Holzschnitt Melancholia I des deut-schen Künstlers Albrecht Dürer. Wir wollen diese im folgenden häufig benutzen.



Sie befindet sich in der rechten oberen Ecke des Bildes und stellt ein sogenanntes Magisches Quadrat dar, dessen Charakteristika zu Dürers Zeiten viele Leute beeindruckten. Wir wollen sie gleich näher anschauen.

Eingabe von Matrizen

Sie können in MATLAB Matrizen auf verschiedene Arten eingeben:

- Eingabe einer Reihe von Elementen
- Matrizen aus externen Dateien laden
- Erzeugen von Matrizen mit Hilfe vorgegebener (integrierter) Funktionen
- Erstellen von Matrizen mit eigenen Funktionen in M-Files

Geben Sie Dürers Matrix A als eine Liste von Elementen ein. Sie müssen nur wenige Grundregeln beachten:

- Die Elemente werden durch Leerzeichen oder Kommas getrennt
- Ein Strichpunkt markiert das Ende einer Zeile
- Die gesamte Matrix wird von eckigen Klammern [ ] eingeschlossen

Um die Matrix von Dürer einzugeben, tippen Sie einfach:

```
A = [ 16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB zeigt sofort die eingegebene Matrix an:

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Beispiel 1.1

Dies stimmt exakt mit den Zahlen auf dem Holzschnitt überein. Jede eingegebene Matrix wird von MATLAB im Arbeitsspeicher abgelegt. Sie können sie einfach mit ihrem Namen A aufrufen. Nun

können Sie einen Blick darauf werfen, was die gespeicherte Matrix A so interessant macht. Warum werden ihr magische Fähigkeiten zugeschrieben?

Summe, Transposition und Spur

Vielleicht haben Sie schon bemerkt, dass die speziellen Eigenschaften des Magischen Quadrats mit der Summierung der Elemente in verschiedenen Richtungen zusammenhängt. Bilden Sie die Summe jeder Zeile oder Spalte. Sie werden immer die selbe Zahl erhalten. Überprüfen Sie dies mit MATLAB. Der erste Befehl ist:

```
sum(A)
```

MATLAB gibt folgendes aus:

```
ans =
    34    34    34    34
```

Beispiel 1.2

Falls keine Ausgabe-Variable definiert wird, benutzt MATLAB die Variable *ans*, kurz für answer (Antwort) um das Ergebnis der Berechnung zu speichern. Sie haben mit dem ***sum***-Befehl einen Zeilenvektor gebildet, der die Spaltensummen von A enthält. Jede der Spalten hat die gleiche Summe – die magische Summe 34.

Wie steht es mit den Zeilensummen? MATLAB-Befehle wirken in der Regel spaltenweise. Die einfachste Art die Zeilensummen zu berechnen, ist die Matrix zu transponieren, anschließend die Spaltensummen der transponierten Matrix zu bilden, und dann das Ergebnis nochmals zu transponieren. Die Transposition wird durch ein Hochkomma (') angegeben, d.h. die Matrix wird an der Hauptdiagonalen gespiegelt. Transposition eines Zeilenvektors liefert einen Spaltenvektor und umgekehrt. So erzeugt

```
A'
```

die Ausgabe

```
ans =
    16     5     9     4
     3    10     6    15
     2    11     7    14
    13     8    12     1
```

Beispiel 1.3

und

```
sum(A')'
```

gibt die Zeilensummen von A in einem Spaltenvektor aus:

```
ans =
    34
    34
    34
    34
```

Beispiel 1.4

Die Summe der Elemente der Hauptdiagonalen, die Spur der Matrix A, wird einfach mit Hilfe der ***diag***-Funktion ermittelt

```
sum(diag(A))
```

ergibt:

```
ans =
     34
```

Beispiel 1.5

Die Gegendiagonale ist mathematisch nicht so wichtig, deshalb gibt es keine MATLAB-Funktion zu ihrer Summierung. Die Funktion ***fliplr***, die eigentlich für Graphik-Anwendungen gedacht war, hilft hier jedoch weiter. Sie spiegelt die Matrix an ihrer Mittelachse. Dadurch wird die Gegendiagonale zur Hauptdiagonalen und ihre Spur ist

```
sum(diag(fliplr(A)))

ans =

     34
```

Beispiel 1.6

Damit haben Sie geprüft, dass die Matrix im Holzschnitt von Dürer wirklich ein magisches Quadrat ist und zugleich einige Beispiele für die Matrizenrechnung kennengelernt. In den folgenden Abschnitten wird die Matrix A benutzt, um weitere MATLAB-Eigenschaften zu veranschaulichen.

Indizes

Das Element in Zeile i und der Spalte j der Matrix A wird mit A(i, j) bezeichnet. Zum Beispiel steht A(4, 2) für die Zahl in der 4. Zeile und 2. Spalte von A. In unserem magischen Quadrat ist A(4, 2) = 15.

Mit dieser Beziehungsweise kann man die Summe der Elemente in der 4. Spalte auch folgendermaßen berechnen:

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

Resultat ist

```
ans =
     34
```

Beispiel 1.7

Dies ist sicher nicht der eleganteste Weg, um die Summe einer einzelnen Spalte zu bilden.

Es ist außerdem möglich, Elemente einer Matrix mit einem einfachen Index anzusprechen, etwa A(k). Dies ist der übliche Weg Komponenten von Zeilen- und Spaltenvektoren zu bezeichnen. Aber einfache Indizes können auch bei Matrizen benutzt werden. In diesem Fall wird das Zahlenfeld A als ein langer Spaltenvektor betrachtet, der sich durch Aneinanderreihen der Spalten der Original-Matrix A ergibt. Für unser magisches Quadrat ist also A(8) eine andere Möglichkeit den Wert 15 in A(4,2) anzusprechen.

Falls Sie einen Wert außerhalb der Matrix aufrufen wollen, gibt MATLAB einen Fehler aus:

```
t = A(4,5)
???  Index exceeds matrix dimensions.
```

Beispiel 1.8

Man kann jedoch einen Wert außerhalb der Feldgrenzen speichern. Die Größe der Matrix wird für das neue Element passend erweitert:

```
X = A;
X(4,5) = 17
X =

    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4    15    14     1    17
```

Beispiel 1.9

Der Doppelpunkt-Operator

Der Doppelpunkt ist einer der wichtigsten Operatoren in MATLAB. Er tritt in verschiedenen Zusammenhängen auf. Der Ausdruck:

```
1:10
```

ist ein Zeilenvektor, der die Zahlen eins bis zehn der Reihe nach enthält:

```
ans =
     1     2     3     4     5     6     7     8     9    10
```

Beispiel 1.10

Um andere Abstände zu erhalten, geben Sie eine beliebige, evtl. auch negative Schrittweite an. Z.B ergibt

```
100:-7:50
ans =
   100    93    86    79    72    65    58    51
```

Beispiel 1.11

und

```
0:pi/4:pi

ans =
         0    0.7854    1.5708    2.3562    3.1416
```

Beispiel 1.12

Indexausdrücke, die den Doppelpunktoperator enthalten, beziehen sich auf Ausschnitte von Matrizen, sog. Teilmatrizen.

```
A(1:3,2)
ans =
         3
        10
         6
```

Beispiel 1.13

ist ein Spaltenvektor und besteht aus den ersten 3 Elementen der 2. Spalte von A.

```
sum(A(1:4,4))
ans =
        34
```

Beispiel 1.14

berechnet die Summe der 4. Spalte von A. Das geht sogar noch einfacher: Der alleinstehende Doppelpunkt bezieht sich auf alle Matrixelemente in einer Zeile oder Spalte. Das Schlüsselwort ***end*** bezieht sich auf die letzte Zeile oder Spalte. Man kann es z.B. verwenden, wenn man nicht mehr weiß, wie groß die Matrix ist.

```
sum(A(:,end))
```

berechnet die Summe der Elemente der letzen Spalte von A.

```
ans =
        34
```

Beispiel 1.15

Ebenso gut hätte man ***sum*(A(:,4))** schreiben können.

Warum ist die magische Summe bei einem 4x4 - Quadrat 34? Falls die ganzen Zahlen von 1 bis 16 in vier Gruppen mit gleichen Summen aufgeteilt sind, muss

```
sum(1:16)/4
```

natürlich

```
ans =
        34
```

sein.

Beispiel 1.16

Die *magic*–Funktion

MATLAB hat übrigens eine eingebaute Funktion, die magische Quadrate nahezu beliebiger Größe erzeugen kann. Diese Funktion hat den Namen ***magic***

```
B = magic(4)
B =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Beispiel 1.17

Diese Matrix ist fast identisch zur Matrix A aus Dürers Holzschnitt. Sie hat auch all deren magische Eigenschaften. Der einzige Unterschied besteht darin, dass die beiden mittleren Spalten vertauscht sind. Um die Matrix B in Dürers Matrix umzuwandeln, müssen nur diese beiden Spalten vertauscht werden. Das kann geschehen mit

```
A = B(:,[1, 3, 2, 4])
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Beispiel 1.18

Diese Anweisung hat bewirkt, dass in jeder Zeile von Matrix B die Elemente in der Reihenfolge 1, 3, 2, 4 angeordnet wurden.

Warum hat wohl Dürer A anstelle von B in seinem Holzschnitt verewigt? – Vermutlich wollte er die Jahreszahl 1514, das Entstehungsjahr seines Werkes, auf diese Weise in der letzten Matrix-Zeile unterbringen.

Übungsaufgaben

1. Erzeugen Sie ein magisches Quadrat der Größe 6x6 und weisen Sie alle Eigenschaften von magischen Quadraten nach.

   Lösung 1.1

2. Erzeugen Sie mit der Funktion ***rand*** eine 5x5-Zufallsmatrix $\mathbf{A}$. Welches sind die Werte der folgenden Ausdrücke?
   a) $\mathbf{A}(2,:)$
   b) $\mathbf{A}(:,5)$
   c) $\mathbf{A}(:,1)$
   d) $\mathbf{A}([1,5])$
   e) $\mathbf{A}(1,1:2:5)$
   f) $\mathbf{A}(4:-1:1,5:-1:1)$

Lösung 1.2

3. Erzeugen Sie eine Diagonalmatrix mit 1,2,3,4,5 auf der Diagonalen.

Lösung 1.3

4.* Erzeugen Sie eine tridiagonale 5x5-Matrix mit Einsen auf der Hauptdiagonalen, Zweien auf der unteren Nebendiagonalen und Dreien auf der oberen Nebendiagonalen. Verwenden Sie einmal den Befehl ***diag*** zum Erzeugen einer vollen Matrix und einmal den Befehl ***spdiags*** zum Erzeugen einer schwach besetzten Sparse-Matrix.

Lösung 1.4

2. Ausdrücke

MATLAB stellt, wie viele andere Programmiersprachen, mathematische Ausdrücke zur Verfügung. Zusätzlich können diese auch komplette Matrizen verarbeiten. Ausdrücke sind unterteilt in folgende Bestandteile

- Variablen
- Zahlen
- Operatoren
- Funktionen

Variablen

MATLAB benötigt keine speziellen Typdeklarationen oder Dimensionsangaben. Wenn in MATLAB eine neue Variable vorkommt, wird sie automatisch deklariert und ihr wird der passende Speicher zugewiesen. Existiert die Variable bereits, so wird sie überschrieben, gegebenenfalls neu deklariert und neuer Speicher zugewiesen. So erzeugt zum Beispiel

```
num_students = 25;
```

Beispiel 2.1

eine 1x1-Matrix mit dem Namen *num_students* und speichert den Wert 25 in ihrem reservierten Speicher.

Variablennamen bestehen aus mindestens einem Buchstaben am Anfang, gefolgt von beliebig vielen Buchstaben, Zahlen oder Unterstrichen. MATLAB verwendet zur Unterscheidung von Namen nur ihre ersten 32 Zeichen. MATLAB unterscheidet zwischen Groß- und Kleinschreibung. a und A sind somit verschiedene Variablen. Um sich den Inhalt einer Matrix anzeigen zu lassen tippt man einfach ihren Namen ein.

Zahlen

MATLAB verwendet eine normale Dezimalschreibweise mit oder ohne Dezimalpunkt und evtl. Minus oder Plus als Vorzeichen. Die wissenschaftliche Schreibweise verwendet den Buchstaben e um eine Zehnerpotenz darzustellen. (1.23e4 = 12300). Imaginäre Zahlen werden durch ein hintangestelltes i oder j dargestellt. Es folgen ein paar Beispiele für gültige Zahlendarstellungen:

3	-99	0.0001
9.6397238	1.60210e-20	6.02252e23
1i	-3.14159i	3e5i

Alle Zahlen werden intern im *double* - Format gespeichert. Dies ist das durch IEEE 754 spezifizierte Standard-Format für Gleitpunktarithmetik. Die Zahlen besitzen eine Genauigkeit von ungefähr 16 Dezimalstellen und haben Werte zwischen ca. 10^{-308} und 10^{308}. Außerdem sind 0 und $\pm\infty$ als Zahlenwerte vorhanden.

Operatoren

In Ausdrücken gelten die geläufigen arithmetischen Operatoren und Prioritätsregeln (Punkt- vor Strichrechnung usw.), die durch Klammern abgeändert werden können. Arithmetische Operatoren sind

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
\	Linke Division
^	Potenzierung
'	Transposition (mit Konjugation)
()	Klammern zur Festlegung der Auswertungsreihenfolge

Funktionen

MATLAB stellt eine große Anzahl von elementaren mathematischen Funktionen, wie ***abs***, ***sqrt***, ***exp*** und ***sin***, zur Verfügung. Das Ziehen einer Wurzel oder das Logarithmieren einer negativen Zahl ist kein Fehler. Stattdessen wird das Ergebnis automatisch als komplexe Zahl ausgegeben. Außerdem stellt MATLAB viele spezielle mathematische Funktionen zur Verfügung, wie z.B. ***bessel***-Funktionen und ***gamma***-Funktion. Die meisten dieser Funktionen akzeptieren komplexe Zahlen und Matrizen als Argumente. Um sich eine Liste aller elementaren mathematischen Funktionen ausgeben zu lassen, gibt man

```
help elfun
```

ein. Für eine Liste der speziellen Funktionen gibt man

```
help specfun
```

ein. Das Repertoire an elementaren Matrizen zeigt

```
help elmat
```

an.

Einige der Funktionen, wie z.B. ***sqrt*** und ***sin***, sind direkt in MATLAB integriert, deshalb sind sie sehr schnell und effektiv. Nachteilig ist aber, dass man dadurch nicht an deren Quellcode herankommt. Dies ist bei anderen Funktionen, wie z.B. bei ***gamma*** und ***sinh*** möglich, die als sogenannte M-Files vorliegen. Man kann deren Code auch selbst modifizieren, sollte das aber besser unterlassen.

Einige spezielle Konstanten stellt MATLAB zur Verfügung:

pi	3.14159265....
i	Imaginärteil von $\sqrt{(-1)}$
j	das gleiche wie ***i***
eps	relativer Gleitpunktfehler, 2^{-52}
realmin	kleinstmögliche Gleitpunktzahl, 2^{-1022}

realmax größtmögliche Gleitpunktzahl, $(2-eps)*2^{1023}$

inf ∞

NaN keine Zahl (Not a Number)

Die Zahl ***inf*** entsteht z.B. bei der Division einer endlichen positiven Zahl durch Null (z.B. 123.4/0) oder bei der Berechnung von mathematischen Ausdrücken bei denen ein Überlauf auftritt, d.h. das Ergebnis würde größer als ***realmax*** werden. ***NaN*** entsteht bei Ausdrücken wie 0/0 oder ***Inf*** - ***Inf***, für die keine eindeutigen Ergebnisse existieren.

Die Namen der Konstanten und Funktionen sind nicht von MATLAB reserviert, d.h. es ist jederzeit möglich, sie zu überschreiben und somit als eigene Variablen- oder Funktionsnamen zu verwenden. Beispiel:

```
eps = 1.e-6
```

Nun kann ***eps*** als Variable mit diesem neuen Wert verwendet werden. Will man der Variablen oder Funktion wieder ihren ursprünglichen Wert zuweisen, so tippt man

```
clear eps
```

Ausdrücke

Beispiele von Ausdrücken mit ihren Ergebnissen:

```
rho = (1+sqrt(5))/2
rho =
    1.6180
a = abs(3+4i)
a =
    5
z = sqrt(besselk(4/3, rho-i))
z =
   0.3730 + 0.3214i
huge = exp(log(realmax))
huge =
  1.7977e+308
toobig = pi*huge
toobig =
   Inf
```

Beispiel 2.2

Vorsicht:
log bezeichnet in MATLAB den natürlichen Logarithmus (ln). Der Zehnerlogarithmus heißt ***log10***.
abs bezeichnet den Betrag einer komplexen Zahl, nicht aber den Betrag eines Vektors oder einer Matrix. Die diesbezügliche Funktion heißt ***norm***.

Übungsaufgaben

1. Wieviel verschiedene MATLAB Variablen stehen in folgender Zeile?
   anna ANNA anNa aNna_anna

   Lösung 2.1

2.► Bestimmen Sie Real- und Imaginärteil von

   a) $z = \dfrac{1}{\frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}}$

   b) $z = \dfrac{2+3i}{4-i} + \dfrac{2-3i}{1+4i}$

   c) $z = \sin(1+\sqrt[3]{1-i})$

   d) $z = 1+\sin\sqrt{2+i}$

   Lösung 2.2

3.► Berechnen Sie die Polardarstellung dieser komplexen Zahlen.
   Hinweis: MATLAB-Funktionen ***angle*** und ***abs***

   a) $z = (1-3i)^5 + (1+3i)^5$

   b) $z = (\frac{1}{2} + i\frac{\sqrt{3}}{2})^{50}$

   c) $z = \dfrac{(2-i)^7}{|-1+i|\cdot(-1+i)}$

   d) $z = 5^{-1+2i} + (-1+2i)^5$

   Lösung 2.3

4. Berechnen Sie e^{1+2i}, $(1+2i)^e$ und $(1+2i)^{(1+2i)}$

   Lösung 2.4

5.► Bestimmen Sie alle Lösungen der Gleichung $|b+2i| + |8-4i| = 30$

   Lösung 2.5

6. Berechnen Sie die nachfolgenden Ausdrücke in MATLAB. Sind die Ergebnisse die, die Sie erwarten?

   a) 1^∞

   b) 2^∞

   c) $\exp(\infty), \exp(-\infty)$

   d) $\mathrm{sign}(\mathrm{NaN}), \mathrm{sign}(-\mathrm{NaN})$

e) NaN^0

f) ∞^0

g) 1^{NaN}

h) $\log(\infty), \log(-\infty), \log(0)$

i) $1/\mathrm{inf}$

j) $\mathrm{inf} + \mathrm{inf}, \mathrm{inf} - \mathrm{inf}$

k) $-1 \cdot \mathrm{inf}$

Lösung 2.6

3. Arbeiten mit Matrizen

Dieser Abschnitt zeigt Ihnen andere Wege zur Erstellung von Matrizen.

Erstellen von Matrizen

MATLAB stellt vier Funktionen zum Erzeugen grundlegender Matrizen zur Verfügung:

zeros	Alle Elemente sind Nullen (Nullmatrix)
ones	Alle Elemente sind Einsen
rand	Gleichverteilte Zufallszahlen aus [0,1[
randn	normalverteilte Zufallszahlen (Mittelwert 0 und Varianz 1)

Einige Beispiele:

```
Z = zeros (4,2)
Z =
     0     0
     0     0
     0     0
     0     0
F = 5 * ones(3,3)
F =
     5     5     5
     5     5     5
     5     5     5
N = fix(10*rand(1,10))
N =
     9     2     6     4     8     7     4     0     8     4
R = randn(4,4)
R =
   -0.4326   -1.1465    0.3273   -0.5883
   -1.6656    1.1909    0.1746    2.1832
    0.1253    1.1892   -0.1867   -0.1364
    0.2877   -0.0376    0.7258    0.1139
```

Beispiel 3.1

load

Das Kommando ***load*** liest binäre Dateien ein, die Matrizen enthalten, welche in früheren MATLAB-Sitzungen erzeugt wurden. ***load*** liest auch Textdateien ein, die numerische Daten tabellarisch enthalten. Die Zahlen werden durch Leerzeichen getrennt. Erstellen Sie z.B. mit einem Texteditor eine Datei, die folgende Einträge enthält:

```
16.0   3.0    2.0    13.0
5.0    10.0   11.0   8.0
9.0    6.0    7.0    12.0
4.0    15.0   14.0   1.0
```

Speichern Sie diese unter **magik.dat**. Der Befehl

```
load magik.dat
```

liest die Datei und speichert ihren Inhalt in einer Matrix namens **magik**.

M-Files

Sie können Ihre eigenen Matrizen erstellen, indem Sie M-Files benutzen. M-Files sind Dateien, die MATLAB-Code enthalten. Erstellen Sie einfach eine Datei, die die selbe Art Anweisungen enthält, die Sie in MATLAB-Kommandozeilen eingeben würden. Geben Sie der Datei einen Namen mit der Erweiterung **.m**

Erzeugen Sie z.B. mit einem Texteditor eine Datei, die die folgenden fünf Zeilen enthält.

```
A = [
    16.0     3.0     2.0     13.0
    5.0      10.0    11.0    8.0
    9.0      6.0     7.0     12.0
    4.0      15.0    14.0    1.0   ];
```

Speichern Sie diese Datei unter dem Namen **magik.m** ab. Die Anweisung

```
magik
```

lädt die Datei und erzeugt die Variable A, die Ihre Beispiel-Matrix enthält.

Zusammensetzen von Matrizen

Angenommen Sie haben eine Matrix A bereits erstellt. Mit Hilfe der eckigen Klammern kann man daraus neue Matrizen zusammensetzen

Der Verkettungsoperator ist das eckige Klammerpaar [ ]. Fangen Sie mit dem magischen Quadrat A aus Kapitel 1 an und erstellen Sie

```
B = [ A A+32; A+48 A+16]
```

Das Resultat ist eine 8x8 Matrix, die Sie erhalten, wenn Sie die 4 Untermatrizen aneinanderhängen.

```
B =
    16     3     2    13    48    35    34    45
     5    10    11     8    37    42    43    40
     9     6     7    12    41    38    39    44
     4    15    14     1    36    47    46    33
    64    51    50    61    32    19    18    29
    53    58    59    56    21    26    27    24
    57    54    55    60    25    22    23    28
    52    63    62    49    20    31    30    17
```

Beispiel 3.2

Diese Matrix ist fast ein weiteres magisches Quadrat. Die Elemente sind eine Anordnung der ganzen Zahlen 1 bis 64. Die Spaltensummen haben einen identischen Wert, wie bei einem magischen Quadrat im Format 8x8.

```
sum(B)
ans =
   260   260   260   260   260   260   260   260
```

Beispiel 3.3

Aber die Zeilensummen, ***sum*(B')'** sind nicht alle gleich. Einige weitere Manipulationen sind nötig, um aus B ein magisches Quadrat zu machen.

Löschen von Zeilen und Spalten

Sie können Zeilen und Spalten einer Matrix löschen, indem Sie diesen einfach ein leeres Paar eckiger Klammern zuweisen. Fangen Sie an mit:

```
X = A;
```

Löschen Sie anschließend die 2. Spalte von X in dem Sie

```
X(:, 2) = [ ]
```

schreiben. Das Resultat ist

```
X =
    16     2    13
     5    11     8
     9     7    12
     4    14     1
```

Beispiel 3.4

Falls Sie ein einzelnes Element der Matrix löschen wollen, ist das Ergebnis keine Matrix mehr. Man muß immer komplette Zeilen und/oder Spalten löschen, um eine reduzierte Matrix zu erhalten.

So liefert der Ausdruck

```
X(1,2) = [ ]
```

eine Fehlermeldung. Dagegen liefert

```
X(2:2:10) = []
```

das Ergebnis

```
X =
    16     9     2     7    13    12     1
```

Beispiel 3.5

Hier wurde die komplette Matrix X als Vektor aufgefaßt, was durch den einfachen Index in den runden Klammern zum Ausdruck kommt. (vergl. Kapitel 1, Indizes)

Übungsaufgaben

1. Erzeugen Sie mit der Funktion ***rand*** zehn gleichmäßig verteilte Zufallszahlen im Intervall zwischen $-\pi$ und π.

   Lösung 3.1

2. Erzeugen Sie mit der Funktion ***randn*** tausend normalverteilte Zufallszahlen mit Mittelwert -5.5 und Standardabweichung 0.25.

   Lösung 3.2

3. Erzeugen Sie eine 6x6 obere (untere) Dreiecksmatrix mit Zufallszahlen zwischen 0 und 1. Hinweis: MATLAB-Funktion ***triu*** bzw. ***tril***

   Lösung 3.3

4. Erzeugen Sie eine 10x10 symmetrische Matrix $\mathbf{S}$ mit Zufallszahlen zwischen 0 und 8.

   Lösung 3.4

5. Es kann wichtig sein eine große Matrix in Untermatrizen zu zerlegen. Beschreiben Sie die Befehle oder Funktionen von MATLAB, die die folgenden Aufgaben ausführen. Dabei ist $\mathbf{A}$ eine 20x30 Matrix.

   a) Erzeugen Sie die Untermatrix von $\mathbf{A}$ von Zeile 15 bis 20 und Spalte 5 bis 10.

   b) Fügen Sie in $\mathbf{A}$ eine 5x10 Matrix $\mathbf{B}$ ein, beginnend bei Zeile 10 und Spalte 20.

   c) Erzeugen Sie eine 50x50 Matrix der Form $\mathbf{B} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^{\mathrm{T}} \end{bmatrix}$

   Lösung 3.5

6. Die MATLAB-Funktion ***rand*** erzeugt eine n x n-Matrix, deren Einträge gleichmäßig verteilte Zufallszahlen aus dem Intervall (0,1) sind. Schreiben Sie ein Funktion-File (M-File), das n x n-Matrizen erzeugt, deren Einträge Zahlen aus {1,2,3,4,5,6} sind.

   Lösung 3.6

4. Das Befehlsfenster

Bisher haben Sie nur die Befehlszeilen, zum Eingeben von Befehlen und Ausdrücken gebraucht und das Befehlsfenster zur Ausgabe. Dieser Abschnitt beschreibt ein paar Möglichkeiten das Aussehen des Befehlsfensters zu verändern. Falls es auf Ihrem System möglich ist Schriftarten für das Befehlsfenster auszuwählen, sollten Sie eine Schrift mit fester Zeichenbreite, wie z.B. **Fixedsys** oder **Courier**, wählen, um keine störenden Verschiebungen zu erhalten.

Der *format* - Befehl

Der ***format***-Befehl kontrolliert das numerische Format der durch MATLAB dargestellten Zahlen. Der Befehl hat nur eine Auswirkung auf die Darstellung der Zahlen und nicht auf das Verhalten von MATLAB und wie es mit den Zahlen rechnet oder sie speichert. Es folgen die unterschiedlichen Formate mit ihren entsprechenden Ausgaben:

```
x = [4/3 1.2345e-6]
x =
 1.3333    0.0000
```

format short

```
    1.3333    0.0000
```

format short e

```
    1.3333e+000      1.2345e-006
```

format short g

```
    1.3333    1.2345e-006
```

format long

```
    1.333333333333330.00000123450000
```

format long e

```
    1.33333333333333e+000 1.23450000000000e-006
```

format long g

```
    1.333333333333331.2345e-006
```

format bank

```
    1.33      0.00
```

format rat

```
    4/3       1/810045
```

format hex

```
    3ff55555555555553eb4b6231abfd271
```

Beispiel 4.1

Sollte das größte Element einer Matrix größer als 10^3 oder das kleinste kleiner als 10^{-3} sein, so klammert MATLAB einen Skalierungsfaktor bei ***short***- oder ***long***-Formaten aus.

Zusätzlich zu den oben gezeigten format Befehlen gibt es noch

format compact

Dieser Befehl unterdrückt alle Leerzeilen, die normalerweise zur Übersichtlichkeit dienen sollen. Auf diese Weise ist es möglich, mehr Informationen im Fenster zu sehen. Mit dem Befehl

format +

schliesslich erreichen Sie, dass lediglich das Vorzeichen der Zahlen ausgegeben wird. Das ist manchmal sinnvoll, um Strukturen in großen Matrizen erkennen zu können.

Mehr Kontrolle über das Ausgabeformat erlauben die Funktionen ***sprintf*** und ***fprintf***, die der Programmiersprache C entlehnt sind.

Ausgabe unterdrücken

Wenn Sie einen Ausdruck eingeben und anschließend die Eingabe-Taste drücken, gibt MATLAB automatisch das Ergebnis auf dem Bildschirm aus. Beenden sie die Zeile jedoch mit einem Strichpunkt, so führt MATLAB zwar die Berechnung durch, gibt das Ergebnis aber nicht aus. Dies ist besonders hilfreich, wenn große Matrizen berechnet bzw. erzeugt werden.

Beispiel:

```
A = magic(100);
```

Die Variable A enthält nun ein magisches Quadrat. Es erfolgte jedoch keine Ausgabe.

Lange Befehlszeilen

Sollte ein Ausdruck nicht in eine Zeile passen, so verwenden Sie drei Punkte ... um zu kennzeichnen, dass der Ausdruck in der nächsten Zeile fortgeführt wird. Beispiel:

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
      - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Leerzeichen vor und nach =, + und - Zeichen sind nicht nötig, aber erlaubt. Sie erhöhen manchmal die Lesbarkeit.

Editieren der Befehlszeile

Pfeil- und Kontrolltasten geben die Möglichkeit, bereits eingegebene Ausdrücke und Befehle zu editieren, zu wiederholen oder sie zu widerrufen. Nehmen wir an, Sie haben folgende Zeile eingegeben:

```
rho = (1 + sqt(5))/2
```

Dabei haben Sie ***sqrt*** falsch eingetippt und erhalten die folgende Ausgabe

```
Undefined function or variable 'sqt'
```

Anstatt die Zeile wieder neu einzutippen drücken Sie besser die „Pfeil-Nach-Oben“-Taste, um die obige Zeile erneut anzuzeigen. Anschließend benutzen Sie die „Pfeil-Nach-Links“-Taste, um den Fehler zu lokalisieren und schließlich zu korrigieren, indem Sie einfach das fehlende *r* ergänzen. Weiteres Drücken der „Pfeil-Nach-Oben“-Taste führt zu früher eingetippten Befehlszeilen. Wenn Sie erst einige Zeichen eintippen und anschließend die „Pfeil-Nach-Oben“-Taste betätigen, so sucht MATLAB nach einer passenden früher eingegebenen Zeile, deren Anfangszeichen mit der Vorgabe übereinstimmen.

Übungsaufgaben

1. Stellen Sie das Format ***bank*** ein und berechnen Sie die Determinante der Matrix $\mathbf{A} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{bmatrix}$ mit dem MATLAB-Befehl ***det***.

   a) Die Matrix $\mathbf{A}$ ist singulär. Beschreiben Sie die Lösungsmenge des Systems $\mathbf{Ax} = \mathbf{b}$, falls $\mathbf{b} = [0.1 \quad 0.3 \quad 0.5]^T$ ist:

   b) Stellen Sie nun wieder das Format ***short*** ein und berechnen Sie noch mal die Determinante.

   c) Wenn Sie die Matrix $\mathbf{A}$ in einen Rechner mit binärer Gleitpunktdarstellung eingeben, so ist sie nicht länger exakt singulär. Nicht alle Einträge der Matrix $\mathbf{A}$ sind binär exakt darstellbar. Somit ist es möglich, dieses Gleichungssystem mit dem Gauss-Verfahren zu lösen. Lösen Sie das System in MATLAB und vergleichen Sie die berechnete Lösung mit der oben beschriebenen Lösungsmenge.

   Lösung 4.1

2. Erzeugen Sie zwei 10x10-Zufallsmatrizen mit sehr kleinen Werten. Berechnen Sie einmal im Format ***bank*** und anschließend im Format ***short*** die Differenz der beiden Matrizen und vergleichen Sie die Ergebnisse.

   Lösung 4.2

5. Graphiken

MATLAB hat besondere Fähigkeiten, Vektoren und Matrizen als Graphen darzustellen, die man auch beschriften und drucken kann. Dieser Abschnitt zeigt eine der wichtigsten graphischen Funktionen und Beispiele für typische Anwendungen.

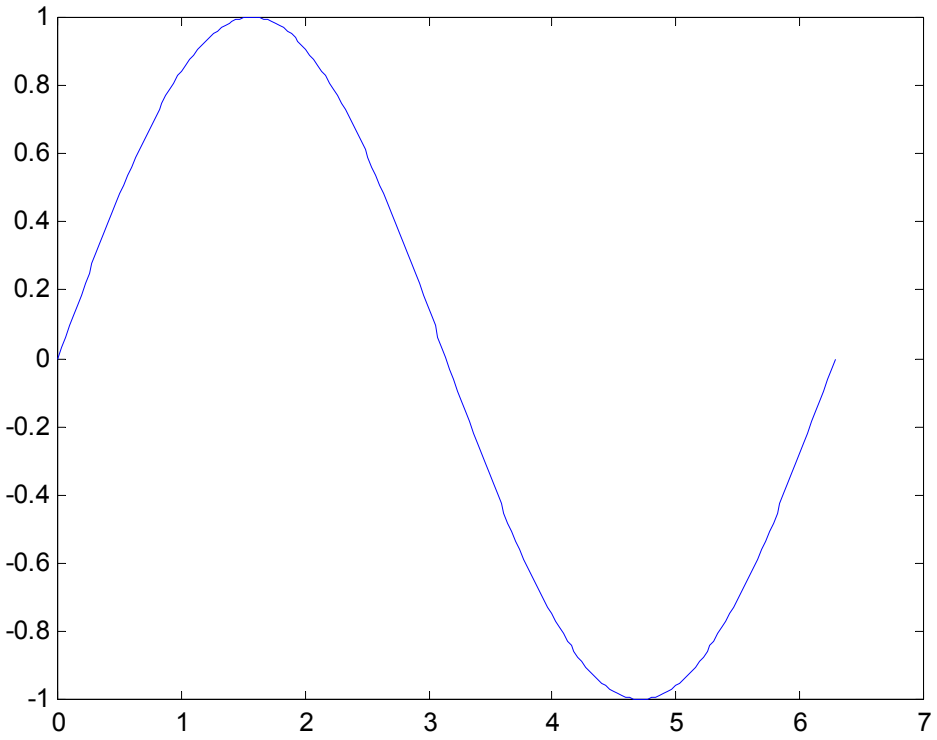
Erstellen eines Plots

Die Funktion ***plot*** hat verschiedene Formen. Diese hängen von den eingegebenen Argumenten ab. Wenn y ein Vektor ist, erzeugt ***plot*(y)** einen stückweisen linearen Graphen der Elemente von y über dem Index der Elemente von y. Falls Sie zwei Vektoren x,y als Argumente angeben, erzeugt ***plot*(x,y)** einen Graphen aus y-Werten über der x-Achse.

Um die Werte der Sinusfunktion von 0 bis 2π darzustellen, geben Sie z.B. ein

```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
```

und erhalten die Graphik

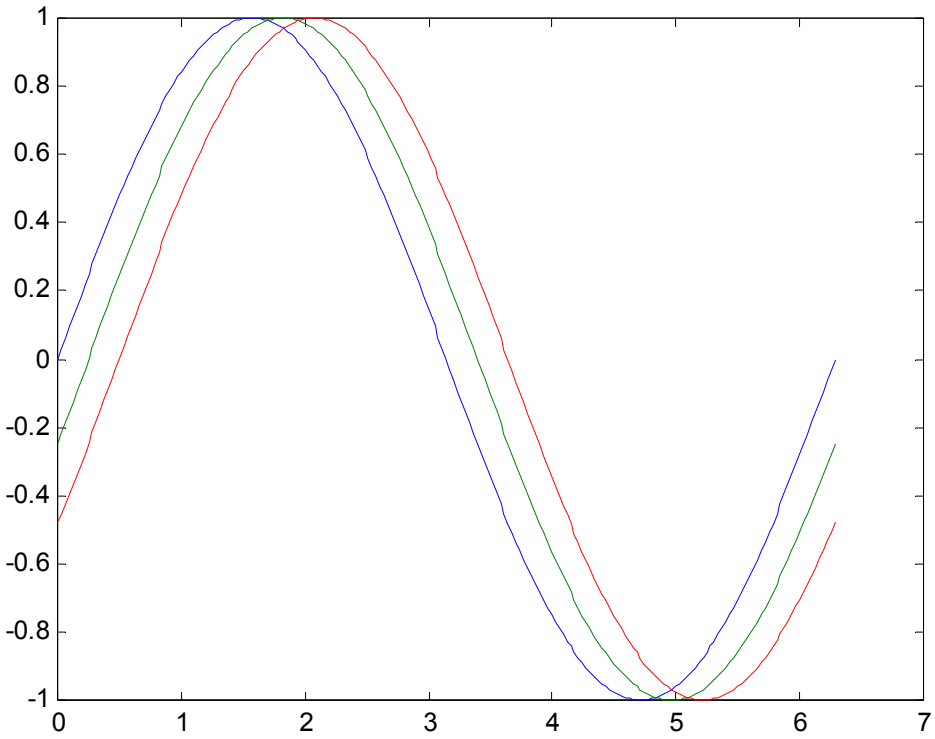


Beispiel 5.1

Mehrere xy-Paare in einem einzigen Aufruf von ***plot*** erzeugen gleich mehrere Graphen. MATLAB verwendet automatisch eine vordefinierte (aber auch vom Benutzer änderbare) Farbliste, um die Unterscheidung zwischen den verschiedenen Graphen zu erleichtern.

Beispiel: die Darstellung von 3 verschobenen Sinuskurven, jeder Graph erhält eine andere Farbe:

```
t = 0:pi/100:2*pi;
y = sin(t);
y2 = sin(t-.25);
y3 = sin(t-.5);
plot(t,y,t,y2,t,y3)
```



Beispiel 5.2

Es ist möglich Farben, Linien und Plotsymbole festzulegen:

```
plot(x,y, color_style_marker)
```

color_style_marker ist eine Funktion mit eins, zwei oder drei Argumenten. Die Übergabeargumente sind Farbe, Linientyp und Plotsymbol:

- Als Farben kann man c, m, y, r, g, b, w oder k angeben, gleichbedeutend mit **c**yan (hellblau), **m**agenta (violett), **y**ellow (gelb), **r**ed (rot), **g**reen (gruen), **b**lue (blau), **w**hite (weiß), und **b**lack (schwarz).

- Linientypen sind durchgezogen (-), gestrichelt (--), gepunktet (:), gestrichpunktet(.-) und außerdem ***none*** für keine Linie.
- Die am häufigsten verwendeten Plotsymbole sind +, o, * und x; es gibt aber eine ganze Reihe weiterer (siehe ***help plot***).

Der Beispielaufruf:

```
plot (x,y, 'r:+')
```

erzeugt einen roten gepunkteten Graphen und markiert jeden Datenpunkt mit einem Pluszeichen als Plotsymbol. Wenn das Plotsymbol festgelegt wird, aber kein Linientyp, dann zeichnet MATLAB nur die Datenpunkte.

Bildfenster

Die ***plot***-Funktion öffnet automatisch ein neues Bildfenster, falls noch kein Bildfenster vorhanden ist. Falls ein Bildfenster existiert, verwendet Plot dieses Fenster als Standard. Um ein neues Fenster zu öffnen und es zur aktuellen Graphikausgabe zu verwenden, geben Sie

```
figure
```

ein. Um ein vorhandenes Bildfenster als aktuelle Abbildung zu definieren, geben Sie

```
figure(n)
```

ein, wobei *n* die Nummer in der Titelleiste des Bildfenster ist. Die Ausgabe der darauf folgenden graphischen Kommandos erfolgt in diesem Fenster.

Graphiken erweitern

Das ***hold***-Kommando ermöglicht es, Teile zu einer bereits vorhandenen Graphik hinzuzufügen. Bei der Eingabe von

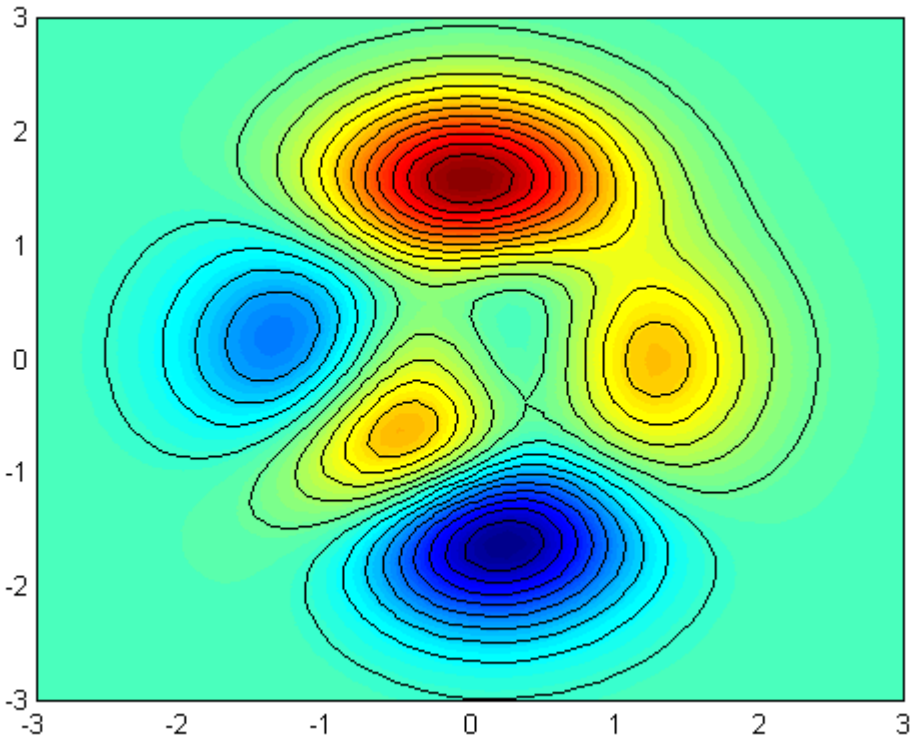
```
hold on
```

fügt MATLAB die Bild-Daten zur aktuellen Graphik hinzu, ohne bereits vorhandene Bestandteile dieser Graphik zu löschen. Im folgenden, schon etwas fortgeschritteneren Beispiel werden zuerst Höhenlinien einer vorgegebenen Funktion ***peaks*** von 2 Variablen gezeichnet (***contour***) und anschließend wird eine Darstellung der selben Funktion mit Falschfarben darübergelegt (***pcolor***). Das Übereinanderlegen wird durch ein zwischengeschaltetes ***hold on*** ermöglicht. (Der ***shading***-Befehl sorgt übrigens für eine stufenlose Schattierung zwischen den Rasterpunkten der Graphik).

```
[x, y, z] = peaks;
contour(x,y,z,20,'k')
hold on
pcolor(x,y,z)
shading interp
```

Beispiel 5.3

Das ***hold on*** Kommando bewirkt also, dass der ***pcolor***-Plot mit dem ***contour*** -Plot in einem Bild kombiniert wird.



Teilbilder

Die ***subplot***-Funktion erlaubt es Ihnen, mehrere komplette Graphiken in einem Fenster anzuordnen (und sie evtl. gemeinsam auf einem Blatt Papier auszudrucken).

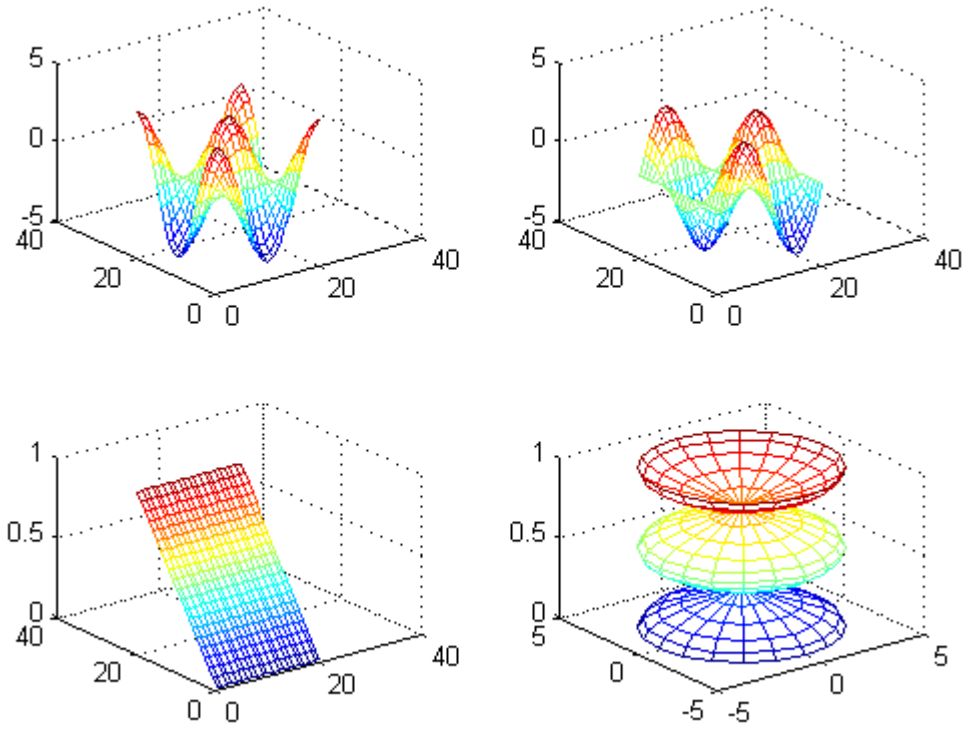
Geben Sie z.B.

```
subplot(2,3,4)
```

ein, so wird das aktuelle Bildfenster in eine 2x3-Matrix aus 6 gleich großen Teilbildern zerlegt und darin das 4. Teilbild als aktuelle Graphik definiert. Die Teilbilder sind zeilenweise durchnumeriert. Unser Teilbild steht also am Anfang der zweiten Zeile.

Das folgende Beispiel soll die Verwendung von 2x2 Teilbildern veranschaulichen. ***mesh*** ist ein Befehl zum Zeichnen von Gitternetzen.

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1)
mesh(X)
subplot(2,2,2); mesh(Y)
subplot(2,2,3); mesh(Z)
subplot(2,2,4); mesh(X,Y,Z)
```



Beispiel 5.4

Imaginäre und komplexe Daten

Beim Plotten komplexer Zahlenwerte werden Imaginärteile normalerweise ignoriert und eine Warnung im Befehlsfenster ausgegeben. Ausnahme: Wenn ***plot*** als einziges Argument einen Vektor mit komplexwertigen Komponenten erhält. In diesem besonderen Fall, wird ***plot*** als Befehl zur Darstellung des Realteiles der Vektorkomponenten gegenüber ihrem Imaginärteil aufgefaßt. Also ist für einen komplexen Vektor z der Befehl

```
plot(z)
```

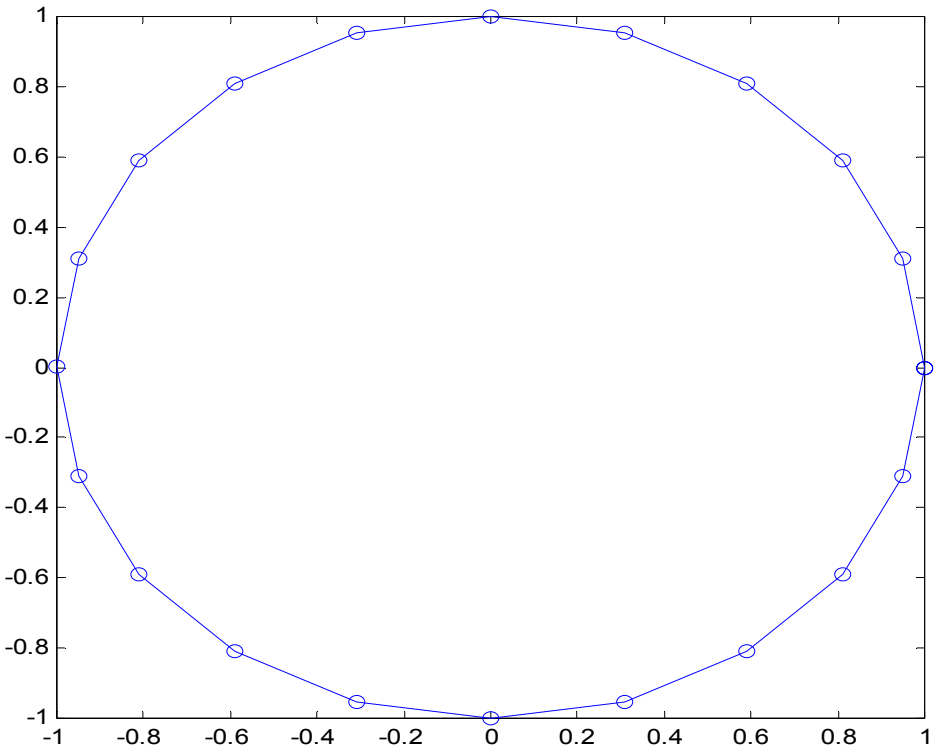
gleichbedeutend mit

```
plot(real(z), imag(z))
```

Beispiel:

```
t = 0:pi/10:2*pi;
plot(exp(i*t),'-o')
```

Hierin liefert die Exponentialfunktion *exp*(i*t) wegen des imaginären Parameters i $(i = \sqrt{-1})$ komplexe Zahlenwerte, die in einem Vektor zusammengefasst werden, der ebenso viele Komponenten hat, wie der Vektor t, nämlich 21.



plot zeichnet damit ein Polygon, dessen 20 Ecken durch kleine Kreise markiert sind. (Weil Anfangspunkt = Endpunkt sieht man im Bild nur 20 und nicht 21 Ecken).

Beispiel 5.5

Achsenwahl

Die ***axis***-Funktion hat eine Reihe von Optionen, um die Skalierung und die Seitenverhältnisse des Plots zu variieren.

Normalerweise bestimmt MATLAB die Maxima und Minima der Daten und wählt dann ein geeignetes Bildformat und eine passende Achsenskalierung selbständig. Die ***axis***-Funktion kann diese automatischen Vorgaben durch Parameterwerte überschreiben:

```
axis([xmin xmax ymin ymax])
```

axis akzeptiert auch eine Reihe von Schlüsselwörtern für die Achsendarstellung.

```
axis square
```

erzeugt x- und y-Achsen gleicher Länge und

```
axis equal
```

erzeugt x- und y-Achse im gleichen Maßstab mit gleicher Skalierung.

Beispiel:

```
plot(exp(i*t))
```

gefolgt von entweder ***axis square*** oder ***axis equal*** stellt das Oval aus dem vorangehenden Bild als Kreis dar.

```
axis auto
```

setzt die Achsenskalierung auf ihren Standardwert zurück.

```
axis on
```

schaltet die Achsen-Anzeige ein und

```
axis off
```

schaltet sie aus. Die Anweisung

```
grid on
```

führt zur Anzeige von Gitternetzlinien und

```
grid off
```

blendet diese wieder aus.

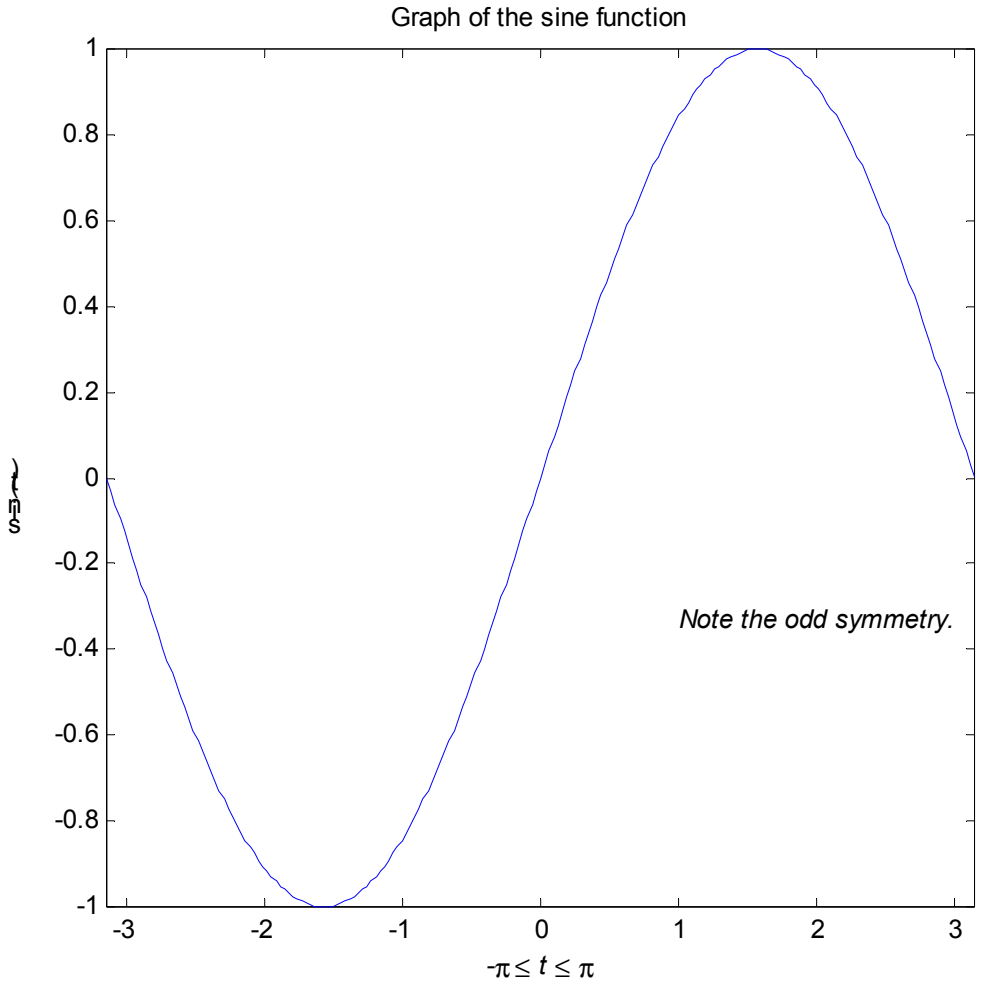
Beispiel 5.6

Achsenbeschriftung und Titel

Die ***xlabel***-, ***ylabel***- und ***zlabel***-Funktionen fügen zu den x-, y- und z-Achsen Beschriftungen hinzu. Die ***title***-Funktion fügt eine Beschriftung oberhalb des Graphen ein und die ***text***-Funktion fügt einen Text an einer beliebigen Stelle in der Zeichnung ein.

Eine Untermenge der Befehle des bekannten TeX-Editors erlaubt es, griechische Buchstaben, mathematische Symbole und andere Schriftarten darzustellen. Das folgende Beispiel benutzt die TeX-Notationen **\leq** für $\leq$, **\pi** für π und **\it** für kursive Schrift.

```
t=-pi:pi/100:pi;
y= sin(t);
plot(t,y);
axis([-pi pi -1 1])
xlabel('-\pi \leq {\itt} \leq \pi')
ylabel('sin(t)')
title('Graph of the sine function')
text(1,-1/3,'\it{Note the odd symmetry.}')
```



Beispiel 5.7

Gitter- und Flächen-Graphiken

MATLAB definiert eine Fläche durch z-Koordinaten über einem Gitter in der xy-Ebene. Benachbarte Raumpunkte werden durch Linien verbunden. Die Funktionen ***mesh*** und ***surf*** stellen Oberflächen dreidimensional dar. ***mesh*** erzeugt ein 3D-Gitter aus farbigen Linien. ***surf*** zeigt die Verbindungslinien mit eingefärbten Zwischenräumen.

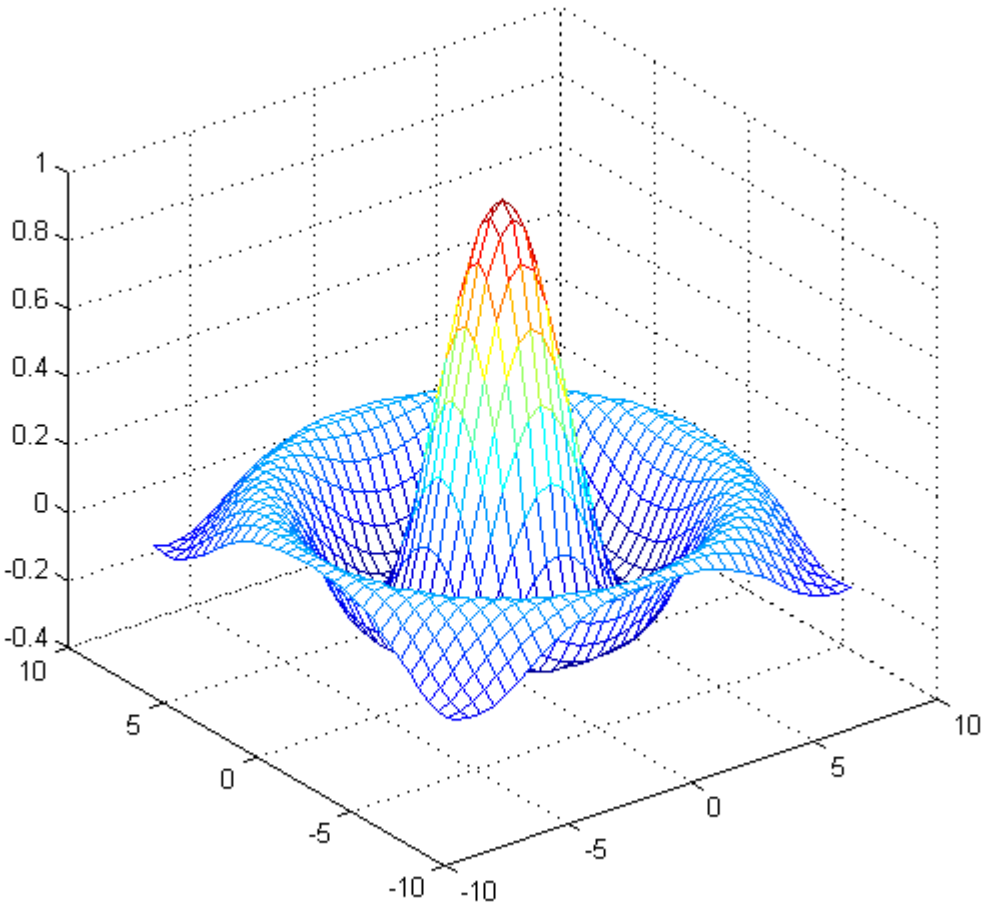
Visualisierung von Funktionen zweier Variabler

Um eine Funktion $z = f(x,y)$ von zwei Variablen graphisch darzustellen, generiert man zwei Matrizen X und Y, die die Koordinaten der Rasterpunkte in der xy-Ebene enthalten. Anschließend werden diese Matrizen benutzt, um die Funktion auszuwerten und graphisch darzustellen. Mit Hilfe der ***meshgrid***–Funktion lassen sich die Matrizen X und Y leicht erzeugen, wenn ein rechteckiges Gebiet der xy-Ebene zugrunde liegt. Als Parameter werden einfach die Rastervektoren angegeben (s.u.). Wenn die Rasterung in x- und y-Richtung übereinstimmt, genügt sogar ein einziger Parametervektor in ***meshgrid***. Sind X und Y berechnet, so erhält man mit Z = f(X, Y) die Matrix der z-Koordinaten über dem Raster in der xy-Ebene.

Beispiel:

Um die rotationssymetrische Funktion sin(r)/r mit $r = \sqrt{x^2 + y^2}$ graphisch darzustellen, geben Sie folgendes ein:

```
[X,Y] = meshgrid(-8:.5:8);
r = sqrt(X.^2 + Y.^2) +eps;
Z = sin(r)./r;
mesh(X,Y,Z)
```



Beispiel 5.8

In unserem Beispiel ist r der Abstand vom Nullpunkt. Der Nullpunkt liegt genau im Mittelpunkt des Rasters und ausgerechnet an dieser Stelle hätte unsere Funktion den undefinierten Wert 0/0. Durch Addition der sehr kleinen Zahl ***eps*** wird vermieden, dass dieser undefinierte Funktionswert im Raster vorkommt.

Bilder

Zweidimensionale Felder können als Bilder dargestellt werden, wobei die Feldelemente die Helligkeit oder Farbe der Bildpunkte festlegen.

```
load durer
whos
```

zeigt, dass die Datei **durer.mat** im Verzeichnis **demo** eine 648x509-Matrix X und eine 128x3-Matrix **map** enthält. Die Elemente von X sind ganze Zahlen zwischen 1 und 128, die als Farbpaletten-Index für **map** dienen.

Die Anweisungsfolge

```
image(X)
colormap(map)
axis image
```

reproduziert Dürers Kupferstich der am Anfang des Skripts gezeigt wurde. Ein hochaufgelöstes Bild des Magischen Quadrats ist in einer anderen Datei zu finden. Geben sie

```
load detail
```

ein und benutzen Sie die „Pfeil-nach oben"-Taste auf der Tastatur um das ***image***-, ***colormap***- und ***axis***-Kommando nochmals auszuführen. Die Eingabe

```
colormap(hot)
```

versieht den Kupferstich aus dem 16. Jahrhundert mit einer modernen Farbgebung.

Drucken von Graphiken

Die **Print**-Option im Datei-Menü und das ***print***-Kommando drucken beide MATLAB-Graphiken. Das Print-Menü zeigt eine Dialogbox an, die Sie zwischen den Druckoptionen auswählen lässt. Das ***print***-Kommando bietet mehr Flexibilität in der Art der Ausgabe und erlaubt es Ihnen, von M-Files aus zu drucken. Das Resultat kann direkt zum Drucker geschickt werden oder in einer angegebenen Datei gespeichert werden. Eine Vielfalt von Ausgabeformaten, inklusive PostScript, stehen zur Verfügung.

Das folgende Kommando sichert den Inhalt des aktuellen Bildfensters als Farb-Encapsulated-Level 2-Postscript in einer Datei mit dem Namen **magicsquare.eps**.

```
print –descp2 magicsquare.eps
```

Es ist wichtig zu wissen, welche Möglichkeiten Ihr Drucker bietet, bevor sie das ***print***-Kommando verwenden. Level 2-Postscript-Dateien sind generell kleiner und gestatten eine schnellere Druckausgabe als Level 1-Postscript-Dateien. Jedoch unterstützen nicht alle Postscript-Drucker den Level 2, daher sollten Sie zuerst herausfinden, welchen Standard Ihr Drucker unterstützt. MATLAB erzeugt eine stufenlose Ausgabe von Oberflächen und Farbverläufen, auch für schwarz/weiß-Ausgabegeräte. Funktionen und Text werden immer in schwarz/weiß gedruckt.

Übungsaufgaben

1. Das Polynom $(x-1)^6$ ist im Punkt $x=1$ Null und sonst positiv. Die ausmultiplizierte Form dieses Polynoms ist $x^6-6x^5+15x^4-20x^3+15x^2-6x+1$ und ist mathematisch äquivalent zu obigem Term, ergibt aber nicht die gleichen numerischen Ergebnisse. Berechnen und

zeichnen Sie die Werte dieses Polynoms für beide Darstellungen im Intervall $[0.995,1.005]$ für 101 im gleichen Abstand befindliche Punkte. Können Sie die Ergebnisse erklären?

Lösung 5.1

2. Die Fläche eines Dreiecks mit den Eckpunkten $P_1(x_1,y_1)$, $P_2(x_2,y_2)$, $P_3(x_3,y_3)$ errechnet sich folgendermaßen:

$$(x_3-x_1)\frac{y_1+y_3}{2}+(x_2-x_3)\frac{y_2+y_3}{2}-(x_2-x_1)\frac{y_1+y_2}{2}$$

Es kann leicht nachvollzogen werden, dass dies der folgende Ausdruck ist:

$$\frac{1}{2}\cdot\det\begin{bmatrix}x_1 & y_1 & 1\\ x_2 & y_2 & 1\\ x_3 & y_3 & 1\end{bmatrix}$$

In dieser Form ist offensichtlich, dass das Vorzeichen der Fläche davon abhängt, wie die Punkte in die Determinante genommen werden.
Berechnen Sie die Fläche des folgenden Dreiecks: $P_1(1,1)$, $P_2(3,1)$, $P_3(1,8)$ und zeichnen Sie das Dreieck.

Lösung 5.2

3. Um den Start eines Flugzeuges zu überwachen, wird die horizontale Position des Flugzeuges jede Sekunde von t = 0 bis t = 12 gemessen. Die Positionen (in Fuß) betragen:
0, 8.8, 29.9, 62.0, 104.7, 159.1, 222.0, 294.5, 380.4, 471.1, 571.7, 686.8, 809.2.

   a) Finden Sie die Kurve der kleinsten Quadrate $y=a+bt+ct^2+dt^3$ für diese Daten. Zeichnen Sie die Kurve.

   b) Berechnen Sie die Geschwindigkeit des Flugzeuges nach t = 4.5 Sekunden.

Lösung 5.3

4. Die Wertepaare

x	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
y	0.64	0.36	0.16	0.04	0.00	0.04	0.16	0.36	0.64	1.00

sollen nach der Methode der kleinsten Quadrate durch eine Funktion der Form

$$f(x)=a\cdot\frac{1}{x}+b\cdot\sqrt{x}+c\cdot e^x$$

approximiert werden. Zeichnen Sie das Ergebnis.
Approximieren Sie die Wertepaare anschließend durch ein Polynom 2.Grades und vergleichen Sie die Ergebnisse.

Lösung 5.4

5. Finden Sie eine Ebene, die am besten zwischen den folgenden Punkte liegt:
(1, 2, -1), (2, 3, -3), (3, 2, 5), (2, 1, 8), (2, 2, 3).
Zeichnen Sie das Ergebnis.

Lösung 5.5

6. Zeichen Sie mit der Funktion ***ezplot*** die Sinus- und Kosinusfunktion im Intervall $[-2\pi,2\pi]$.
In einem Graphikfenster sollen untereinander zuerst die Sinusfunktion, dann die Kosinus-

funktion und dann beide zusammen gezeichnet werden. Beschriften Sie auch die einzelnen Koordinatensysteme.

Lösung 5.6

7.► Berechnen Sie von den folgenden komplexen Zahlen den Real- und Imaginärteil und zeichnen Sie die Ergebnisse in unterschiedlichen Farben in ein Koordinatensystem ein. Verwenden Sie dazu die MATLAB-Funktion ***compass***. Fertigen Sie in einem zweiten Bild eine normale Zeichnung der komplexen Zahlen.

a) $z = \sqrt{\frac{\sin(2+i)}{2+i}}$

b) $z = e^{-i} + (-i)^{e}$

c) $z = \sqrt{\frac{e^{\pi(1+i)}}{\sin(\pi(1+i))}}$

d) $z = \left|(1-i)^{(1-i)}\right|$

Lösung 5.7

8. Zeichnen Sie $e^{i\pi/4}$, $e^{i\pi/2}$ und ihre Summe

Lösung 5.8

6. Hilfe und die Onlinedokumentation

Es gibt viele verschiedene Möglichkeiten, an die Informationen von MATLAB - Funktionen heranzukommen:

- Der ***help*** - Befehl
- Das Hilfe-Fenster
- Der MATLAB Help-Desk
- Online Bezugsseiten
- Link zu The MathWorks, Inc. Homepage im Internet

Der *help*-Befehl

Der ***help***-Befehl ist der einfachste Weg um die Syntax und das Verhalten einer speziellen Funktion festzustellen. Die Information wird direkt im Befehlsfenster angezeigt.

Beispiel:

```
help magic
```

gibt folgendes aus

```
MAGIC  Magic square.
    MAGIC(N) is an N-by-N matrix constructed from the integers
    1 through N^2 with equal row, column, and diagonal sums.
    Produces valid magic squares for N = 1,3,4,5,...
```

Beispiel 6.1

Anmerkung

MATLAB ***help***-Einträge verwenden für Funktions- und Variablennamen Großschreibung, um sie vom restlichen Text hervorzuheben. Wenn Sie aber selbst einen Funktionsnamen eingeben, müssen Sie ihn klein schreiben, weil MATLAB zwischen Groß- und Kleinschreibung unterscheidet.

Alle MATLAB-Funktionen sind in logischen Gruppen organisiert. Die Verzeichnisstruktur basiert auf dieser logischen Gruppierung. Zum Beispiel sind alle Funktionen zur Matrizenrechnung im Verzeichnis **matfun** zusammengefasst. Um sich alle diese Funktionsnamen mit einer Kurzbeschreibung ausgeben zu lassen, gibt man

```
help matfun
```

ein und erhält

```
Matrix functions - numerical linear algebra.

Matrix analysis.
   norm        - Matrix or vector norm.
   normest     - Estimate the matrix 2-norm.
   rank        - Matrix rank.
   det         - Determinant.
   trace       - Sum of diagonal elements.
   null        - Null space.
   orth        - Orthogonalization.
   rref        - Reduced row echelon form.
   subspace    - Angle between two subspaces.

Linear equations.
   \ and /     - Linear equation solution; use "help slash".
   inv         - Matrix inverse.
   cond        - Condition number with respect to inversion.
   condest     - 1-norm condition number estimate.
   chol        - Cholesky factorization.
   cholinc     - Incomplete Cholesky factorization.
   lu          - LU factorization.
   luinc       - Incomplete LU factorization.
   qr          - Orthogonal-triangular decomposition.
   lsqnonneg   - Linear least squares with nonnegativity constraints.
   pinv        - Pseudoinverse.
   lscov       - Least squares with known covariance.

Eigenvalues and singular values.
   eig         - Eigenvalues and eigenvectors.
   svd         - Singular value decomposition.
   gsvd        - Generalized ingular value decomposition.
   eigs        - A few eigenvalues.
   svds        - A few singular values.
   poly        - Characteristic polynomial.
   polyeig     - Polynomial eigenvalue problem.
   condeig     - Condition number with respect to eigenvalues.
   hess        - Hessenberg form.
   qz          - QZ factorization for generalized eigenvalues.
   schur       - Schur decomposition.

Matrix functions.
   expm        - Matrix exponential.
   logm        - Matrix logarithm.
   sqrtm       - Matrix square root.
   funm        - Evaluate general matrix function.

Factorization utilities
   qrdelete    - Delete column from QR factorization.
   qrinsert    - Insert column in QR factorization.
   rsf2csf     - Real block diagonal form to complex diagonal form.
   cdf2rdf     - Complex diagonal form to real block diagonal form.
   balance     - Diagonal scaling to improve eigenvalue accuracy.
   planerot    - Given's plane rotation.
   cholupdate  - rank 1 update to Cholesky factorization.

qrupdate    - rank 1 update to QR factorization.
```

Beispiel 6.2

Der Befehl

```
help
```

alleine gibt alle Verzeichnise mit einer kurzen Beschreibung der Funktionskategorie aus.

```
HELP topics:

general\Beispiele     -  (No table of contents file)
MATLAB\general        -  General purpose commands.
MATLAB\ops            -  Operators and special characters.
MATLAB\lang           -  Programming language constructs.
MATLAB\elmat          -  Elementary matrices and matrix manipulation.
MATLAB\elfun          -  Elementary math functions.
MATLAB\specfun        -  Specialized math functions.
MATLAB\matfun         -  Matrix functions - numerical linear algebra.
MATLAB\datafun        -  Data analysis and Fourier transforms.
MATLAB\polyfun        -  Interpolation and polynomials.
MATLAB\funfun         -  Function functions and ODE solvers.
MATLAB\sparfun        -  Sparse matrices.
MATLAB\graph2d        -  Two dimensional graphs.
MATLAB\graph3d        -  Three dimensional graphs.
MATLAB\specgraph      -  Specialized graphs.
MATLAB\graphics       -  Handle Graphics.
MATLAB\uitools        -  Graphical user interface tools.
MATLAB\strfun         -  Character strings.
MATLAB\iofun          -  File input/output.
MATLAB\timefun        -  Time and dates.
MATLAB\datatypes      -  Data types and structures.
MATLAB\winfun         -  Windows Operating System Interface Files (DDE/ActiveX)
MATLAB\demos          -  Examples and demonstrations.
wavelet\wavelet       -  Wavelet Toolbox.
wavelet\wavedemo      -  Wavelet Toolbox Demos.
fuzzy\fuzzy           -  Fuzzy Logic Toolbox.
fuzzy\fuzdemos        -  Fuzzy Logic Toolbox Demos.
toolbox\stats         -  Statistics Toolbox.
toolbox\ncd           -  Nonlinear Control Design Blockset
nnet\nnet             -  Neural Network Toolbox.
nnet\nndemos          -  Neural Network Demonstrations.
nnet\nnutils          -  (No table of contents file)
nnet\nnobsolete       -  (No table of contents file)
toolbox\splines       -  Spline Toolbox.
toolbox\optim         -  Optimization Toolbox.
toolbox\control       -  Control System Toolbox.
control\ctrlguis      -  Control System Toolbox -- GUI support functions.
control\obsolete      -  Control System Toolbox -- obsolete commands.
stateflow\sfdemos     -  Stateflow demonstrations and samples.
toolbox\sb2sl         -  SystemBuild to Simulink Translator
stateflow\stateflow   -  Stateflow
simulink\simulink     -  Simulink
simulink\blocks       -  Simulink block library.
simulink\simdemos     -  Simulink 3 demonstrations and samples.
simulink\dee          -  Differential Equation Editor
toolbox\tour          -  MATLAB Tour
MATLABR11\work        -  (No table of contents file)
toolbox\local         -  Preferences.

    For more help on directory/topic, type "help topic".
```

Beispiel 6.3

Das Hilfe-Fenster

Das MATLAB Hilfe-Fenster ist auf PCs durch das Anwählen der HelpWindow-Option im Menüpunkt **Help** oder durch das Anklicken des Fragezeichens auf der Menüleiste aufzurufen. Grundsätzlich ist es aber auf allen Computern durch das Eingeben von

```
helpwin
```

aufzurufen.

Beispiel 6.4

Um das Hilfe-Fenster auf ein bestimmte Thema anzuwenden, geben sie

```
helpwin [THEMA]
```

ein.

Das Hilfe-Fenster gibt Ihnen den Zugriff auf die gleichen Informationen wie der ***help***-Befehl, zusätzlich bietet es aber nützliche Links zu anderen Themen.

Der *lookfor*-Befehl

Der ***lookfor***-Befehl erlaubt es Ihnen, anhand eines Schlüsselwortes nach Funktionen zu suchen. Der Befehl untersucht die erste Zeile des ***help***-Textes jeder MATLAB-Funktion – man nennt sie auch die H1-Zeile – nach dem eingegebenen Schlüsselwort und gibt diese bei einem Fund aus. Beispiel: MATLAB besitzt keine Funktion namens „inverse“. Die Ausgabe von

```
help inverse
```

wird lauten

```
inverse.m not found
```

Aber

```
lookfor inverse
```

findet über ein Dutzend Treffer. Je nachdem, welche Toolboxen installiert sind, bekommt man z.B. folgende Ausgabe

```
INVHILB      Inverse Hilbert matrix.
ACOSH        Inverse hyberbolic cosine.
ERFINV       Inverse of the error function.
INV          Matrix inverse.
PINV         Pseudoinverse.
IFFT         Inverse discrete Fourier transform.
IFFT2        Two-dimensional inverse discrete Fourier transform.
ICCEPS       Inverse complex cepstrum.
IDCT         Inverse discrete cosine transform.
```

Wird die Option ***-all*** an den ***lookfor***-Befehl angehängt, so wird nicht nur die H1-Zeile, sondern alle H-Zeilen des ***help***-Texteintrags untersucht.

Beispiel:

```
lookfor -all inverse
```

Beispiel 6.5

Der *HelpDesk*

Der MATLAB-HelpDesk bietet einen guten Zugang zu mehr Hilfe und mehr Informationen auf der Festplatte oder CD-ROM. Viele dieser Informationen basieren auf HTML und werden somit durch einen Browser, wie z.B. den Netscape Browser oder Internet Explorer, dargestellt. Der HelpDesk wird über das Menü Help und den Punkt HelpDesk oder durch das Eintippen von ***helpdesk*** gestartet.

Für alle MATLAB-Operatoren und Funktionen gibt es eine HTML-Dokumentation die über den HelpDesk erreicht werden kann. Diese Texte enthalten genauere und ausführlichere Beschreibungen mit Beispielen. Außerdem gibt es auch noch weitere MATLAB-Dokumente.

Der *doc*-Befehl

Kennen Sie den Namen einer speziellen MATLAB-Funktionen, über die Sie näheres wissen wollen, so können sie die Online Hilfe direkt mit dem Befehl ***doc*** starten. Dabei wird automatisch ein Browser zur Darstellung benutzt.

Beispiel:

```
doc eval
```

doc öffnet den Browser, falls er noch nicht geöffnet ist.

Beispiel 6.6

Ausdrucken der Online-Hilfe

Verschiedene Versionen der Online-Hilfe sowie die restliche MATLAB Dokumentation ist auch im PDF-Format vorhanden. Man erhält sie durch den HelpDesk. Die PDF-Dokumente werden mit Hilfe des Adobe Acrobat Reader dargestellt. Dieses Format gibt sehr gut das Erscheinungsbild eines gedruckten Dokuments wieder, mit allen Schrifttypen, Graphiken, Formaten und Bildern. Mit dem Acrobat Reader können Sie die gewünschten PDF-Dokumente am besten anschauen und ausdrucken.

Internetverbindung zu MathWorks

Sollten sie über einen Internetzugang verfügen, so haben sie die Möglichkeit über den HelpDesk eine Verbindung zu MathWorks, den Herstellern von MATLAB, zu öffnen. Sie können Fragen per E-Mail stellen, Vorschläge machen und Fehler bekanntgeben. Ebenfalls können sie die Ergebnis-Such-Maschine bei MathWorks in Anspruch nehmen, um immer die aktuellsten Lösungsvorschläge und technischen Hinweise zu erhalten.

Übungsaufgaben

1. Informieren Sie sich mit der MATLAB-Hilfe wie man ein Skalarprodukt zweier Vektoren berechnet. Was macht der Befehl ***cross***?

   Lösung 6.1

2. In MATLAB gibt es die folgenden Rundefunktionen: ***ceil***, ***fix***, ***floor***, und ***round***. Informieren Sie sich wie jede dieser Funktionen rundet und berechnen Sie die folgenden Ausdrücke zunächst per Hand und überprüfen Sie dann Ihre Ergebnisse in MATLAB.
   a) round(-2.6)

b) fix(-2.6)
c) floor(-2.6)
d) ceil(-2.6)
e) floor(ceil(10.8))

Lösung 6.2

7. Die MATLAB Umgebung

Die MATLAB Umgebung enthält beides, die Variablen, die während der MATLAB Sitzung angelegt wurden, und die Dateien, die Programme enthalten und Daten zwischen den Sitzungen speichern.

Der Workspace

Der Workspace ist ein Speicherbereich auf den von der MATLAB Kommandozeile zugegriffen werden kann. Zwei Befehle, ***who*** und ***whos*** zeigen den aktuellen Stand des Workspace. Das ***who***-Kommando liefert eine kurze Liste zurück, während ***whos*** auch Matrixformate und Speicherinformation zurückliefert.

Hier ist eine Ausgabe die von ***whos*** erstellt wurde. Sie enthält einige Resultate der Beispiele aus diesem Buch. Sie zeigt verschiedene MATLAB Datentypen.

```
whos
Name          Size            Bytes  Class
  A             4x4               128  double array
  B             8x8               512  double array
  F             3x3                72  double array
  N             1x10               80  double array
  R            33x33             8712  double array
  X           359x371          1065512  double array
  Y            33x33             8712  double array
  Z            33x33             8712  double array
  ans           1x1                 8  double array
  caption       2x43              172  char array
  map          64x3              1536  double array
  t             1x201             1608  double array
  x            49x49             19208  double array
  y             1x201             1608  double array
  y2            1x201             1608  double array
  y3            1x201             1608  double array
  z            49x49             19208  double array
Grand total is 142440 elements using 1139004 bytes
```

Beispiel 7.1

Um alle Variablen aus dem Workspace zu löschen, geben Sie

```
clear
```

ein.

Der *save*-Befehl

Das ***save***-Kommando sichert den MATLAB Workspace in eine mat-Datei, diese kann mit dem ***load***-Kommando in einer späteren MATLAB Sitzung eingelesen werden. Z.B.:

```
save August17th
```

sichert den gesamten Workspace in die Datei **August17th.mat**. Es können auch einzelne ausgewählte Variablen durch die Angabe der Variablennamen nach dem Dateinamen gesichert werden.

Normalerweise werden die Variablen in Binärformat abgespeichert und können so schnell wieder von MATLAB eingelesen werden. Falls Sie die Dateien außerhalb von MATLAB verwenden wollen, können Sie ein alternatives Format angeben:

-ascii	8 Ziffern Text
-ascii –double	16 Ziffern Text
-ascii -double –tabs	Zahlen werden durch Tab getrennt
-v4	Erzeugt eine Datei für MATLAB 4
-append	Anhängen an eine bereits existierende mat – Datei

Wenn Sie den Inhalt des Workspace im Textformat speichern, sollten Sie in jeder Datei nur eine Variable sichern. Falls Sie mehr als eine Variable speichern, erstellt MATLAB eine Textdatei, die sie nicht einfach wieder in MATLAB laden können.

Der Suchpfad

MATLAB benutzt einen Suchpfad, das ist eine geordnete Liste von Ordnern und Unterverzeichnissen, um festzustellen, aus welchem Ordner aufgerufene Funktionen zu laden sind. Wenn eine Standardfunktion aufgerufen wird, führt MATLAB das erste m-File mit dem Funktionsnamen aus, das es in einem Verzeichnis des voreingestellten Suchpfades findet. Sie können dieses Verhalten ändern, in dem Sie private Verzeichnisse und Dateien dem Suchpfad hinzufügen.

Das Kommando

```
path
```

zeigt den Suchpfad auf Ihrem Rechner. Verwenden Sie aus dem File-Menü den Punkt **Set Path...**, um den Pfad anzeigen zulassen oder um ihn zu ändern. Sie können zu diesem Zweck aber auch den ***path***-Befehl in erweiterter Form aufrufen oder mit ***addpath*** Ordner und Unterverzeichnisse dem Suchpfad hinzufügen.

Beispiel 7.2

Dateiverarbeitung

Mit den Kommandos ***dir***, ***type***, ***delete*** und ***cd*** werden einige Systemkommandos zur Dateiverarbeitung zur Verfügung gestellt, die dem DOS-Betriebssystem entlehnt sind. Die Tabelle zeigt, wie diese Befehle in anderen Betriebssystemen heißen.

MATLAB	MS-DOS	UNIX	VAX/VMS
dir	dir	Ls	dir
type	type	Cat	type
delete	del oder erase	Rm	delete
cd	chdir oder cd	Cd	set default

Mit den meisten dieser Befehle können Pfadnamen, Wildcards und Laufwerksbezeichnungen verwendet werden.

Der *diary* Befehl

Der ***diary*** Befehl protokolliert Ihre MATLAB Sitzung in einer Datei. Sie können sich diese anzeigen lassen und den Text editieren, indem Sie ein Textverarbeitungsprogramm benutzen. Um eine Datei mit dem Namen "diary" zu erzeugen, die alle Befehle enthält, die Sie eingegeben haben, sowie auch die zugehörige Ausgabe von MATLAB (allerdings ohne Graphiken), geben Sie

```
diary
```

ein. Um die MATLAB Sitzung in eine Datei mit anderen Namen zu speichern, geben Die den Dateinamen an.

```
diary [Dateiname]
```

Um die Protokollierung der Sitzung zu beenden, geben Sie

```
diary off
```

ein.

Starten von externen Programmen

Ein Ausrufezeichen ! gibt an, dass der Rest der Zeile für das Betriebssystem gedacht ist, also kein MATLAB-Kommando enthält. Dies wird sinnvoll um Utilities oder andere Programme zu starten, ohne MATLAB zu beenden. Im Betriebsystem DOS oder VMS z.B. lädt

```
!edit magik.m
```

einen Editor mit dem Namen **edit** in dem die Datei **magik.m** zum editieren geöffnet wird. Wenn dieses externe Editorprogramm beendet wird, gibt das Betriebssystem an MATLAB die Kontrolle zurück.

Übungsaufgaben

1.* Die Datei *sterbewk.dat* enthält die Sterbewahrscheinlichkeiten für Männer und Frauen eines bestimmten Lebensalter. In der ersten Spalte steht das jeweilige Alter (von 1 bis 100) und in der zweiten und dritten Spalte steht die Wahrscheinlichkeit für Männer bzw. Frauen im Laufe des nächsten Jahres zu sterben.

   a) Lesen Sie die Daten ein und speichern Sie die Werte in drei Vektoren.

b) Zeichnen Sie den Sterblichkeitsverlauf für Männer als blaue Kurve und für Frauen als rote Kurve in ein Koordinatensystem. Zeichnen Sie zunächst ein Bild bis zum Alter 60 und dann ein zweites Bild für die Alter zwischen 10 und 30 Jahren.

c) Berechnen Sie die mittlere Lebenserwartung eines 20, 40, 60 bzw. 80-jährigen Mannes. Dabei gilt: Mittlere Lebenserwartung = Summe aller zukünftigen Lebenden : Anzahl der momentan Lebenden + 0,5.
Für die Anzahl der Lebenden eines bestimmten Alters braucht man eine Grundgesamtheit, z.B. 1.000.000, die sich jedes Jahr um die Anzahl der Sterbenden reduziert. (Die Anzahl der Sterbenden steckt in den Sterbewahrscheinlichkeiten.) Also gilt:
Anzahl Lebende = Anzahl Lebende im Vorjahr · (1 – Sterbewahrscheinlichkeit).

d) Erstellen Sie eine neue Tabelle, indem Sie jeweils von 10 Zeilen den Mittelwert bilden. Runden Sie dabei das Alter auf die nächst kleinere ganze Zahl ab. Die Sterbewahrscheinlichkeiten sollen als ganzzahlige Promillewerte gespeichert werden. Legen Sie mit !md einen Ordner C:\sterbewk auf Ihrer Festplatte an und speichern Sie die neue Tabelle im Ascii-Format für eine evtl. spätere Weiterverwendung darin ab.

Lösung 7.1

8. Mehr über Matrizen, Vektoren und Felder

Die folgenden Abschnitte zeigen Ihnen mehr über den Umgang und die Arbeit mit Matrizen und Vektoren in Zusammenhang mit Begriffen der linearen Algebra und Statistik.

Lineare Algebra

Eine Matrix ist ein zweidimensionales Feld, das eine lineare Transformation darstellt. Mathematische Operationen mit Matrizen sind Gegenstand der linearen Algebra.

Dürers magisches Quadrat:

```
A = magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

bietet eine Möglichkeit, Matrixoperationen mit MATLAB zu veranschaulichen. Sie haben bereits die Transposition A‘ einer Matrix A kennengelernt. Die Addition einer Matrix zu ihrer Transponierten ergibt eine symmetrische Matrix.

```
A + A'
ans =
    32     7    12    17
     7    22    17    22
    12    17    12    27
    17    22    27     2
```

Das Multiplikationssymbol *, bezieht sich auf die Matrix-Multiplikation, d.h. zur Berechnung jedes Matrixelements werden Skalarprodukte zwischen einer Zeile und einer Spalte gebildet. Die Multiplikation einer Matrix mit ihrer Transponierten ergibt eine symmetrische Matrix.

```
A' * A
ans =
   378   206   212   360
   206   370   368   212
   212   368   370   206
   360   212   206   378
```

Die Determinante unserer Matrix A ist 0 und zeigt somit an, dass A singulär ist, also keine Inverse hat.

```
d = det(A)
d =
```

```
     0
```

Den Rang von A findet man mit

```
r = rank(A)
r =
     3
```

Beispiel 8.1

Wie man hieraus erkennt, hat A nicht den Höchstrang 4.

Noch mehr Information liefert die MATLAB-Funktion ***rref***, die linear abhängige Zeilen in A eliminiert und die Restmatrix diagonalisiert, so weit das möglich ist. Aus

```
R = rref(A)
R =
  1     0     0     1
  0     1     0    -3
  0     0     1     3
  0     0     0     0
```

kann man weitere Eigenschaften von A ablesen

Dadurch dass unsere Matrix A singulär ist, besitzt sie auch keine Inverse. Sollten sie trotzdem versuchen, die Inverse auszurechnen, etwa durch

```
X = inv(A)
```

so erhalten sie folgende Warnmeldung

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.567374e-017.
X =
  1.0e+014 *
    0.9382    2.8147   -2.8147   -0.9382
    2.8147    8.4442   -8.4442   -2.8147
   -2.8147   -8.4442    8.4442    2.8147
   -0.9382   -2.8147    2.8147    0.9382
```

Beispiel 8.2

Rundefehler haben den Inversionsalgorithmus davon abgehalten, die Singularität von A exakt festzustellen, deshalb liefert er nur eine Warnung und keine Fehlermeldung. Der Wert von ***rcond***, der reziproken Konditionszahl, ist aber so nahe an ***eps***, der relativen Gleitpunktgenauigkeit, dass die Inverse von A wohl kaum auch nur mit minimaler Genauigkeit berechnet worden sein kann! Aus diesem Grund wird eine Warnung ausgegeben.

Das Format NxM einer Matrix A erhält man übrigens mit

```
[N,M] = size(A)
```

Die Eigenwerte von magischen Quadraten sind ebenfalls von Interesse

```
e = eig(A)
```

liefert

```
e =
   34.0000
    8.9443
   -8.9443
    0.0000
```

Beispiel 8.3

Einer der Eigenwerte ist 0, dies ist eine weitere Folge der Singularität von A. Der größte Eigenwert von A ist 34, dies ist unsere magische Summe. Dies liegt daran, dass der Vektor v, belegt mit lauter Einsen, ein Eigenvektor ist:

```
v = ones(4,1)
v =
     1
     1
     1
     1
```

Beispiel 8.4

Er gehört zum Eigenwert 34, denn 34*v = A*v:

```
A*v
ans =
    34
    34
    34
    34
```

Beispiel 8.5

Wird ein magisches Quadrat durch seine magische Summe dividiert,

```
P = A/34
P =
    0.4706    0.0588    0.0882    0.3824
    0.1471    0.3235    0.2941    0.2353
    0.2647    0.2059    0.1765    0.3529
    0.1176    0.4118    0.4412    0.0294
```

Beispiel 8.6

so erhält man eine Matrix, deren Zeilen- und Spaltensummen alle 1 ergeben.

Solche Matrizen enthalten Übergangswahrscheinlichkeiten in einem Markov-Prozess. Wiederholtes Potenzieren der Matrix P entspricht fortgesetzten Prozeßschritten. In unserem Beispiel ist die fünfte Potenz der Übergangsmatrix P

```
P^5
ans =
    0.2511    0.2491    0.2492    0.2506
    0.2495    0.2504    0.2502    0.2499
    0.2501    0.2498    0.2496    0.2505
    0.2494    0.2508    0.2509    0.2489
```

Beispiel 8.7

Wird die Matrix P k-mal potenziert und ist k sehr groß, so nähern sich die Elemente dem Zahlenwert ¼. – Probieren Sie es aus.

Schließlich lassen sich die Koeffizienten des charakteristischen Polynoms unserer Matrix A, die das Magische Quadrat enthält, ermitteln mit

```
a = poly(A)
a =
    1   -34    -8    2176    0
```

Beispiel 8.8

Dies zeigt, dass das charakteristische Polynom

$$\det(\mathbf{A}-\lambda\mathbf{E}) = \lambda^4 - 34*\lambda^3 - 8*\lambda^2 + 2176*\lambda$$

ist. Da die Matrix A singulär ist, hat der konstante Term den Faktor 0 und der kubische Faktor im charakteristischen Polynom ist 34 - wieder unsere magische Zahl!

Übrigens, mit $\mathbf{E}$ wird die Einheitsmatrix bezeichnet. In MATLAB wird sie von der Funktion ***eye*** generiert.

Beispiel 8.9

Elementweise Operationen

Matrizen kann man auch einfach als 2-dimensional angeordnete Zahlenfelder ansehen. Felder identischen Formats werden in praktischen Anwendungen oft durch arithmetische Operationen verknüpft, die *elementweise* ausgeführt werden. D.h. die Rechenoperation wirkt jeweils auf diejenigen Paare von Elementen zweier Felder, die an übereinstimmenden Positionen stehen.

Addition und Subtraktion von Matrizen sind ohnehin elementweise definiert, aber schon die Multiplikation funktioniert anders. MATLAB beinhaltet verschiedene weitere Operationen, die elementweise ausgeführt werden, und kennzeichnet diese durch einen Punkt vor dem Operationssymbol. Ausgenommen sind + und – Zeichen, da die beiden Operationen immer elementweise arbeiten.

Elementweise arbeitende Operatoren sind:

.*	Multiplikation
./	Division
.	Links-Division
.^	Potenzierung

Ebenfalls mit einem Punkt versehen kann man den Transpositions-Operator **'**. Bei der Transposition komplexer Matrizen werden normalerweise zusätzlich alle Elemente der Matrix durch ihre komplex konjugierten ersetzt. Das kann man unterdrücken durch den Operator

.'	Matrix–Transposition ohne komplexe Konjugation

Mit

```
A.*A
```

entsteht

```
ans =
   256     4     9   169
    25   121   100    64
    81    49    36   144
    16   196   225     1
```

Beispiel 8.10

Wird z.B. die Dürermatrix elementenweise mit sich selbst multipliziert. So erhält man ein Feld, welches die Quadrate aller ganzen Zahlen zwischen 1 und 16 in unregelmäßiger Anordnung enthält.

Elementweise Operationen sind nützlich, um Tabellen zu erstellen und zu verarbeiten. Wir belegen z.B. einen Spaltenvektor n mit

```
n = (0:9)
n =
     0
     1
     2
     3
     4
     5
     6
     7
     8
     9
```

Dann ergibt

```
pows = [n n.^2 2.^n]
pows =
        0     0     1
        1     1     2
        2     4     4
        3     9     8
        4    16    16
        5    25    32
        6    36    64
        7    49   128
        8    64   256
        9    81   512
```

die folgende Tabelle mit Quadraten und Zweierpotenzen der Elemente von n

Beispiel 8.11

Die elementaren mathematischen Funktionen wirken immer elementenweise. So erstellt z.B.

```
format short g;
x = (1:0.1:2)';
logs = [x log10(x)]
```

eine Logarithmentafel:

```
logs =
       1       0
     1.1     0.041393
     1.2     0.079181
     1.3     0.11394
     1.4     0.14613
     1.5     0.17609
     1.6     0.20412
     1.7     0.23045
     1.8     0.25527
     1.9     0.27875
       2     0.30103
```

Beispiel 8.12

Multivariate Daten

MATLAB verwendet Spalten für multivariate statistische Daten. Jede Spalte in einem Datensatz repräsentiert eine Variable und jede Zeile eine Stichprobe. Das (i,j)te Element ist die i-te Stichprobe für die j-te Variable.

Als Beispiel betrachten wir einen Datensatz mit drei Variablen:

- Herzschlagzahl
- Gewicht
- Übungsstunden pro Woche

Nach fünf Beobachtungen schaut das Ergebnisfeld vielleicht wie folgt aus:

```
D = [
        72   134   3.2
        81   201   3.5
        69   156   7.1
        82   148   2.4
        75   170   1.2 ]
D =
   72.0000  134.0000    3.2000
   81.0000  201.0000    3.5000
   69.0000  156.0000    7.1000
   82.0000  148.0000    2.4000
   75.0000  170.0000    1.2000
```

Die erste Zeile enthält die Herzschlagzahl, das Gewicht und die Übungsstunden pro Woche des ersten Patienten, die zweite Zeile enthält die Daten für den zweiten Patienten usw. . Nun können Sie viele von MATLAB's Analysefunktionen auf diesen Datensatz anwenden. Zum Beispiel um den Mittelwert ***mean*** und die Standardabweichung ***std*** jeder Spalte zu erhalten:

```
mu = mean(D), sigma = std(D)
mu =
   75.8000  161.8000    3.4800
sigma =
    5.6303   25.4990    2.2107
```

Beispiel 8.13

Um eine Liste aller statistischen Analysefunktionen in MATLAB zu erhalten geben sie

```
help datafun
```

ein.

```
Data analysis and Fourier transforms.

  Basic operations.
    max         - Largest component.
    min         - Smallest component.
    mean        - Average or mean value.
    median      - Median value.
    std         - Standard deviation.
    var         - Variance.
    sort        - Sort in ascending order.
    sortrows    - Sort rows in ascending order.
    sum         - Sum of elements.
    prod        - Product of elements.
    hist        - Histogram.
    histc       - Histogram count.
    trapz       - Trapezoidal numerical integration.
    cumsum      - Cumulative sum of elements.
    cumprod     - Cumulative product of elements.
    cumtrapz    - Cumulative trapezoidal numerical integration.

  Finite differences.
    diff        - Difference and approximate derivative.
    gradient    - Approximate gradient.
    del2        - Discrete Laplacian.

  Correlation.
    corrcoef    - Correlation coefficients.
    cov         - Covariance matrix.
    subspace    - Angle between subspaces.

  Filtering and convolution.
    filter      - One-dimensional digital filter.
    filter2     - Two-dimensional digital filter.
    conv        - Convolution and polynomial multiplication.
    conv2       - Two-dimensional convolution.
    convn       - N-dimensional convolution.
    deconv      - Deconvolution and polynomial division.
    detrend     - Linear trend removal.

  Fourier transforms.
    fft         - Discrete Fourier transform.
    fft2        - Two-dimensional discrete Fourier transform.
    fftn        - N-dimensional discrete Fourier Transform.
    ifft        - Inverse discrete Fourier transform.
    ifft2       - Two-dimensional inverse discrete Fourier transform.
    ifftn       - N-dimensional inverse discrete Fourier Transform.
    fftshift    - Shift DC component to center of spectrum.
    ifftshift   - Inverse FFTSHIFT.

  Sound and audio.
    sound       - Play vector as sound.
    soundsc     - Autoscale and play vector as sound.
    speak       - Convert input string to speech (Macintosh only).
    recordsound - Record sound (Macintosh only).
    soundcap    - Sound capabilities (Macintosh only).
    mu2lin      - Convert mu-law encoding to linear signal.
    lin2mu      - Convert linear signal to mu-law encoding.

  Audio file inport/export.
    auwrite     - Write NeXT/SUN (".au") sound file.
    auread      - Read NeXT/SUN (".au") sound file.
    wavwrite    - Write Microsoft WAVE (".wav") sound file.
    wavread     - Read Microsoft WAVE (".wav") sound file.
    readsnd     - Read SND resources and files (Macintosh only).
```

```
    writesnd    - Write SND resources and files (Macintosh only).
```

Beispiel 8.14

Falls Sie die Statistic Toolbox implementiert haben, geben Sie

```
help stats
```

Beispiel 8.15

ein, um die zusätzlichen Analysefunktionen dieser Toolbox kennen zu lernen.

Verwendung von Skalaren

Matrizen und Skalare können auf verschiedene Arten und Weisen kombiniert werden. Zum Beispiel kann ein Skalar zu einer Matrix elementenweise addiert oder subtrahiert werden. Der durchschnittliche Wert aller Elemente unseres magischen Quadrats A ist 8.5, so dass

```
B = A-8.5
```

eine Matrix B erzeugt, deren Spaltensummen alle 0 ergeben.

```
B =
          7.5         -5.5         -6.5          4.5
         -3.5          1.5          2.5         -0.5
          0.5         -2.5         -1.5          3.5
         -4.5          6.5          5.5         -7.5
sum(B)
ans =
     0     0     0     0
```

Beispiel 8.16

Die mathematisch korrekte Schreibweise wäre

```
B = A-8.5*eye(size(A))
```

wobei die ***eye*(*size*(*A*))** die Einheitsmatrix im Format von A darstellt. Da die Kurz-Schreibweise B = A – 8.5 aber keine Verwechslung zulässt, ist sie in MATLAB erlaubt.

Man kann auf diese Weise auch Teilbereichen einer Matrix skalare Werte zuweisen. So setzt z.B.

```
B(1:2,2:3) = 0
```

den angegebenen Bereich in der Matrix B auf Null.

```
B =
          7.5            0            0          4.5
         -3.5            0            0         -0.5
          0.5         -2.5         -1.5          3.5
         -4.5          6.5          5.5         -7.5
```

Beispiel 8.17

Logische Indizierung

Felder, die von logischen und relationalen Operationen erzeugt werden, können als Verweise auf Teilfelder dienen. Feldelemente ungleich 0 stehen für „wahr“, Nullen für „falsch“.

Nehmen Sie an, X ist irgend eine Matrix oder ein Vektor und L ist ein Feld im selben Format wie X. Mit X(L) werden diejenigen Elemente von X herausgegriffen, in deren Positionen L Elemente ungleich 0 enthält.

Beispiel: Angenommen, Sie haben die folgende Datenreihe vorliegen:

```
x = [2.1, 1.7, 1.6, 1.5, NaN, 1.9, 1.8, 1.5, 5.1, 1.8, 1.4, 2.2, 1.6, 1.8]
```

NaN ist ein Kennzeichnen für eine fehlende Stichprobe. Um fehlende Daten mit Hilfe von logischer Indizierung zu löschen, kann man die logische Funktion ***finite*** verwenden. ***finite*(*x*)** ist „wahr“ für alle endlichen Zahlenwerte und „falsch“ für ***NaN*** und ***Inf***.

```
L = finite(x); x = x(L)
```

entfernt das ***NaN***-Element aus x:

```
x =
  2.1  1.7  1.6  1.5  1.9  1.8  1.5  5.1  1.8  1.4  2.2  1.6  1.8
```

Diese Art der Indizierung kann auch direkt in einem Schritt erfolgen, ohne dass explizit ein Feld L erzeugt wird:

```
x = x(finite(x))
```

hätte die gleiche Wirkung gehabt.

Nun gibt es noch eine Stichprobe, die etwas von den anderen abweicht: 5.1 ist ein „Ausreißer“. Der folgende Ausdruck löscht Ausreißer. In unserem Beispiel wollen wir diejenigen Elemente entfernen, die mehr als die dreifache Standardabweichung vom Mittelwert entfernt sind.

```
x = x( abs(x-mean(x)) <= 3*std(x) )
x =
  2.1  1.7  1.6  1.5  1.9  1.8  1.5  1.8  1.4  2.2  1.6  1.8
```

Die Relation ≤ wird in MATLAB wie in anderen Programmiersprachen durch **<=** ausgedrückt. Angewandt auf Matrizen wirkt sie elementweise. Beachten Sie, dass in unserem Beispiel intern ein Vektor **L = *abs*(x-*mean*(x)) <= 3**std*(x)** erzeugt wurde, der in all denjenigen Positionen eine 1 enthält, in denen die angegebene Bedingung erfüllt ist, und sonst nur aus Nullen besteht. Mehr über Relationen erfahren Sie im nächsten Kapitel.

Ein weiteres Beispiel ist die Markierung der Primzahlen in Dürers magischem Quadrat. Alle anderen Matrixelemente wollen wir aus A entfernen. Hierzu verwendet man die logische Funktion ***isprime*** folgendermaßen

```
A(~isprime(A)) = 0
A =
     0     3     2    13
     5     0    11     0
     0     0     7     0
     0     0     0     0
```

Beispiel 8.18

Die Funktion ***isprime*(*A*)** liefert eine Matrix im Format von A, die in allen Primzahl-Positionen von A eine 1 enthält, in allen anderen Positionen eine 0. Der Operator ~ („not“) invertiert diese Wahrheitswerte: In der intern erzeugten Matrix **L = ~*isprime*(*A*)** stehen überall dort Einsen, wo das entsprechende Element von A keine Primzahl ist, und Nullen in den Primzahl-Positionen. Also werden durch A(L) diejenigen Positionen von A ausgewählt, die keine Primzahl enthalten. Mehr über MATLABs logische Operationen folgt im nächsten Kapitel.

Die *find* Funktion

Die ***find***-Funktion legt die Indizes von Elementen eines Feldes fest, welche bestimmten logischen Bedingungen unterliegen. In seiner einfachsten Form liefert ***find*** einen Spaltenvektor von Indizes zurück.

Beispiel:

```
k = find(isprime(A))'
```

ermittelt die Positionen der Primzahlen im magischen Quadrat.

```
k =
     2     5     9    10    11    13
```

Beispiel 8.19

Wie Sie sehen, wird dabei eindimensionale Indizierung verwendet. Die Elemente einer Matrix A werden einfach Spalte für Spalte fortlaufend durchnumeriert. Folgerichtig wäre auch k ein Spaltenvektor, den wir aber gerade noch zum Zeilenvektor transponiert haben.

Um die Primzahlen, welche in A gefunden wurden, in der durch k festgelegten Reihenfolge auszugeben, schreiben Sie einfach

```
Primzahlen = A(k)
```

Da A spaltenweise indiziert wird, entsteht

```
ans =
     5     3     2    11     7    13
```

k kann als Indexvektor bei einem Zuweisungsausdruck verwendet werden. Um etwa alle Primzahlen in A zu löschen kann man schreiben

```
 A(k) = NaN
```

und erhält

```
A =
     0   NaN   NaN   NaN
   NaN     0   NaN     0
     0     0   NaN     0
     0     0     0     0
```

Beispiel 8.20

Die Matrixstruktur von A bleibt bei diesem Vorgehen erhalten.

Übungsaufgaben

1. Erzeugen Sie die folgende Matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 32 \\ 2 & 4 & 8 & 16 & 32 & 64 \\ 4 & 8 & 16 & 32 & 64 & 128 \\ 8 & 16 & 32 & 64 & 128 & 256 \\ 16 & 32 & 64 & 128 & 256 & 512 \\ 32 & 64 & 128 & 256 & 512 & 1024 \end{bmatrix}$$

Lösung 8.1

2. Wir betrachten die folgende Matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 5.5 & 0.4 & 3.1 \\ 9.4 & 5.5 & 3.3 \\ -0.3 & 4.6 & -4.3 \\ 0.4 & -4.6 & 9.0 \\ 5.0 & 5.5 & 7.7 \end{bmatrix}$$

Geben Sie den Rang dieser Matrix an. Wie kann man in MATLAB die Größe von $\mathbf{A}$ bestimmen? Geben Sie alle Indizes an, deren Matrixelemente 5.5 sind.

Lösung 8.2

3.► Gegeben seien die beiden Matrizen $\mathbf{A} = \begin{bmatrix} 2 & -2 & -4 \\ -1 & 3 & 4 \\ 1 & -2 & -3 \end{bmatrix}$ und $\mathbf{B} = \begin{bmatrix} 1 & 0 & 3 \\ 5 & -2 & 7 \end{bmatrix}$.

Berechnen Sie, falls möglich, $\mathbf{AB}$, $\mathbf{BA}$, $\mathbf{A}^2$ und $\mathbf{B}^2$.

Lösung 8.3

4.► Berechnen Sie mit den Matrizen $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ und $\mathbf{D}$ folgende Matrizenalgebra.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 1 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -1 & 2 \\ 4 & -2 \\ 7 & -1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 5 \\ -5 & 3 \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} 4 & 3 & -2 \\ 1 & 0 & 5 \\ 2 & -1 & 6 \end{bmatrix}$$

a) $\mathbf{A} + \mathbf{B}$
b) $\mathbf{B} + \mathbf{C}$
c) $\mathbf{DA}$
d) $2\mathbf{A} - 3\mathbf{B}$
e) $\mathbf{A}^T$
f) $\mathbf{C}^2$

Lösung 8.4

5. Berechnen Sie die Matrix $\mathbf{X}$ aus folgender Gleichung:

$$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \\ 4 & 1 & 3 \end{bmatrix} \cdot \mathbf{X} = \begin{bmatrix} 5 & 0 & 0 \\ 3 & 7 & 0 \\ 10 & 15 & 5 \end{bmatrix}$$

Lösung 8.5

6.► Berechnen Sie

a) $\det \begin{bmatrix} 1 & 0 & 3 & -2 \\ 7 & 5 & 4 & 3 \\ 4 & 1 & -6 & 2 \\ -2 & 3 & 0 & 9 \end{bmatrix}$ b) $\det \begin{bmatrix} 4 & 5 & 2 & 1 \\ 3 & 9 & 5 & 7 \\ -2 & -3 & -1 & 0 \\ 1 & 5 & 8 & 2 \end{bmatrix}$

Lösung 8.6

7. Berechnen Sie die Inversen der folgenden Matrizen:

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 3 & 2 \\ 3 & 6 & 5 & 2 \\ 2 & 5 & 2 & -3 \\ 4 & 5 & 14 & 14 \end{bmatrix} \quad \mathbf{B} = \frac{1}{6}\begin{bmatrix} 3 & 3 & 3 & 3 \\ 3 & -5 & 1 & 1 \\ 3 & 1 & -5 & 1 \\ 3 & 1 & 1 & -5 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -3 & 5 & 1 & 0 \\ 4 & 2 & 3 & 1 \end{bmatrix}$$

Lösung 8.7

8. Es sei $\mathbf{A}$ die folgende Matrix: $\mathbf{A} = \begin{bmatrix} 4 & 3 & 0 \\ 3 & 6 & 2 \\ 0 & 2 & 4 \end{bmatrix}$

a) Bestimmen Sie die drei Eigenwerte der Matrix $\mathbf{A}$.

b) Bestimmen Sie zu jedem Eigenwert einen zugehörigen Eigenvektor.

c) Berechnen Sie $\det(\mathbf{A} - \lambda\mathbf{I})$ und zeigen Sie, daß dies für die Eigenwerte Null ist.

d) Zeigen Sie die Gültigkeit von $\mathbf{AQ} = \mathbf{QD}$, wobei $\mathbf{Q}$ diejenige Matrix ist, in deren Spalten die Eigenvektoren stehen, und $\mathbf{D}$ die Diagonalmatrix mit den Eigenwerten in der Diagonalen.

Lösung 8.8

9. Berechnen Sie das charakteristische Polynom der Matrix $\mathbf{B} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 3 & -5 & 2 \end{bmatrix}$.

Zeigen Sie, daß $\mathbf{B}^3 - 2\mathbf{B}^2 + 5\mathbf{B} - 3\mathbf{I} = 0$.

Lösung 8.9

9. Ablaufkontrolle

MATLAB hat fünf Kontroll-Anweisungen

if	(Verzweigung)
switch	(Mehrfachverzweigung)
while	(Schleifen)
for	(Zähler-Schleifen)
break	(Abbruch)

Logische Ausdrücke zur Ablaufkontrolle enthalten meistens relationale oder logische Operationen. MATLAB stellt hier folgende Operatoren zur Verfügung:

Relationen:

<	kleiner
<=	kleiner/gleich
>	größer
>=	größer/gleich
==	gleich
~=	ungleich

Von der Notation in anderen Programmiersprachen weicht eigentlich nur das ungleich-Zeichen ab. Beachten Sie, dass Relationen elementweise wirkende Operationen sind. Bei einem Vergleich, z.B. A > B, müssen beide Felder A und B das gleiche Format haben. Intern entsteht eine Matrix L, die in all den Positionen eine 1 enthält, die dem Vergleich standhalten, und sonst nur aus Nullen besteht.

Ähnlich verhält es sich bei logischen Operationen. MATLAB kennt

&	logisches „und”
|	logisches „oder”
~	logisches „nicht”
xor	exklusives „oder” (entweder/oder)
any	1 falls irgendein Element eines Spalte ungleich 0 (also „wahr”) ist
all	1 falls alle Elemente einer Spalte ungleich 0 (also „wahr“) sind.

if-Anweisung

Die ***if***-Anweisung wertet einen logischen Ausdruck aus und führt Gruppen von Anweisungen aus, wenn der Ausruck „wahr“ ist, also einen Wert ungleich 0 ergibt. Die optionalen Schlüsselwörter ***elseif*** und ***else*** bieten einen alternativen Block von Anweisungen an, die ausgeführt werden, wenn der Ausdruck falsch ist. Das Schlüsselwort ***end*** beendet immer den letzten Block der Anweisung. Die Anweisungsblöcke werden von je einem dieser vier Schlüsselwörter getrennt. Klammern, wie in C und verwandten Programmiersprachen üblich, werden in MATLAB nicht benötigt.

MATLABs Algorithmus zum Erzeugen von magischen Quadraten der Ordnung n beinhaltet drei verschiedene Fälle: n ungerade, n gerade aber nicht durch 4 teilbar, n gerade und durch 4 teilbar. Dies wird beschrieben durch

```
if rem(n,2) ~= 0,       'ungerade';
elseif rem(n,4) ~= 0,   'nur durch 2 teilbar';
else                    'durch 4 teilbar';
end
```

Beispiel 9.1

In diesem Beispiel schließen sich alle drei Fälle gegenseitig aus. Im Allgemeinfall wird diejenige Anweisung abgearbeitet, die der ersten wahren Aussage in der ***if-else***-Kette folgt.

Es ist wichtig zu verstehen, wie relationale Operatoren und die ***if***-Anweisungen mit Matrizen wirken. Falls Sie 2 Variablen auf Gleichheit überprüfen wollen, verwenden Sie vielleicht

```
if A==B, ...
```

Dies ist zwar syntaktisch richtiger MATLAB-Code und arbeitet auch wie erwartet, wenn A und B Skalare sind. Aber wenn A und B Matrizen sind, kann man damit nicht testen, ob A und B übereinstimmen! Es wird lediglich festgestellt, in welchen Positionen ihre Elemente übereinstimmen. Das Ergebnis von A==B ist eine Matrix aus Nullen und Einsen. Die Einsen stehen in denjenigen Positionen, in denen die Elemente von A und B übereinstimmen.

Übrigens: Falls A und B nicht das selbe Format haben, wird der Ausdruck A == B sogar als Fehler betrachtet!

Der richtige Weg um 2 Felder auf Gleichheit zu überprüfen ist die Verwendung der Funktion ***isequal*** :

```
if isequal(A,B), ...
```

Sie liefert einen skalaren Wert, 1 oder 0, je nachdem, ob A und B komplett gleich sind oder nicht.

Es folgt ein weiteres Beispiel, um diese Angelegenheit zu verdeutlichen. Wenn A und B Skalare sind, wird der folgende Code niemals eine Fehlermeldung produzieren. Aber für die meisten Matrizenpaare wird keine der Bedingungen A > B, A < B oder A == B jemals für alle Elemente zugleich wahr sein, deshalb wird praktisch immer eine Fehlermeldung ausgegeben.

```
if A > B,       'A > B'
elseif A < B,   'A < B'
elseif A == B,  'A = B'
else            'Nanu?'
end
```

Beispiel 9.2

Einige MATLAB-Funktionen sind hilfreich um, das Ergebnis der Matrixvergleiche auf Skalare zu reduzieren um sie in ***if***–Anweisungen verwenden zu können. Das sind

Isequal (Gleichheit)

isempty (leere Matrix)

sowie ***all*** und ***any***.

switch und *case*

Der ***switch*** Befehl führt Befehlsgruppen abhängig von den Werten einer Variablen oder eines Ausdrucks aus. Die Schlüsselwörter ***case*** und ***otherwise*** begrenzen diese Blöcke. Nur die Anweisungen nach dem ersten passenden ***case*** werden ausgeführt. Ein ***switch*** muß immer mit einem ***end*** abgeschlossen werden.

Die Bedingung zur Konstruktion des magischen Quadrats kann mit ***switch*** folgendermaßen beschrieben werden:

```
switch (rem(n,4) == 0) + (rem(n,2) == 0)
  case 0
       'ungerade'
  case 1
       'nur durch 2 teilbar'
  case 2
       'durch 4 teilbar'
  otherwise
       'das gibt''s nicht'
end
```

Beispiel 9.3

Hinweis für C-Programmierer

Anders als in der Programmiersprache C benötigt die *switch* - Anweisung kein *break*. Der *switch*-Block wird nach der Abarbeitung der Befehle und vor der nächsten *case* -Anweisung verlassen.

---

for-Schleifen

Die ***for***-Schleife durchläuft eine Gruppe von Anweisungen, bis die vorgegebene Anzahl an Durchläufen erreicht ist. ***end*** beendet den Anweisungsblock.

```
for n=3:32
  r(n) = rank(magic(n));
end
r
```

Beispiel 9.4

Der Strichpunkt hinter dem Befehl in der Schleife verhindert die Ausgabe und das r nach der Schleife gibt das endgültige Ergebnis aus.

Es ist sinnvoll, Schleifen aus Gründen der Übersichtlichkeit einzurücken, besonders wenn sie geschachtelt sind. Ein Kästchen H mit Lottozahlen entsteht z.B. aus 2 geschachtelten Schleifen

```
for i=1:7
  for j=1:7
     H(i,j) = (i-1)*7+j;
  end
end
```

Beispiel 9.5

So sollte man das aber keinesfalls programmieren!

Erstens können Sie nach Ablauf dieser Schleifen die komplexe Einheit ***i*** bzw. ***j*** nicht mehr verwenden, weil sie durch die Laufvariablen der beiden Schleifen überschrieben wurden. Jetzt gilt eben nicht mehr $i = j = \sqrt{-1}$, sondern $i = j = 7$.

Zweitens sind Schleifen in der Regel durch die viel schnelleren Vektor- und Matrix-Operationen ersetzbar. Alternative Anweisungen zur Erzeugung des Lotto-Kästchens H wären z.B.

```
H=[1:7; 8:14; 15:21; 22:28; 29:35; 36:42; 43:49]
```

oder eine einfache Schleife

```
H=[]; for m=1:7, H=[H; (m-1)*7 + (1:7)]; end;
```

oder auch

```
n=1:7; for m=n, H(m,:)=n+(m-1)*7; end;
```

while-Schleifen

Die ***while***-Schleife wiederholt einen Anweisungsblock beliebig oft, solange bis eine logische Abbruchbestimmung erfüllt ist. ***end*** legt das Ende des Anweisungsblocks fest.

Es folgt ein komplettes Programm, das ***while***, ***if***, ***else*** und ***end*** veranschaulicht. Es benutzt die Intervallschachtelungmethode, um eine Nullstelle des Polynoms $x^3 - 2x - 5$ zu finden.

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
  x = (a+b)/2; fx = x^3-2*x-5;
  if sign(fx) == sign(fa),
     a = x; fa = fx;
  else
     b = x; fb = fx;
  end
end
x
```

ergibt

```
x =
   2.09455148154233
```

Beispiel 9.6

Beachten Sie auch bei der Laufbedingung von ***while***-Schleifen die elementweise (!) Wirkung von Vergleichsoperationen zwischen Feldern. Sie führt leicht zu Fehlern, wie schon zuvor bei der ***if***-Anweisung diskutiert wurde.

break

Die ***break***-Anweisung erlaubt ein vorzeitiges Verlassen einer ***for***- oder ***while*** - Schleife. In geschachtelten Schleifen, verläßt ***break*** nur die innerste Schleife.

Zur Verdeutlichung nochmals unser obiges Beispiel. Warum ist es sinnvoll, hier ***break*** zu benutzen?

```
a = 0; fa = -Inf; b = 3; fb = Inf;
while b-a > eps*b
   x = (a+b)/2; fx = x^3-2*x-5;
   if fx == 0, break
   elseif sign(fx) == sign(fa)
        a = x; fa = fx;
   else
        b = x; fb = fx;
   end
end
x
```

Beispiel 9.7

Übungsaufgaben

1.► Man stelle fest, ob folgende Vektoren linear abhängig oder linear unabhängig sind!

$$\mathbf{a} = \begin{bmatrix} 5 \\ 2 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 10 \\ 4 \end{bmatrix} \qquad \mathbf{c} = \begin{bmatrix} -15 \\ -6 \end{bmatrix}$$

Lösung 9.1

2. Multiplizieren Sie den Vektor $\mathbf{u}_0 = [1,0]^T$ mit der Matrix $\mathbf{A} = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$. Das Ergebnis sei $\mathbf{u}_1$. Berechnen Sie dann $\mathbf{u}_2 = \mathbf{A}\mathbf{u}_1$ und schließlich $\mathbf{u}_3 = \mathbf{A}\mathbf{u}_2$. Welche Eigenschaft haben die vier Vektoren $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$?
Rechnen Sie mit MATLAB nach der Idee der vorhergehenden Aufgabe bis zum Vektor $\mathbf{u}_7$. Machen Sie das gleiche, aber beginnen Sie die Iteration mit dem Startvektor $\mathbf{v}_0 = [0,1]^T$. Wiederholen Sie schließlich den Vorgang mit dem Startvektor $\mathbf{w}_0 = [0.5, 0.5]^T$. Was beobachten Sie bei den Vektoren $\mathbf{u}_7, \mathbf{v}_7$ und $\mathbf{w}_7$? Plotten Sie die Ergebnisse.

Lösung 9.2

3. Wir betrachten n x n-Matrizen $\mathbf{A_n}$ mit 2 auf der Diagonalen und –1 auf der oberen und unteren Nebendiagonale. Die übrigen Matrixeinträge von $\mathbf{A_n}$ sind alle 0. Somit gilt:

$$\mathbf{A_1} = [2]$$

$$\mathbf{A_2} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

$$\mathbf{A_3} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Diese Matrizen lassen sich in MATLAB einfach erzeugen. Zum Beispiel erzeugt man $\mathbf{A_4}$ durch folgende Befehle:

```
>> n=4;
>> A=toeplitz([2,-1,zeros(1,n-2)]);
```

a) Berechnen Sie für n = 2,3,4,5,6 die Determinante von $\mathbf{A_n}$. Welche Berechnungsformel vermuten sie für $\det(\mathbf{A_n})$?

b) Zeigen Sie, daß die Rekursionsgleichung $\det(\mathbf{A_{n+1}}) = 2\det(\mathbf{A_n}) - \det(\mathbf{A_{n-1}})$ für $n \geq 3$ gilt.

Lösung 9.3

4. Sei $\mathbf{A} = \begin{bmatrix} -6 & 28 & 21 \\ 4 & -15 & -12 \\ -8 & a & 25 \end{bmatrix}$. Berechnen Sie für jeden Wert a aus der Menge {32, 31.9, 31.8, 32.1, 32.2} das charakteristische Polynom und die Eigenwerte von $\mathbf{A}$. Zeichnen Sie in jedem Fall den Graph des charakteristischen Polynoms p(t) = $\det(\mathbf{A} - t\mathbf{I})$ für $0 \leq t \leq 3$. Zeichnen Sie alle Graphen in ein Koordinatensystem. Beschreiben Sie wie die Graphen die Veränderungen der Eigenwerte zeigen, wenn a verändert wird.

Lösung 9.4

5. Bestimmte dynamische Systeme können durch Untersuchung der Potenzen von Matrizen studiert werden. Bestimmen Sie was mit $\mathbf{A}^k$ und $\mathbf{B}^k$ bei steigendem k (z.B. k = 2,...,16) passiert. Versuchen Sie herauszufinden was bei den Matrizen $\mathbf{A}$ und $\mathbf{B}$ besonders ist. Untersuchen Sie große Potenzen anderer Matrizen dieses Typs und stellen Sie eine Vermutung für solche Matrizen auf.

$$\mathbf{A} = \begin{bmatrix} .4 & .2 & .3 \\ .3 & .6 & .3 \\ .3 & .2 & .4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & .2 & .3 \\ .1 & .6 & .3 \\ .9 & .2 & .4 \end{bmatrix}$$

Lösung 9.5

6. Lösen Sie das folgende Gleichungssystem mit dem Gauß-Jacobi-Verfahren und mit dem Gauß-Seidel-Verfahren. Vergleichen Sie die Ergebnisse nach 10 Schritten.

$$\begin{aligned} 5x_1 - x_2 - 2x_3 &= 2 \\ x_1 - 6x_2 + 3x_3 &= -2 \\ -2x_1 + x_2 + 4x_3 &= 3 \end{aligned}$$

Lösung 9.6

7. Lösen Sie das folgende Gleichungssystem mit dem Gauß-Jacobi-Verfahren und mit dem Gauß-Seidel-Verfahren mit $\mathbf{x}^{(0)} = 0$.

$$\begin{aligned} 50x_1 - x_2 &= 149 \\ x_1 - 100x_2 + 2x_3 &= -101 \\ 2x_2 + 50x_3 &= -98 \end{aligned}$$

Vergleichen Sie die Anzahl der notwendigen Iterationen bis die Näherungslösung nur noch um maximal 0.001 abweicht.

Lösung 9.7

10. Andere Datenstrukturen

Dieses Kapitel führt sie in erweiterte Datenstrukturen von MATLAB ein. Es beinhaltet:

Mehrdimensionale Felder

Zellenfelder

Zeichen und Text

Strukturen

Mehrdimensionale Felder

Als mehrdimensionale Felder werden in MATLAB Felder mit mehr als 2 Indizes bezeichnet. Sie können durch die Funktionen ***zeros***, ***ones***, ***rand*** oder ***randn*** erzeugt werden.

Beispiel:

```
R = randn(3,4,5);
```

Beispiel 10.1

erzeugt ein Feld im Format 3x4x5 mit 60 normalverteilten Zufallszahlen.

Solch ein dreidimensionales Feld kann z.B. Daten aus der realen dreidimensionalen Welt repräsentieren, wie z.B. die Temperatur in einem Zimmer. Es kann natürlich auch eine Folge von Matrizen $A^{(k)}$ oder eine zeitabhängige Matrix A(k) darstellen. In den letzten beiden Beispielen kann das (i,j)-te Element der k-ten Matrix durch A(i,j,k) erreicht werden.

MATLABs und Dürers Version des magischen Quadrates der Ordnung 4 unterscheiden sich durch eine Vertauschung von 2 Spalten. Unterschiedliche magische Quadrate können durch weitere Vertauschung von Spalten erzeugt werden. Der Ausdruck

```
p = perms(1:4);
```

Beispiel 10.2

erzeugt alle 4! = 24 Permutationen der Zahlen 1:4. Die k-te Permutation ist der Zeilenvektor p(k,:). Und so erzeugt

```
A = magic(4);
M = zeros(4,4,24);
for k = 1:24
   M(:,:,k) = A(:,p(k,:));
end
```

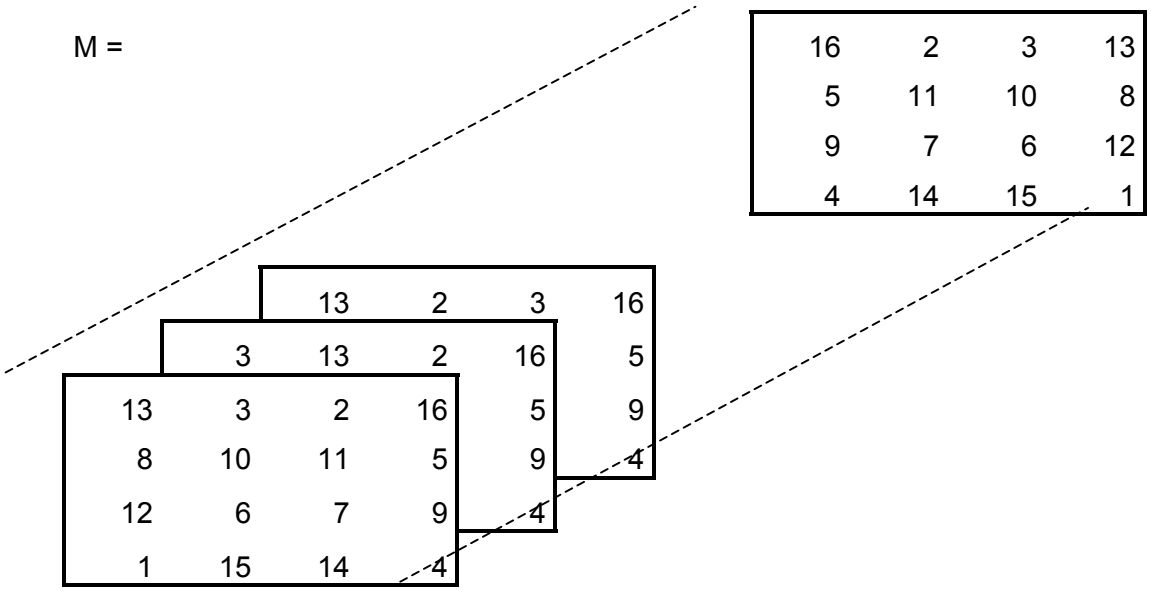
Beispiel 10.3

eine Folge von 24 magischen Quadraten in einem dreidimensionalen Feld M. Das Format von M ist

```
size(M)
```

```
ans =
     4     4    24
```

Beispiel 10.4



Es zeigt sich, dass die 22. Matrix in dieser Reihe die Dürer-Matrix ist:

```
M(:,:,22)
ans =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Der Ausdruck ***sum*(*M,d*)** berechnet Summen über den Index Nr. d im Feld M. Also ist z.B.

```
sum(M,1)
```

ein dreidimensionales Feld im Format 1x4x24, dessen Elemente als Summen über den 1. Index von M, entstehen. Im Prinzip ist das zwar eine Matrix, ausgegeben wird aber 24 mal der selbe Zeilenvektor. Warum?

```
ans(:,:,1) =
    34    34    34    34
ans(:,:,2) =
    34    34    34    34
ans(:,:,3) =
    34    34    34    34
.  .  .  .  .  .  .  .  .
```

```
.   .   .   .   .   .   .   .   .
.   .   .   .   .   .   .   .   .
ans(:,:,23) =
    34    34    34    34
ans(:,:,24) =
    34    34    34    34
```

Beispiel 10.5

Formal handelt es sich bei ***sum*(M,1)** immer noch um ein dreidimensionales Feld, da 3 unabhängige Indizes vorliegen. MATLAB gibt drei- und mehrdimensionale Felder als Matrix-Folgen aus, deren einzelne Matrizen sich durch Variation der ersten beiden Indizes ergeben. In unserem Fall nimmt der 1. Index nur einen einzigen Wert an, also erscheint unser dreidimensionales Feld als Folge von Zeilenvektoren. Anders bei Summation über den 2. Index.

```
sum(M,2)
```

ergibt ein Feld im Format 4x1x24. Mit der gleichen Argumentation wird diesmal eine Folge von 24 Spaltenvektoren ausgegeben.

```
ans(:,:,1) =
    34
    34
    34
    34
ans(:,:,2) =
    34
    34
    34
    34
ans(:,:,3) =
    34
    34
    34
    34
.   .   .   .
.   .   .   .
.   .   .   .
ans(:,:,23) =
    34
    34
    34
    34
```

```
ans(:,:,24) =
     34
     34
     34
     34
```

Beispiel 10.6

Schliesslich liefert

```
S=sum(M,3)
```

ein Feld S im Format 4x4x1, die Summe aller 24 Matrizen. Diesmal hat die Matrix-Folge als einziges Element eine 4x4-Matrix S, die ausgegeben wird wie folgt.

```
S =
   204   204   204   204
   204   204   204   204
   204   204   204   204
   204   204   204   204
```

Beispiel 10.7

Zellenfelder

Zellenfelder sind in MATLAB mehrdimensionale Felder, deren Elemente Kopien von anderen Feldern sind, die auch unterschiedliche Formate haben können. Ein Zellenfeld aus leeren Matrizen kann durch den ***cell***-Befehl erzeugt werden. Häufiger werden Zellenfelder jedoch durch eine Ansammlung unterschiedlicher Daten in geschweiften Klammern { } erzeugt. Die geschweiften Klammern werden außerdem in Verbindung mit Indizes benutzt, um auf den Inhalt unterschiedlicher Zellen zuzugreifen.

Zum Beispiel erzeugt

```
C = {A sum(A) prod(prod(A))}
```

ein 1x3-Zellenfeld. Die drei unterschiedlich großen Zellen enthalten das magische Quadrat A, den Zeilenvektor seiner Spaltensummen und das Produkt aller Elemente von A. Wenn C ausgegeben wird, erscheint

```
C =
  [4x4 double]    [1x4 double]    [20922789888000]
```

Diese Darstellung ist ungewöhnlich, weil nur Formatangaben der Zellen erfolgen und lediglich die letzte Zelle in Form eines Zahlenwertes erscheint. Das Feld wird deshalb so komprimiert in einer Zeile dargestellt, weil die Zellen sowieso nicht in ein gemeinsames Schema passen. Nur der dritte Wert, 16!, passt explizit in eine Zeile.

An dieser Stelle noch mal zwei Punkte, die man nicht vergessen sollte. – Erstens: Um den Inhalt einer der Zellen zu kennzeichnen, benutzt man Indizes in geschweiften Klammern. Zum Beispiel liefert C{1}, das magische Quadrat A und C{3} den Wert 16!. – Zweitens: Zellenfelder enthalten Kopien von Feldern und keine Zeiger auf diese Felder. Wenn im Feld A etwas verändert wird, so ändert sich nichts in C.

Dreidimensionale Felder können zum Speichern von Folgen von Matrizen gleichen Formats verwendet werden. Zellenfelder können auch zum Speichern von Matrix-Folgen mit unterschiedlichen Formats Verwendung finden.

Zum Beispiel erzeugt folgender MATLAB-Code eine Folge von Magischen Quadraten in ansteigendem Format. Sie werden in einem Zellenfeld M gespeichert.

```
M = cell(8,1);
for n = 1:8
   M{n} = magic(n);
end
M
```

Weil die Zellen von M in einer Spalte M angeordnet sind, wird ausgegeben

```
M =
    [         1]
    [2x2 double]
    [3x3 double]
    [4x4 double]
    [5x5 double]
    [6x6 double]
    [7x7 double]
    [8x8 double]
```

Beispiel 10.8

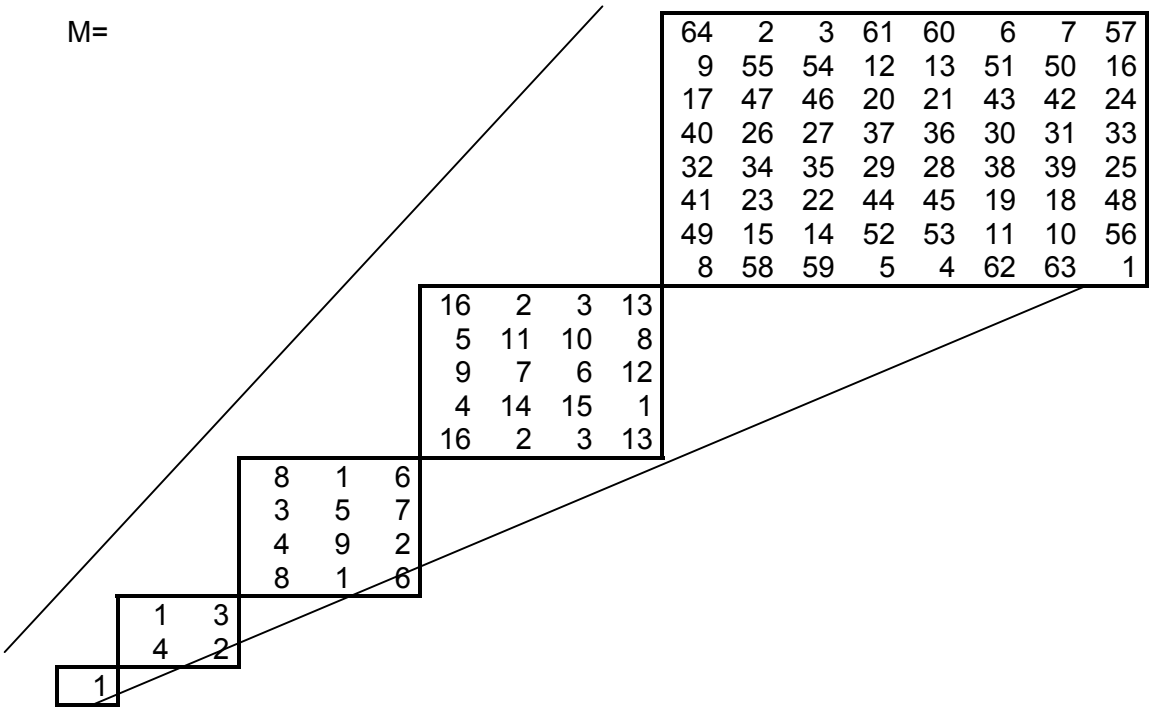
Die Vereinbarung ***cell*(M,N)** erlaubt natürlich beliebige Anordnung der Zellen.

Unsere Dürer Matrix A entsteht diesmal als

```
M{4}
ans =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Beispiel 10.9

Die folgende Skizze soll das Zellenfeld M veranschaulichen.



Zeichen und Text

Um Text in MATLAB zu kennzeichnen, werden Hochkommas verwendet. Beispiel:

```
s = 'Hallo'
s =
  Hallo
```

Beispiel 10.10

Das Ergebnis ist nicht die gleiche Art von Matrix wie wir sie bisher kennengelernt und verwendet haben. s ist ein Zeichenfeld im Format 1x5. Intern werden die Zeichen als ganze Zahlen gespeichert, nicht im Gleitpunktformat. Der Ausdruck

```
a = double(s)
```

wandelt das Zeichenfeld in eine Matrix aus Gleitpunktzahlen um. Dabei wird der ASCII-Code als Grundlage verwendet. Das Ergebnis ist

```
a =
    72    97   108   108   111
```

Beispiel 10.11

Der Ausdruck

```
s = char(a)
s =
  Hallo
```

Beispiel 10.12

macht die Umwandlung rückgängig.

Das Umwandeln von Zahlen in Zeichen bietet eine gute Möglichkeit, die unterschiedlichen zur Verfügung stehenden Zeichensätze auszuprobieren, die Ihr Computer zur Verfügung hat. Die normal darstellbaren Zeichen im ASCII-Zeichensatz werden durch die Nummern von 32:127 repräsentiert. (Nummern kleiner als 32 stellen nichtdarstellbare Kontrollzeichen dar.) Diese Nummern werden in einem 6x16-Feld angeordnet durch

```
F = reshape(32:127,16,6)';
```

Die darstellbaren Zeichen des erweiterteten ASCII-Zeichensatzes werden durch F+128 repräsentiert. Werden diese Nummern als Zeichen interpretiert, so kommt es auf den aktuell verwendeten Zeichensatz an, welche Zeichen Sie darstellen.

Geben sie die Zeilen

```
char(F)
char(F+128)
```

ein und verändern Sie die Einstellung des Zeichensatzes in Ihrem MATLAB-Kommando-Fenster durch die Menüpunkte **Tools / Font...**. Versuchen Sie die Schriften Symbol und Wingdings, wenn sie installiert sind. Dann erhalten Sie folgende Ausgabezeilen

```
ans =
   !"#$%&'()*+,-./
  0123456789:;<=>?
  @ABCDEFGHIJKLMNO
  PQRSTUVWXYZ[\]^_
  `abcdefghijklmno
  pqrstuvwxyz{|}~□
ans =
   ¡¢£¤¥¦§¨©ª«¬-®¯
  °±²³´µ¶·¸¹º»¼½¾¿
  ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
  ÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
  àáâãäåæçèéêëìíîï
  ðñòóôõö÷øùúûüýþÿ
```

Beispiel 10.13

Textvariablen kann man verbinden. Durch aneinanderreihen in eckigen Klammern werden längere Zeichenketten erzeugt.

```
h = [s, ' Leute!']
```

führt zu

```
h =
  Hallo Leute!
```

Beispiel 10.14

Der Ausdruck

```
v = [s; 'Leute']
```

führt zu

```
v =
  Hallo
  Leute
```

Beispiel 10.15

Man beachte: Beim ersten Ausdruck ist ein Leerzeichen vor dem L nötig, um die beiden Wörter zu trennen, und beim zweiten Ausdruck müssen beide Zeichenketten gleich lang sein, damit die Speicherung in einer Matrix erfolgen kann! - Die resultierenden Felder jedesmal Zeichenfelder. h ist ein Zeilenvektor im 1x11-Format und v ist eine 2x5-Matrix.

Um Textinhalte mit unterschiedlicher Längen zu manipulieren hat man zwei Möglichkeiten: Ein aufgefülltes Zeichenfeld oder Ein Zellenfeld aus Zeichenketten.

Zu Auffüllen eines Zeichenfeldes benutzt man ***char***. Die Funktion ***char*** akzeptiert jegliche Anzahl von Zeilen und fügt automatisch Leerzeichen an jede Zeile um sie alle gleich lang zu machen.

```
S = char('1','Byte','speichert','ein','Textsymbol.')
```

erzeugt zum Beispiel ein aufgefülltes 5x9-Feld aus gleich langen Zeilen.

```
S =
  1
  Byte
  speichert
  ein
  Textsymbol.
```

Beispiel 10.16

In den ersten vier Zeilen sind genügend viele Leerzeichen automatisch von der Funktion angehängt worden, um alle Zeilen gleich lang zu machen.

Alternativ kann man den Text in einem Zellenfeld abspeichern.

```
C = {'1';'Byte';'speichert';'ein';'Textsymbol.'}
```

erzeugt ein 5x1 Zellenfeld.

```
C =
  '1'
  'Byte'
  'speichert'
  'ein'
  'Textsymbol.'
```

Beispiel 10.17

Man kann ein Zeichenfeld in ein Zellenfeld umwandeln durch

```
C = cellstr(S)
C =
  '1'
  'Byte'
  'speichert'
  'ein'
  'Textsymbol.'
```

Beispiel 10.18

und umgekehrt ein Zellenfeld in ein aufgefülltes Zeichenfeld

```
S = char(C)
S =
  1
  Byte
  speichert
  ein
  Textsymbol.
```

Beispiel 10.19

Strukturen

Strukturen sind Felder mit Komponenten unterschiedlichen Typs. Diese Komponenten werden nicht über Indizes, sondern über Komponentennamen angesprochen.

Beispiel:

```
S.name  = 'Robert Wittl';
S.punkte = 83;
S.note = '2+';
```

erzeugt eine skalare Struktur mit drei Komponenten. Sie wird ausgegeben durch den Aufruf

```
S
S =
    name: 'Robert Wittl'
  punkte: 83
    note: '2+'
```

Beispiel 10.20

Wie alle anderen Daten in MATLAB, werden auch Strukturen immer von vorneherein als Feldelemente betrachtet, denen man jederzeit weitere Elemente hinzufügen kann. Jedes solche Feldelement ist eine komplette Struktur mit den vorgegebenen Komponenten. Eine zweite Struktur kann man S elementenweise hinzufügen. Sie trägt den Index 2 und lautet z.B.

```
S(2).name = 'Ulf Wilke';
```

```
S(2).punkte = 91;
S(2).note = '1-';
```

oder auch durch einen einzigen Ausdruck anhängen, wie die dritte Struktur

```
S(3) = struct('name','Hans Wagner','punkte',70,'note','3')
```

Die Ausgabe eines Strukturen-Feldes S erfolgt nur summarisch, da solch ein Feld zu komplex aufgebaut ist:

```
S
S =
1x3 struct array with fields:
    name
    punkte
    note
```

Beispiel 10.21

Es gibt mehrere Wege, Komponenten einer Struktur in andere Feldtypen zu überführen. Sie basieren alle auf der Schreibweise von durch Kommas getrennten Listen. Wenn Sie eintippen

```
S.punkte
ans =
    83
ans =
    91
ans =
    70
```

Beispiel 10.22

entspricht dies im einzelnen der Liste

```
S(1).punkte, S(2).punkte, S(3).punkte
ans =
    83
ans =
    91
ans =
    70
```

Beispiel 10.23

Ohne Zusätze macht diese Liste wenig Sinn, denn es werden nur die drei *punkte*-Komponenten nacheinander der Variablen ***ans*** zugewiesen und ausgegeben.

Wenn Sie aber eine Komponente des Strukturen-Feldes in eckige Klammern setzen,

```
[S.punkte]
```

```
ans =
    83    91    70
```

entspricht dies dem Zeilenvektor

```
[S(1).punkte, S(2).punkte, S(3).punkte]
```

mit den Inhalten der 3 Elemente.

```
ans =
    83    91    70
```

Beispiel 10.24

Das Gleiche gilt für

```
S.name
ans =
    Robert Wittl
ans =
    Ulf Wilke
ans =
    Hans Wagner
```

Beispiel 10.25

nacheinander werden die Inhalte ***ans*** zugewiesen. Wird der Ausdruck aber in geschweiften Klammern geschrieben

```
{S.name}
ans =
    'Robert Wittl'    'Ulf Wilke'    'Hans Wagner'
```

Beispiel 10.26

so entsteht ein Zellenfeld ***ans*** im Format 1x3, welches die drei Namen beinhaltet.

Schließlich ruft

```
char(S.name)
```

die ***char***-Funktion mit drei Argumenten auf, um so ein aufgefülltes Zeichenfeld ***ans*** zu erzeugen

```
ans =
   Robert Wittl
   Ulf Wilke
   Hans Wagner
```

Beispiel 10.27

11. Skripte und Funktionen

MATLAB ist sowohl eine Programmiersprache als auch eine interaktive Rechenumgebung. Dateien, die MATLAB-Quellcode enthalten, heißen M-Files. Sie können M-Files mit Hilfe eines beliebigen Texteditors erstellen und diese dann wie jede andere MATLAB-Funktion oder wie einen MATLAB-Befehl aufrufen.

Es gibt zwei Arten von M-Files:

- **Skripte** akzeptieren keine Eingabeargumente und geben keine Ergebnisse zurück. Sie arbeiten mit dem Daten im Workspace (und entsprechen den MATLAB-Befehlen).
- **Funktionen** akzeptieren Eingabeargumente und liefern Ergebnisse zurück. Interne Variablen sind lokal zur Funktion, d.h. außerhalb nicht definiert.

Falls Sie ein Neuling in der MATLAB - Programmierung sind, können Sie M-Files einfach im aktuellen Verzeichnis erstellen. (Meistens ist **C:\Matlab\work** voreingestellt.) Wenn Sie häufig eigene M-Files entwickeln wollen, sollten Sie diese in mehreren Verzeichnissen und in persönlichen Toolboxen ablegen, deren Namen Sie in MATLAB zum Suchpfad hinzufügen können, z.B. mit ***addpath***.

Falls Funktionsnamen im Pfad mehrfach vorhanden sind, führt MATLAB diejenige Funktion aus, die es zuerst im Suchpfad findet. (In der Regel steht eine selbstgeschriebene Funktion vor den MATLAB-Funktionen. Dafür sorgt ***addpath***, indem es eigene Ordner dem MATLAB-Pfad voranstellt.)

Um den Inhalt eines M-Files anzuzeigen, etwa **myfunction.m**, benutzen Sie

```
type myfunction
```

Skripte

Wenn Sie ein Skript aufrufen, führt MATLAB einfach die in der Datei gefundenen Befehle aus. Skripte können mit bereits vorhandenen Daten im Workspace rechnen. Obwohl Skripte keine Rückgabewerte liefern, verbleiben alle innerhalb des Skripts erstellten Variablen im Workspace, um für weitere Berechnungen zur Verfügung zu stehen. Skripte können auch graphische Ausgaben erzeugen, etwa mit Hilfe der Funktion ***plot***.

Als Beispiel, erstellen Sie eine Datei mit dem Namen **magicrank.m**, die die folgenden MATLAB Anweisungen enthält.

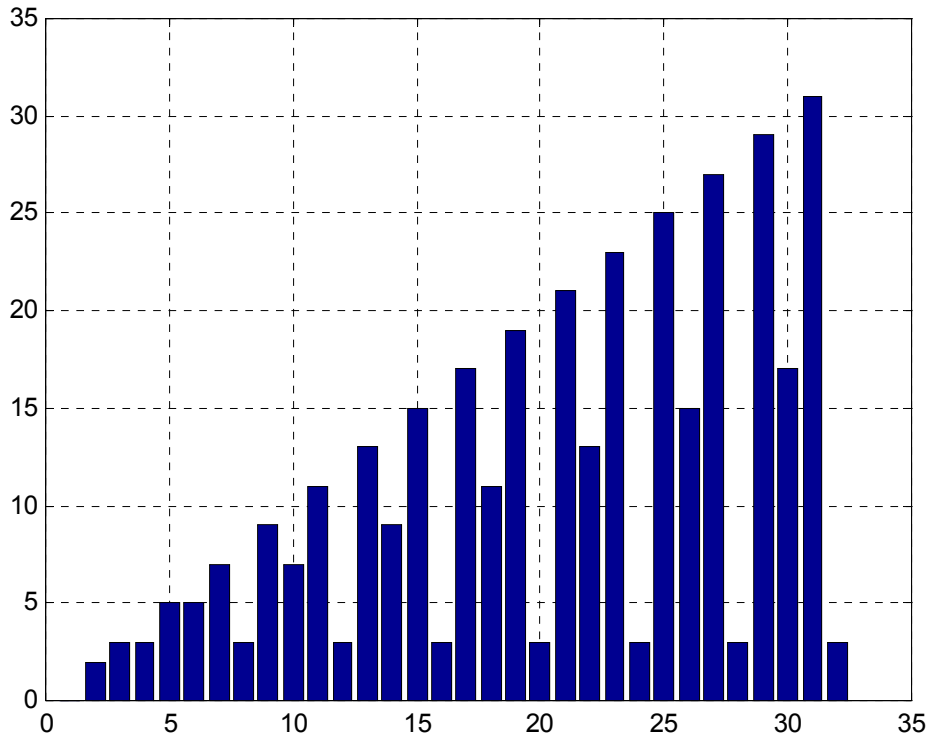
```
r = zeros(1,32);
for n = 3:32
   r(n) = rank(magic(n));
end
r
bar(r)
```

Beispiel 11.1

Im Befehlsfenster vom MATLAB können Sie nun den „neuen“ Befehl

```
magicrank
```

eingeben und veranlassen damit MATLAB die Befehle auszuführen, die in diesem Skript gespeichert sind, d.h. den Rang der ersten 30 magischen Quadrate zu berechnen und die Ergebnisse als Histogramm darzustellen. Nachdem die Abarbeitung der Datei erfolgt ist, bleiben die Variablen n und r im Workspace erhalten.



Funktionen

Funktionen sind M-Files, die Eingabeargumente akzeptieren und Ergebnisse zurückgeben können. Der Name eines M-Files sollte mit dem Namen der darin beschriebenen Funktion übereinstimmen. Funktionen arbeiten mit eigenen lokalen Variablen in ihrem separaten Workspace, getrennt von dem Workspace, den Sie von der MATLAB Kommandozeile ansprechen können.

Als Beispiel betrachten wir die MATLAB-Funktion ***rank***, deren Quellcode Sie ansehen können. Das zugehörige M-File **rank.m** ist im Verzeichnis **C:\Matlab\toolbox/matlab/matfun** oder einem ähnlich aufgebauten Pfad gespeichert.

Sie können sich die Datei mit

```
type rank
```

ansehen und erhalten die Ausgabe

```
function r = rank(A,tol)
%RANK   Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.7 $  $Date: 1997/11/21 23:38:49 $

s = svd(A);
if nargin==1
   tol = max(size(A)') * max(s) * eps;
end
r = sum(s > tol);
```

Beispiel 11.2

Die erste Zeile der Funktion im M-File beginnt mit der Schlüsselwort ***function***. Sie gibt der Funktion einen Namen und legt die Reihenfolge der Argumente fest. In unserem Fall gibt es zwei Eingabeparameter A und tol sowie einen Rückgabewert r. Die Werte der Eingabeparameter erhält die Funktion bei ihrem Aufruf vom rufenden Programm, also z.B. aus dem MATLAB-Befehlsfenster oder von einem MATLAB-Skript. Rückgabewerte liefert die Funktion nach ihrem Aufruf an das rufende Programm.

Die nächsten Zeilen bis hin zur ersten Leer- oder ausführbaren Zeile, sind Kommentare, die eine Beschreibung der Funktion ermöglichen. Diese Zeilen werden ausgegeben, wenn Sie

```
help rank
```

eingeben. (Kommentare beginnen immer mit einem %-Zeichen und reichen bis zum Zeilenende.)

Beispiel 11.3

Die erste Zeile des Hilfetexts ist die H1-Zeile, eine Kommentarzeile, welche MATLAB ausgibt, wenn Sie das ***lookfor*** Kommando oder ***help*** zu einer Toolbox benutzen. ***help rank*** dagegen zeigt alle Kommentarzeilen ab der H1-Zeile bis zur ersten Leerzeile.

Der Rest der Datei ist ausführbarer MATLAB-Quellcode, der die Arbeitsweise der Funktion festlegt. Die Variable s wird innerhalb der Funktion eingeführt, ebenso die Parameter r, A und tol. Alle 4 sind lokale Variablen in ***rank***, außerhalb vom MATLAB Workspace.

Besonders wichtig ist die Wertzuweisung an die Ausgabevariable der Funktion, in unserem Fall ist das r. Der letzte an r zugewiesene Wert wird beim Funktionsaufruf an das rufende Programm übergeben.

Das ***rank***-Beispiel zeigt auch eine Eigenschaft von MATLAB-Funktionen die normalerweise nicht in anderen Programmiersprachen zu finden ist: Bei Funktionsaufrufen ist eine variable Anzahl von Argumenten erlaubt. Die ***rank***-Funktion kann nämlich auf verschiedene Arten aufgerufen werden, z.B.

```
rank(A)           % 1 Parameter, keine Wertzuweisung
r = rank(A)       % 1 Parameter + Wertzuweisung
r = rank(A,1.e-6) % 2 Parameter + Wertzuweisung
```

Beispiel 11.4

Viele vorgefertigte M-Files arbeiten nach diesem Prinzip. Falls beim Funktionsaufruf keine Wertzuweisung an eine Variable erfolgt, wird das Ergebnis in ***ans*** gespeichert. Wenn das zweite Eingabeargument fehlt, setzt die Funktion dafür einen Standardwert ein. Automatisch sind innerhalb einer Funktion sind zwei vordefinierte Variablen mit den Namen ***nargin*** und ***nargout*** verfügbar. Diese enthalten jeweils die Anzahl der Ein- und Ausgabeparameter des aktuellen Funktionsaufrufs. Die ***rank***-Funktion benutzt nur ***nargin***, denn ***nargout*** spielt in diesem Zusammenhang keine Rolle.

Globale Variablen

Falls Sie eine Variable in mehreren Funktionen benutzen wollen, definieren Sie einfach die Variable als ***global*** in allen diesen Funktionen, ebenso in der Kommandozeile des MATLAB- Befehlsfensters, wenn Sie auf die Variable auch im Befehlsfenster Zugriff haben wollen. Die globale Deklaration muss aber unbedingt zuerst erfolgen, d.h. bevor die Variable zum ersten Mal benutzt wird.

Zur Unterscheidung von anderen Variablen benutzt man für globale Variablen gerne Großbuchstaben. Das ist zwar nicht vorgeschrieben, trägt aber dazu bei, dass man sie leicht von anderen (lokalen) Variablen unterscheiden kann.

Erstellen Sie ein M-File mit dem Namen **falling.m**, das die Höhe eines Punktes anhand der Fallzeit eines Steins ermittelt. Man benötigt dazu die Erdbeschleunigung, die als globale Variable GRAVITY geführt wird. Je nachdem, ob Sie Entfernungen in Foot oder Meter messen, hat GRAVITY den Wert 32 (Foot/sec^2) oder 9.81 (m/sec^2). Dieses Detail können Sie außerhalb der Funktion *falling* einstellen, indem Sie GRAVITY vor Aufruf von *falling* mit dem passenden Wert belegen.

```
function h = falling(t)
global GRAVITY
h = 1/2 * GRAVITY * t.^2;
```

Geben Sie im Befehlsfenster die Anweisungen ein:

```
global GRAVITY
GRAVITY = 32;
y = falling((0:.1:5)');
```

Beispiel 11.6

Die beiden ***global*** Anweisungen legen

im Befehlsfenster von MATLAB den Wert fest und machen ihn innerhalb der Funktion verfügbar. Sie können nun die Variable GRAVITIY jederzeit ändern und abfragen, ohne irgendwelche Dateien zu editieren.

Dualität zwischen Kommandos und Funktionen

MATLAB Kommandos sind Befehle wie

```
load
help
```

Viele Befehle arbeiten auch mit Operanden, etwa

```
load August17th.dat
help magic
type rank
```

Alternativ kann man die Operanden auch als Parameter angeben, dann aber in Klammern, wie bei Funktionen üblich. Da es sich um Text handelt, sind die Parameter Zeichenketten und man kann anstelle der Befehle Funktionsaufrufe formulieren

```
load('August17th.dat')
help('magic')
type('rank')
```

Beispiel 11.7

Beide Schreibweisen sind gleichwertig. Grundsätzlich kann jeder Befehl ebenso gut in der Funktionsform geschrieben werden.

Der Vorteil der Funktionsschreibweise zeigt sich bei Zeichenketten, die aus mehreren Bestandteilen zusammengesetzt sind. Außerdem kann man bei der funktionsschreibweise auch Variablennamen für Zeichenketten verwenden.

Das folgende Beispiel verarbeitet mehrere Dateien, **August1.dat**, **August2.dat**, usw. Es verwendet die Funktion ***int2str***, diese konvertiert eine ganze Zahl in eine Zeichenkette. Aus den 3 Textstücken in eckigen Klammern wird ein Dateiname zusammengesetzt.

```
sonne = [];
for d=1:31
   s = [ 'August' int2str(d) '.dat' ];
   % Datei s laden und Inhalt an Vektor sonne anhaengen
   sonne = [ sonne load(s)];
end
sonne, sonne_pro_tag = mean(sonne)
```

Im Beispiel wollen wir davon ausgehen, dass jede unserer Dateien als einzige Zahl die Anzahl Sonnenstunden an diesem Tag des Ferienmonats August enthält. Ausgegeben wird die mittlere Sonnenscheindauer.

Beispiel 11.8

Die *eval*-Funktion

Die ***eval***-Funktion arbeitet mit Textvariablen deren Inhalte als arithmetische Ausdrücke interpretiert werden. Der Befehl

```
x=3; s='7+x^2'; eval(s)
```

Beispiel 11.9

benutzt den MATLAB-Interpreter, um den in der Zeichenkette s enthaltenen Ausdruck auszuwerten bzw. den in der Zeichenkette s enthaltenen Befehl zu erkennen und auszuführen.

Der Einlesevorgang im vorletzten Beispiel könnte auch mit folgendem Code gelöst werden, obwohl dieser nicht zweckmäßig ist, weil er für ***eval*** den kompletten Interpreter benötigt, nicht nur einen Funktionsaufruf von ***load***. Außerdem steht das Resultat nun in einer Variablen, deren Name vom eingegebenen Wert für d abhängt, wie das abschließende ***who*** zeigt.

```
d = input('Ein Tag im August: ');
s = [ load 'August' int2str(d) '.dat' ];
eval(s);
who
```

Nebenbei haben Sie auch die Arbeitsweise der MATLAB-Funktion ***input*** kennengelernt.

Beispiel 11.10

Vektorisierung

Um höchstmögliche Arbeitsgeschwindigkeit zu erreichen, sollte man in MATLAB Algorithmen „vektorisieren“, wo immer möglich. Dort, wo andere Programmiersprachen vielleicht ***for***- oder ***do***-Schleifen benötigen, kann MATLAB meistens Vektor- oder Matrix-Operationen ausführen. Das bringt gewöhnlich einen Geschwindigkeitsvorteil gegenüber der Schleifentechnik. Ein einfaches Beispiel:

```
t = 0.1;
for k = 1:1000
   x(k) = t;
   y(k) = log10(t);
   t = t + 0.1;
end
plot(x,y)
```

Beispiel 11.11

Die vektorisierte Version des gleichen Codes sieht folgendermaßen aus:

```
x = 0.1:.01:100;
y = log10(x);
plot(x,y)
```

Beispiel 11.12

Für komplizierteren Code ist Vektorisierung nicht immer von Vorteil, weil die Formulierung zu komplex werden kann. Wenn Geschwindigkeit wichtig ist, sollten Sie aber immer versuchen, Ihre Algorithmen zu vektorisieren.

Vorbelegung

Falls Sie ein Stück Code nicht vektorisieren können, können Sie die Schleifen schneller machen wenn Sie jeden Vektor oder jedes Feld vorbelegen, in das dann die Ergebnisse geschrieben werden. Der Code im folgenden Beispiel verwendet die Funktion ***zeros*** um den Vektor r vorzubelegen, der in der ***for***-Schleife erzeugt wird. Dies macht die Schleife wesentlich schneller.

```
r = zeros(32,1);
for n=1:32
   r(n) = rank(magic(n));
end
r
```

Beispiel 11.13

Ohne die Vorbelegung im vorherigen Beispiel, erweitert der MATLAB Interpreter den r-Vektor bei jedem Schleifendurchlauf um eins. Vorbelegung eliminiert diesen Schritt und das Ergebnis ist eine schnellere Ausführungsgeschwindigkeit.

Funktionen von Funktionen

Eine Klasse von Funktionen, genannt „Funktionenfunktionen", oder „Funktionale“ hat als Parameter wieder Funktionen. D.h. eine Funktion arbeitet mit einer anderen Funktion. Die Funktionenfunktionen beinhalten z.B.

- Nullstellenbestimmung
- Extremwertbestimmung
- Integration
- Lösung gewöhnlicher Differentialgleichungen

MATLAB stellt Funktionen als M-Files zur Verfügung. Es folgt die vereinfachte Version der Funktion ***humps*** aus dem Ordner **C:\Matlab\demos** (o.ä.):

```
function y = humps(x)
y = 1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6;
```

Beispiel 11.14

Diese Funktion wird über einer Reihe von 501 Rasterpunkten im Intervall von 0 < × < 1 ausgewertet

```
x = linspace(0,1,501);
y = humps(x);
[ [x(1:5) x(497:501)]' [y(1:5) y(497:501)]' ]
```

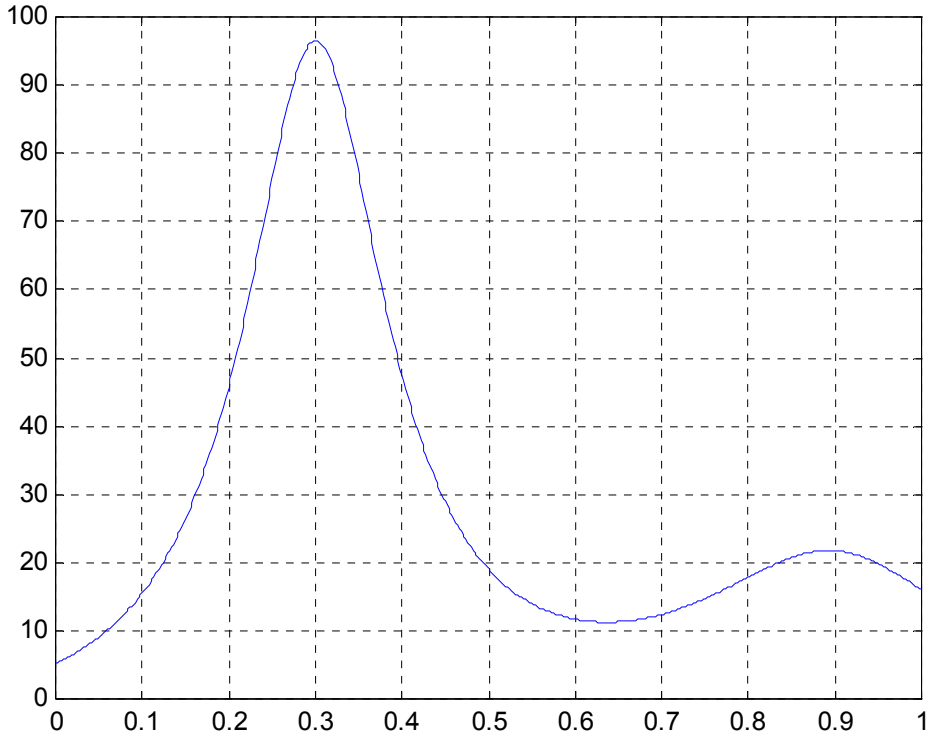
Beispiel 11.15

Dazu wurde die Funktion ***linspace*** verwendet, die über ein vorgegebenes Intervall (in unserem Fall das Intervall [0, 1]) ein äquidistantes Punktraster legt. Beachten Sie die ungerade Punktanzahl 501. Da beide Ränder des Intervalls mitgezählt werden, ergibt sich die „glatte“ Zahl 0.02 als Schrittweite. In der dritten Zeile des Beispiels werden die ersten und die letzten 5 Wertepaare tabellarisch ausgegeben.

Der Graph von *humps* wird nun gezeichnet wie folgt

```
plot(x,y)
grid on
```

Beispiel 11.16



Er zeigt, dass die Funktion ein lokales Minimum nahe 0.6 hat. Die Funktion ***fmins*** findet die Stelle p, wo das Minimum der Funktion liegt. Das erste Argument von ***fmins*** ist der Name der Funktion, die minimiert werden soll, und das zweite Argument ist ein Wert, in dessen Nähe ein Minimum gesucht werden soll. Also schreiben wir

```
p = fmins('humps',.6)
p =
    0.6370
```

Beispiel 11.17

Um den Funktionswert im Minimum zu berechnen, schreibt man

```
humps(p)
```

```
ans =
   11.2528
```

Beispiel 11.18

Leider gibt es keine Funktion zur Maximumsuche. Stattdessen kann man aber schreiben

```
y = fmins(inline('-humps(x)'),.9), top = humps(y)
y =
    0.3004
top =
   96.5014
```

Beispiel 11.19

Die ***inline***-Funktion erlaubt die direkte Eingabe einer Funktion als Zeichenkette. Das gesuchte Maximum von ***humps***(x) liegt gerade dort, wo –***humps***(x) ein Minimum hat.

„Quadratur“ ist ein anderes Wort für „Integralberechnung“ oder „Integration“. MATLABs Quadratur-Routinen heißen ***quad*** und ***quad8***. Der Befehl

```
Q = quad8('humps',0,1)
Q =
   29.8583
```

Beispiel 11.20

berechnet das Integral der Funktion ***humps*** zwischen 0 und 1.

Der Graph zeigt, dass ***humps*** in Bild nie Null wird. Falls Sie nach einer Nullstelle z suchen wollen, schreiben Sie einfach

```
z = fzero('humps',.5)
```

und Sie, werden feststellen, dass eine Nullstelle außerhalb des Bildes liegt:

```
z =
   -0.1316
```

Beispiel 11.21

Übungsaufgaben

1.* Die Binomialkoeffizienten lassen sich leicht mit Hilfe des Pascal`schen Dreiecks berechnen. Schreiben Sie dazu ein Programm (M-File), das in Abhängigkeit von n die Binomialkoeffizienten in einer n x n-Matrix $\mathbf{A}$ berechnet.

z.B. $\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$

Lösung 11.1

2. Schreiben Sie ein Funktion-File (M-File), um folgende Funktion zu berechnen:

$$f(x)=\begin{cases} x & x<0 \\ x^2 & 0\le x<2 \\ 4 & x\ge 2 \end{cases}$$

Testen Sie Ihre Funktion für die Werte x = -2, 1.5, 2 und 6. Zeichnen Sie die Funktion f über dem Intervall $[-3,3]$.

Lösung 11.2

3. Schreiben Sie jeweils ein Funktion-File (M-File), um folgende Funktionen zu berechnen:

a) $$h(x)=\begin{cases} 1 & |x|\le 0.5 \\ 0 & sonst \end{cases}$$

b) $$g(x)=\begin{cases} 0 & x<0 \\ \sin(\frac{\pi\cdot x}{2}) & 0\le x\le 1 \\ 1 & x>1 \end{cases}$$

Zeichnen Sie die Funktionen über dem Intervall [-2, 2].

Lösung 11.3

4.* Erzeugen Sie zwei 5x5-Zufallsmatrizen $\mathbf{A}$ und $\mathbf{B}$. Die Funktion f lautet: $f(\lambda)=\det(\mathbf{A}-\lambda\cdot\mathbf{B})$

a) Zeichen Sie die Funktion wenn $\lambda\in[-5,5]$.

b) Suchen Sie die Nullstellen der Funktion.

c) Suchen Sie ein Minimum der Funktion.

Lösung 11.4

12. Manipulation von Graphiken

MATLAB stellt einige Funktionen für die Erzeugung und Manipulation von Linien, Oberflächen und anderen Graphikobjekten zur Verfügung. Dieses Graphiksystem nennt sich Handle Graphics®.

Graphische Objekte

Graphische Objekte sind die einfachsten Zeichnungselemente in MATLABs Graphik-System. Diese Objekte sind in einer Baumhierarchie organisiert, die die Abhängigkeiten der einzelnen Objekte zueinander regelt. Linienobjekte benötigen z.B. Achsenobjekte als Bezugsrahmen. Achsenobjekte existieren wiederum nur innerhalb von Bildobjekten.

Es gibt 11 Arten von Handle-Graphics-Objekten:

- Das Root Objekt ist die Spitze der Hierarchie. Das Root Objekt wird von MATLAB automatisch zum Beginn einer Sitzung erzeugt.
- Abbildungsobjekte sind Fenster auf dem Root-Bildschirm.
- Uicontrol-Objekte (User Interface Control Objects) dienen zur Kontrolle und führen eine Funktion aus, sobald der Benutzer das Objekt aktiviert. Push-Buttons, Radio-Buttons und Sliders sind z.B. solche Objekte.
- Achsenobjekte definieren einen Bereich innerhalb eines Abbildungsfensters und richten Unterobjekte innerhalb dieses Bereiches aus.
- Uimenu-Objekte (User Interface Menu Objects) sind in einem Figurenfenster am oberen Rand platziert.
- Image-Objekte (Bildobjekte) sind zweidimensional. MATLAB benutzt die Elemente eines rechteckigen Feldes, um sie darzustellen, dabei werden die Feldelemente als Indizes bezüglich einer Farbtabelle verwendet.
- Line-Objekte (Linienobjekte) werden für das zweidimensionale Zeichnen verwendet.
- Patch-Objekte (Flickenobjekte) werden zum Zeichnen von ausgefüllten Polygonen mit Kanten verwendet. Ein einzelner Flicken kann mehrere Ansichten haben. Jede wird unterschiedlich eingefärbt, entweder mit fest vorgegebenen oder interpolierten Farben.
- Surface-Objekte (Oberflächenobjekte) sind dreidimensionale Repräsentanten von Matrizen. Sie werden in einem xyz-Raum gezeichnet.
- Text-Objekte sind Zeichenketten zur Bildbeschriftung.
- Lichtobjekte legen Lichtquellen fest. Sie beeinflussen alle Objekte bis hinauf zu den Achsenobjekten.

Umgang mit Objekten

Jedes einzelne Graphikobjekt bekommt von MATLAB bei der Erzeugung einen eindeutigen Bezeichner zugewiesen. Dieser wird als Handle (Griff, Henkel) bezeichnet. Einige Graphiken enthalten mehrere Kurven, sind also aus mehreren Objekten zusammengesetzt worden, von denen

jedes seinen eigenen Handle hat. Es kann sich bei aufwendigeren Graphiken als sinnvoll erweisen, Handles in Variablen zu speichern, um sich später einen leichteren Zugriff auf Graphische Objekte zu ermöglichen.

Der Handle des Rootobjekts ist immer Null. Der Handle einer Figur ist eine ganze Zahl die, wenn nichts anderes eingestellt wurde, als Fensterüberschrift erscheint. Beispiel:

```
nr = figure(3)
```

erzeugt Bildfenster 3 und zeigt die in der Fensterüberschrift an.

Alle anderen Objekthandles sind Gleitpunktzahlen, die MATLAB-interne Informationen beinhalten. Ist z.B. A das Magische Quadrat von Dürer, dann erzeugt

```
A = magic(4);
h = plot(A)
```

eine Liniendarstellung aus 4 Linien, eine für jede Spalte von A. h ist ein Vektor mit 4 Handles als Komponenten, z.B.

```
h =
   72.0002
   74.0015
   75.0005
   76.0005
```

Beispiel 12.1

Die Zahlenwerte in diesem Beispiel spielen keine Rolle, sie werden automatisch von MATLAB erzeugt und sind von Mal zu Mal verschieden. h(1) beinhaltet den Handle auf die erste Linie, h(2) auf die zweite usw..

MATLAB stellt mehrere Funktionen für den Zugriff auf häufig verwendete Handles zur Verfügung:

- ***gcf*** Get Handle to current figure
- ***gca*** Get Handle to current axis
- ***gco*** Get Handle to current object

Sie können diese Funktionen als Aufrufargumente für andere Funktionen verwenden, die Figuren- u. Achsenhandles benötigen. Die Handles der anderen Objekte erhält man, wenn sie erzeugt werden. Alle MATLAB-Funktionen, die Objekte erzeugen, liefern den Handle (oder einen Vektor mehrerer Handles) zu jedem dieser Objekte zurück. Um ein Objekt zu löschen, verwenden sie die Funktion ***delete*** (löschen). Der Funktion wird als Funktionsargument der zu löschende Handle übergeben. Zum Beispiel wird das aktuelle Achsenobjekt und all seine "Kinder" mit dem folgenden Befehl gelöscht:

```
delete(gca)
```

Funktionen zur Erzeugung von Objekten

Die meisten Graphikobjekte können mit einer Funktion erzeugt werden, die genauso heißt wie das gewünschte Objekt. Zum Beispiel erzeugt die Funktion ***text*** ein Textobjekt, die Funktion ***figure*** erzeugt ein Abbildungsobjekt usw.. Höhere Funktionen in MATLAB, wie z.B. ***plot*** und ***surf***, rufen die benötigten niederen Funktionen zum Zeichnen auf.

Niedere Funktionen erzeugen einfach eins der elf oben beschriebenen Graphikobjekte. Die Ausnahme ist das Root-Objekt, welches nur beim Start von MATLAB erzeugt werden kann.

Beispiel:

```
line([1 3 6], [8 -2 0], 'Color', 'red')
```

Objekteigenschaften

Jedes Objekt besitzt Eigenschaften die seine Darstellung zu kontrollieren. MATLAB bietet zwei Möglichkeiten die Eigenschaftswerte zu setzen. Objekteigenschaften können durch die Funktionen gesetzt werden, welche das Objekt erzeugen. Sollen sie erst nach der Erzeugung gesetzt werden, verwendet man die ***set***-Funktion. Die folgenden Zeilen zeigen wie z.B. Objekte erzeugt und deren Voreinstellungen überschrieben werden:

```
days = ['Su';'Mo';'Tu';'We';'Th';'Fr';'Sa']
temp = [21.1 22.2 19.4 23.3 23.9 21.1 20.0]
f = figure
a = axes('YLim', [16 26], 'Xtick', 1:7, 'XTickLabel', days)
h = line(1:7, temp)
days =
   Su
   Mo
   Tu
   We
   Th
   Fr
   Sa
temp =
   21.1    22.2     19.4     23.3     23.9     21.1     20
f =
   2
a =
   180
h =
   181
```

Beispiel 12.2

In diesem Beispiel ist *days* ist ein Textfeld mit den Wochentagkürzeln (in Englisch) und *temp* ist ein Vektor mit Temperaturen als Komponenten. Das Bildfenster wurde durch die Funktion ***figure*** ohne Argumente erzeugt; hierfür werden die voreingestellten Eigenschaften (Default) verwendet. Die Achsen existieren innerhalb der Figur und erhalten voreingestellte Skalierungen. Die drei Objekthandles werden in den Variablen f, a und h gespeichert, um sie später wieder zu verwenden. Beachten Sie auch die Funktion ***axes*** (im Unterschied zu ***axis***), die Bildachsen definiert.

Set und get

Um Objekteigenschaften zu verändern, werden deren Handles benötigt. Die Funktion ***set*** erlaubt es alle Eigenschaften des angesprochenen Objektes zu verändern. Dabei muß der Handle, die Eigenschaftsbezeichnung und dessen neuer Wert angegeben werden.

```
set(h,'Color',[0  .8  .8],'LineWidth',3)
```

verändert im Linienobjekt (Handle h) die Farbe auf [0 0.8 0.8] (RGB-Werte) und die Linienbreite auf 3.

Um sich eine Liste von allen prinzipiell einstellbaren Eigenschaften eines Objekts anzeigen zu lassen, ruft man ***set*** mit dem Handle als alleinigem Parameter auf.

Beispiel:

```
set(h)
Color
EraseMode: [ {normal} | background | xor | none ]
LineStyle: [ {-} | -- | : | -. | none ]
LineWidth
Marker: [ + | o | * | . | x | square | diamond | v | ^ | > | < | pen-
    tagram | hexagram | {none} ]
MarkerSize
MarkerEdgeColor: [ none | {auto} ] -or- a ColorSpec.
MarkerFaceColor: [ {none} | auto ] -or- a ColorSpec.
XData
YData
ZData
ButtonDownFcn
Children
Clipping: [ {on} | off ]
CreateFcn
DeleteFcn
BusyAction: [ {queue} | cancel ]
HandleVisibility: [ {on} | callback | off ]
HitTest: [ {on} | off ]
Interruptible: [ {on} | off ]
Parent
Selected: [ on | off ]
SelectionHighlight: [ {on} | off ]
Tag
UIContextMenu
UserData
Visible: [ {on} | off ]
```

Beispiel 12.3

Die Funktion ***get*** zeigt alle momentan eingestellten Eigenschaften eines Objekts. Beispiel:

```
get(h)
Color = [0 0 1]
EraseMode = normal
LineStyle = -
LineWidth = [0.5]
Marker = none
MarkerSize = [6]
MarkerEdgeColor = auto
MarkerFaceColor = none
XData = [ (1 by 7) double array]
YData = [ (1 by 7) double array]
ZData = []
ButtonDownFcn =
Children = []
Clipping = on
CreateFcn =
DeleteFcn =
BusyAction = queue
HandleVisibility = on
HitTest = on
Interruptible = on
Parent = [180]
Selected = off
SelectionHighlight = on
Tag =
Type = line
UIContextMenu = []
UserData = []
Visible = on
```

Beispiel 12.4

Um einen speziellen Eigenschaftswert zu erfragen, verwendet man

```
get(h,'Color')
ans =
     0     0     1
```

Beispiel 12.5

Das Achsenobjekt beinhaltet viele detaillierte Eigenschaftsangaben zur ganzen Graphik. Der Titel ist zum Beispiel eines der Unterobjekte des Achsenobjekts. Man erhält durch folgenden Ausdruck den Handle zu diesem Unterobjekt.

```
t = get(a,'title')
```

Dieser Handle kann natürlich weiterverwendet werden.

```
set(t,'String','Temperature','FontAngle','oblique');
```

Beispiel 12.6

Die Funktion ***title*** bietet ebenfalls eine Zugriffsmöglichkeit auf dieses Objekt.

Graphische Benutzerschnittstellen

Es folgt ein einfaches Beispiel, das zeigt, wie mit Handle Graphics eine Benutzerschnittstelle erzeugt werden kann.

Der Ausdruck

```
b = uicontrol('Style', 'pushbutton', 'Units', 'normalized',...
              'Position', [.5 .5 .2 .1], 'String', 'click here');
```

erzeugt einen Schaltknopf in der Mitte eines Bildfensters und gibt einen Handle zurück. Wenn auf den Button geklickt wird passiert aber noch nichts. Der Ausdruck

```
s = 'set(b,''Position'',[.8*rand .9*rand .2 .1])';
```

erzeugt eine Zeichenkette, die ein Kommando enthält, welches die Position des Buttons verändert. Wiederholtes Ausführen von ***eval***(s) verschiebt den Button an unterschiedliche Positionen. Schließlich installiert

```
set(b,'Callback',s)
```

Beispiel 12.7

s als die auszuführende Aktion, so dass bei jedem Klick auf den Button, dieser seine Position verändert.

Animationen

MATLAB stellt eine Reihe von Möglichkeiten zur Verfügung, Graphiken zu bewegen, zu animieren. Die Eigenschaft ***EraseMode*** ist für lange Folgen einfacher Graphiken angebracht, in deren Verlauf sich von Bild zu Bild nur wenig ändert.

Im folgenden Beispiel wird die Brownsche Bewegung simuliert. Zuerst wird die Anzahl von Punkten festgelegt, etwa

```
n = 20;
```

dann eine Temperatur oder Geschwindigkeit, wie z.B.

```
s = .02;
```

Die optimalen Werte für diese beiden Parameter hängen von der Arbeitsgeschwindigkeit Ihres Computers ab. Nun erzeugen wir n zufällige gleichverteilte Punkte mit xy-Koordinaten zwischen -1/2 und +1/2

```
x = rand(n,1)-0.5;
y = rand(n,1)-0.5;
```

Wir zeichnen diese Punkte in ein Quadrat mit Seiten von -1 bis +1, speichern den Handle des Punktvektors und setzen die Eigenschaft ***EraseMode*** auf ***xor***. Diese Einstellung teilt MATLAB mit, dass es nicht immer die komplette Darstellung neu zeichnen soll, wenn sich nur ein Punkt verändert hat. Durch ***xor*** wird nur der geänderte Hintergrund wieder hergestellt.

```
h = plot(x,y,'.');
axis([-1 1 -1 1])
axis square
grid off
set(h,'EraseMode','xor','MarkerSize',18);
```

Nun beginnen wir, die Animation zu erstellen. Hier benutzen wir eine Endlosschleife. (Um sie später wieder abzubrechen, müssen Sie [Strg][C] drücken.) Bei jedem Schleifendurchlauf wird eine zufällig gewählte normalverteilte Störung zu den Koordinaten jedes einzelnen Punktes hinzuaddiert. Anstatt nun eine vollständig neue Graphik zu erzeugen, werden nur die XData- und YData-Eigenschaften des Bildes verändert.

```
while 1
   drawnow
   x = x + s*randn(n,1);
   y = y + s*randn(n,1);
   set(h,'XData',x,'YData',y)
end
```

Beispiel 12.8

Filme

Wenn Sie nun die Anzahl der Punkte in der Brownschen Bewegung auf einen größeren Wert, z.B. n = 500, erhöhen, ist die Bewegung nicht mehr länger flüssig, sondern erscheint abgehackt. Es dauert einfach zu lange, jeden Zeitabschnitt zu zeichnen. Jetzt wird es effektiver eine vorgegebene Anzahl von Bildern als Bitmap zu speichern und sie später als Film abzuspielen.

Zuerst muß man sich auf die Anzahl der Bilder (Frames) festlegen, sagen wir

```
nframes = 50;
```

Als nächstes bereiten wir die Darstellung vor wie oben mit dem einzigen Unterschied, dass die Eigenschaft ***EraseMode*** nicht verwendet wird.

```
x = rand(n,1)-0.5;
y = rand(n,1)-0.5;
h = plot(x,y,'.');
axis([-1 1 -1 1])
axis square; grid off
```

Nun wird genügend Speicher reserviert, um den kompletten Film aufzunehmen.

```
M = moviein(nframes);
```

Dies ergibt eine riesige Matrix mit *nframes* Spalten. Jede Spalte ist groß genug, um ein komplettes Bild zu speichern. Die Größe des gesamten benötigten Speichers ist proportional zu *nframes* und den Achsenabschnitten, aber unabhängig von der Komplexität der entsprechenden Darstellung. Für 50 Bilder mit den oben eingestellten Achsenwerten würden 7,5 Megabyte Speicher benötigt. In unserem folgenden Beispiel werden allerdings nur 6 Megabyte benötigt, weil etwas kleinere Achsen für das Quadrat verwendet werden.

Wir generieren nun den Film und verwenden die Funktion ***getframe*** für die Erfassung eines jeden Bildes.

```
for k = 1:nframes
   x = x + s*randn(n,1);
   y = y + s*randn(n,1);
   set(h,'XData',x,'YData',y)
   M(:,k) = getframe;
end
```

Zum Schluß wird der Film 30 mal abgespielt mit

```
movie(M,30)
```

Beispiel 12.9

13. Weiterführendes Material und Literatur:

Um mehr MATLAB-Beispiele zu erhalten, wählen sie Beispiele und Demos im Menü oder geben Sie

```
demo
```

ein. Starten Sie die gewünschten Demos und folgen sie den dort angezeigten weiteren Anweisungen.

Detailliertere Beschreibungen und Erklärungen, der in diesem Buch angeschnittenen Themen, enthalten folgende Handbücher

- „The MATLAB Installation Guide“ beschreibt wie MATLAB richtig installiert wird.
- „Using MATLAB“ gibt einen tiefen Einblick in die Sprache, die Arbeitsumgebung und die mathematischen Möglichkeiten von MATLAB.
- „Using MATLAB Graphics“ beschreibt die Verwendung von MATLAB's Graphik- und Darstellungswerkzeuge.
- „The MATLAB Application Program Interface Guide“ beschreibt, wie man C/C++- oder Fortran- Programme in Interaktion mit MATLAB benutzt.
- „MATLAB Product Family New Features“ bietet Informationen über die Unterschiede der verschiedenen Versionen und gibt Hilfestellungen beim Wechsel von einer Version zu einer anderen.

MATLAB Toolboxen sind Sammlungen von M-Files für die unterschiedlichsten technischen Bereiche und erweitern die fachspezifischen Möglichkeiten von MATLAB. Für die Toolboxen gibt es entsprechende Bücher. Einige Toolboxen sind z.B.

- Communications
- Control Systems
- Financial Computation
- Frequency-Domain System Identification
- Fuzzy Logic
- Higher-Order Spectral Analysis
- Image Processing
- Linear Matrix Inequalities
- Model Predictive Control
- Mu-Analysis and Synthesis
- Numerical Algorithms
- Neural Networks
- Optimization
- Partial Differential Equations

- Quantitative Feedback Theory
- Robust Control
- Signal Processing
- Simulation (Simulink)
- Splines
- Statistics
- Symbolic Mathematics
- System Identification
- Wavelets

Die neuesten Informationen über MATLAB und andere MathWorks-Produkte finden Sie auf unserer Homepage im Internet:

http://www.mathworks.com

Mit ihrem Internet News Leser haben Sie Zugriff auf die Newsgroup

comp.soft-sys.MATLAB

Es gibt weitere Bücher über MATLAB von anderen Verlagen, informieren Sie sich in ihrer Buchhandlung oder holen sie sich die Broschüre „MATLAB-Based Books“ von The MathWorks oder im Internet über unsere Homepage.

Haben Sie es bis hierher geschafft, alles gelesen und die Übungen gemacht, dann gratulieren wir Ihnen. Sie haben einen ausführlichen Einstieg hinter sich und werden viel Freude mit MATLAB haben. Viel Spaß, ihr MATLAB-Team.

14. Übungen aus verschiedenen Themenbereichen

14.1 Grundrechenarten

1.► Man löse die folgenden Vektorgleichungen zunächst allgemein nach $\vec{x}$ auf und berechne $\vec{x}$ für die angegebenen Werte!

a) $2\vec{a}-3\vec{b}+\vec{c}-2\vec{x}=\vec{0}$; $\quad \vec{a}=\begin{bmatrix}4\\-3\\-2\end{bmatrix}$ $\quad \vec{b}=\begin{bmatrix}-7\\-1\\2\end{bmatrix}$ $\quad \vec{c}=\begin{bmatrix}5\\3\\1\end{bmatrix}$

b) $\vec{a}+3\vec{b}+\vec{x}=4(\vec{x}-\vec{a})+5\vec{a}+2(\vec{x}-\vec{b})$; $\quad \vec{b}=\begin{bmatrix}-1\\2\\0\end{bmatrix}$

Lösung 14.1.1

2.► Es sei $\mathbf{A}=\begin{bmatrix}6&5&4\\3&2&1\end{bmatrix}$, $\mathbf{B}=\begin{bmatrix}2&2&0\\1&1&0\end{bmatrix}$ und $\mathbf{C}=\begin{bmatrix}2&2&0\\4&2&8\end{bmatrix}$. Man bestimme die Lösung der Matrizengleichung: $4\mathbf{A}+5\mathbf{X}+\mathbf{B}=3\mathbf{C}+3\mathbf{X}$.

Lösung 14.1.2

3.► Gegeben seien die Matrizen $\mathbf{X}=\begin{bmatrix}1\\2\\3\\4\end{bmatrix}$ und $\mathbf{Y}=\begin{bmatrix}-1&0&1&2\end{bmatrix}$.

Berechnen Sie, falls möglich, die Matrizen $\mathbf{Y}^{\mathrm{T}}(2\mathbf{X}^{\mathrm{T}}-\mathbf{Y})$, $\mathbf{Y}(\mathbf{X}+\mathbf{Y}^{\mathrm{T}})$.

Lösung 14.1.3

4.► Mit $\mathbf{A}=\begin{bmatrix}1&2&1\\-2&0&2\\3&0&-3\end{bmatrix}$, $\mathbf{B}=\begin{bmatrix}4&2&0\\-4&1&12\\0&12&-17\end{bmatrix}$ und $\mathbf{x}=\begin{bmatrix}0\\4\\-3\end{bmatrix}$ berechne man die Matrix:

$\mathbf{C}=\mathbf{x}^{\mathrm{T}}(\mathbf{B}+\mathbf{A}\mathbf{A}^{\mathrm{T}})$

Lösung 14.1.4

5.► Man berechne $(\mathbf{AB})\mathbf{C}$ und $\mathbf{A}(\mathbf{BC})$ und zeige die Gültigkeit des assoziativen Gesetzes am Beispiel der Matrizen:

$\mathbf{A}=\begin{bmatrix}4&-1\\2&3\end{bmatrix}$ $\quad \mathbf{B}=\begin{bmatrix}1&-1\\-1&1\end{bmatrix}$ $\quad \mathbf{C}=\begin{bmatrix}2&-3\\-1&3\end{bmatrix}$

Lösung 14.1.5

6.► Berechnen Sie das Produkt der Matrizen $\mathbf{A}$ und $\mathbf{B}$

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3\\ 3 & 1 & -2\\ 5 & -1 & 0\end{bmatrix} \qquad \mathbf{B}=\begin{bmatrix}1 & 3\\ -1 & 5\\ 2 & -2\end{bmatrix}$$

Lösung 14.1.6

7.► Bilden Sie die Matrizenprodukte $\mathbf{AB}$ und $\mathbf{BA}$ mit den Matrizen:

$$\mathbf{A}=\begin{bmatrix}5 & 7 & 0 & 6\\ 3 & -8 & 4 & 7\\ 6 & 0 & 2 & 1\end{bmatrix}, \quad \mathbf{B}=\begin{bmatrix}7 & 0 & 1\\ -3 & 6 & 4\\ 0 & -2 & -3\\ 6 & 0 & -1\end{bmatrix}$$

Lösung 14.1.7

8. Berechnen Sie das Matrizenprodukt

$$\mathbf{AB}=\begin{bmatrix}3.12 & 4.00 & -5.07\\ 5.23 & 7.00 & 3.89\\ 11.40 & 0.00 & 5.27\end{bmatrix}\cdot\begin{bmatrix}0.35 & 4.01 & -1.00\\ 1.00 & 3.00 & 4.30\\ 1.87 & 6.37 & -9.09\end{bmatrix}.$$

Berechnen Sie auch $3\mathbf{A}^2\mathbf{B}+5\mathbf{B}^3-7\mathbf{BA}$.

Lösung 14.1.8

9. Erzeugen Sie eine m x n-Matrix $\mathbf{A}$ und eine Einheitsmatrix $\mathbf{I}$ und zeigen Sie:

a) $\mathbf{I}_m\mathbf{A}=\mathbf{A}$

b) $\mathbf{A}\mathbf{I}_n=\mathbf{A}$

Lösung 14.1.9

10. Erzeugen Sie 6x6-Matrizen $\mathbf{A}$ und $\mathbf{B}$ und überprüfen Sie:

a) $(\mathbf{A}+\mathbf{I})(\mathbf{A}-\mathbf{I})=\mathbf{A}^2-\mathbf{I}$

b) $(\mathbf{A}+\mathbf{B})(\mathbf{A}-\mathbf{B})=\mathbf{A}^2-\mathbf{B}^2$

c) $(\mathbf{A}+\mathbf{B})^{\mathrm{T}}=\mathbf{A}^{\mathrm{T}}+\mathbf{B}^{\mathrm{T}}$

d) $(\mathbf{AB})^{\mathrm{T}}=\mathbf{A}^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}$

Lösung 14.1.10

11. $\mathbf{A}$ und $\mathbf{B}$ sind quadratische Matrizen gleicher Größe. Zeigen Sie das $(\mathbf{AB})^2-\mathbf{A}^2\mathbf{B}^2=\mathbf{A}(\mathbf{BA}-\mathbf{AB})\mathbf{B}$.

Wann ist $(\mathbf{AB})^2=\mathbf{A}^2\mathbf{B}^2$.

Lösung 14.1.11

12. Angenommen $\mathbf{A}$ ist eine symmetrische n x n-Matrix und $\mathbf{B}$ ist eine n x m-Matrix. Zeigen Sie, dass $\mathbf{B}^T\mathbf{AB}$, $\mathbf{B}^T\mathbf{B}$ und $\mathbf{BB}^T$ auch symmetrische Matrizen sind.

Lösung 14.1.12

13. Die nachfolgenden Matrix-Gleichungen sind nicht alle richtig. Benutzen Sie MATLAB, um die falschen herauszufinden und durch Beispiele zu belegen.

a) $\mathbf{A}+\mathbf{B}=\mathbf{B}+\mathbf{A}$

b) $\mathbf{AB}=\mathbf{BA}$

c) $(\mathbf{A}+\mathbf{B})^2=\mathbf{A}^2+2\mathbf{AB}+\mathbf{B}^2$

d) $(\mathbf{AB})^2=\mathbf{A}^2\mathbf{B}^2$

e) $\mathbf{A}(\mathbf{B}+\mathbf{C})=\mathbf{AB}+\mathbf{AC}$

f) $(\mathbf{A}+\mathbf{B})\mathbf{C}=\mathbf{CA}+\mathbf{CB}$

g) Falls $\mathbf{AB}=0$ ist, dann ist $\mathbf{A}=0$ oder $\mathbf{B}=0$

h) Falls $\mathbf{A}^2=0$ ist, dann ist $\mathbf{A}=0$

Lösung 14.1.13

14. Berechnen Sie $\mathbf{S}^k$ für k = 2,...,6.

$$\mathbf{S}=\begin{bmatrix}0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\\0&0&0&0&0\end{bmatrix}$$

Lösung 14.1.14

15. Beschreiben Sie was passiert, wenn Sie $\mathbf{A}^{10}, \mathbf{A}^{20}$ und $\mathbf{A}^{30}$ berechnen.

$$\mathbf{A}=\begin{bmatrix}1/6&1/2&1/3\\1/2&1/4&1/4\\1/3&1/4&5/12\end{bmatrix}$$

Lösung 14.1.15

14.2 Quadratische Gleichungssysteme

1.► Gegeben sind die Matrizen $\mathbf{A}=\begin{bmatrix}2&1\\2&3\end{bmatrix}$ und $\mathbf{B}=\begin{bmatrix}1&-1\\2&2\end{bmatrix}$.

Bestimmen Sie die Matrix $\mathbf{X}$, die der folgenden Gleichung genügt: $\mathbf{XA}+(\mathbf{I}-\mathbf{X})\mathbf{B}=\mathbf{I}$.

Lösung 14.2.1

2. Ein Experiment führt zu folgendem Gleichungssystem:

$$\begin{aligned} 4.5x_1 &+ 3.1x_2 &= 19.249 \\ 1.6x_1 &+ 1.1x_2 &= 6.843 \end{aligned}$$

a) Lösen sie das System und lösen Sie auch das nachfolgende System, bei dem die rechte Seite auf zwei Nachkommastellen gerundet wurde.

$$\begin{aligned} 4.5x_1 &+ 3.1x_2 &= 19.25 \\ 1.6x_1 &+ 1.1x_2 &= 6.84 \end{aligned}$$

b) Die Koeffizienten des zweiten Gleichungssystems weichen um weniger als 0.05 % von den entsprechneden Koeffizienten des ersten Systems ab. Ist die Abweichung der Lösungen entsprechend?

Lösung 14.2.2

3.► Bestimmen Sie die Lösungen der folgenden Gleichungssysteme

a)
$$\begin{aligned} x_2 &+ 3x_3 &= 1 \\ 7x_1 &- 4x_2 &- 2x_3 &= 0 \\ 5x_1 &+ x_2 &+ 3x_3 &= -2 \end{aligned}$$

b)
$$\begin{aligned} 8x_1 &- 10x_2 &- 228x_3 &= -112 \\ 2x_1 &- x_2 &- 4x_3 &= -16 \\ 4x_1 &- 5x_2 &- 14x_3 &= -56 \end{aligned}$$

Lösung 14.2.3

4. Um eine Getreidepest zu kontrollieren wird empfohlen das Getreide mit einer Mischung aus 6 Einheiten der Chemikalie A, 10 Einheiten der Chemikalie B und 8 Einheiten der Chemikalie C zu besprühen. Diese Zutaten sind in drei schon gemischten Packungen erhältlich: Eine Tonne des Spray S1 enthält 1, 3 und 4 Einheiten der Chemikalien A, B und C; eine Tonne des Spray S2 enthält 3 Einheiten von jeder Chemikalie und eine Tonne des Spray S3 enthält 2 Einheiten von A und 5 Einheiten von B. Wieviel von jedem herkömmlichen Spray sollte verwendet werden um die exakt benötigte Menge der Chemikalien zu erhalten?

Lösung 14.2.4

5.► Lösen Sie das folgende Gleichungssystem für $\sin\alpha$, $\cos\beta$ und $\tan\gamma$ und berechnen Sie dann die unbekannten Winkel.

$$\begin{aligned} -2\sin\alpha &- 3\cos\beta &+ 5\tan\gamma &= 1 \\ 4\sin\alpha &+ 2\cos\beta &- 2\tan\gamma &= 2 \\ 2\sin\alpha &- 5\cos\beta &+ 3\tan\gamma &= 7 \end{aligned}$$

Lösung 14.2.5

6.► Eine Farm hat 3 Arten Vieh: Schafe, Ziegen und Schweine. Es gibt drei verschiedene Futtersorten: A, B und C. Jedes Schaf braucht jedes Monat 3 Einheiten vom Futter A, 7 Einheiten von B und 5 Einheiten von C. Jede Ziege braucht 4 Einheiten von A, 8 Einheiten von B und 7 Einheiten von C. Jedes Schwein verbraucht 4 Einheiten von A, 9 Einheiten von B und 6 Einheiten von C in jedem Monat. Wie viele Tiere von jeder Art kann die Farm halten, wenn

jedes Monat 129 Einheiten des Futter A, 285 Einheiten von B und 210 Einheiten von C zur Verfügung stehen und alles verbraucht werden soll?

Lösung 14.2.6

7.► Ist das folgende Gleichungssystem lösbar? Bestimmen Sie gegebenenfalls die Lösungsmenge.

$$\begin{array}{rcrcrcrcl} x_1 & + & x_2 & + & x_3 & + & x_4 & = & 1 \\ x_1 & - & x_2 & + & x_3 & - & x_4 & = & 2 \\ 4x_1 & + & 2x_2 & & & + & 2x_4 & = & 7 \\ x_1 & + & x_2 & - & x_3 & - & x_4 & = & 3 \end{array}$$

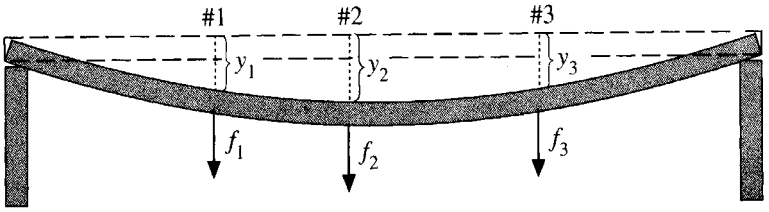
Lösung 14.2.7

8. Lösen Sie das folgende Gleichungssystem:

$$\begin{bmatrix} 0.4095 & 0.0124 & 0.3678 & 0.2945 \\ 0.3654 & 0.1902 & 0.3718 & 0.0634 \\ 0.1748 & 0.4000 & 0.2708 & 0.3925 \\ 0.2251 & 0.3842 & 0.4015 & 0.1129 \end{bmatrix} \cdot \mathbf{X} = \begin{pmatrix} 3.142 \\ 5.312 \\ 1.000 \\ -4.312 \end{pmatrix}$$

Lösung 14.2.8

9. Auf einen horizontalen elastischen Balken, der an den Enden fest aufliegt, greifen in den Punkten 1 bis 4 Kräfte an. Im Vektor f im $\Re^4$ stehen die Kräfte an diesen Punkten und der Vektor y im $\Re^4$ listet die Abweichung von der Horizontalen (d.h. die Bewegung des Balkens) in den vier Punkten auf. Das Gesetz von Hooke sagt: $\mathbf{y} = \mathbf{Df}$.
($\mathbf{D}$ ist die Biegsamkeitsmatrix)



(Skizze mit 3 Kräften)

Die nachfolgende Matrix $\mathbf{D}$ ist eine Biegsamkeitsmatrix für 4 Punkte an denen Kräfte angreifen. Die Einheit ist Zentimeter/Newton.

$$\mathbf{D} = \begin{bmatrix} 0.0040 & 0.0030 & 0.0010 & 0.0005 \\ 0.0030 & 0.0050 & 0.0030 & 0.0010 \\ 0.0010 & 0.0030 & 0.0050 & 0.0030 \\ 0.0005 & 0.0010 & 0.0030 & 0.0040 \end{bmatrix}$$

Messungen an den 4 Punkten zeigen Auslenkungen von 0.25, 0.30, 0.35 und 0.30 cm. Berechnen Sie die Kräfte an den 4 Punkten.

Berechnen Sie außerdem die Kräfte, die eine Bewegung von 0.03 cm im 4. Punkt und keine Abweichung in den anderen 3 Punkten hervorrufen.

Lösung 14.2.9

10.► Bestimmen Sie die Lösung des folgenden quadratischen Gleichungssystems

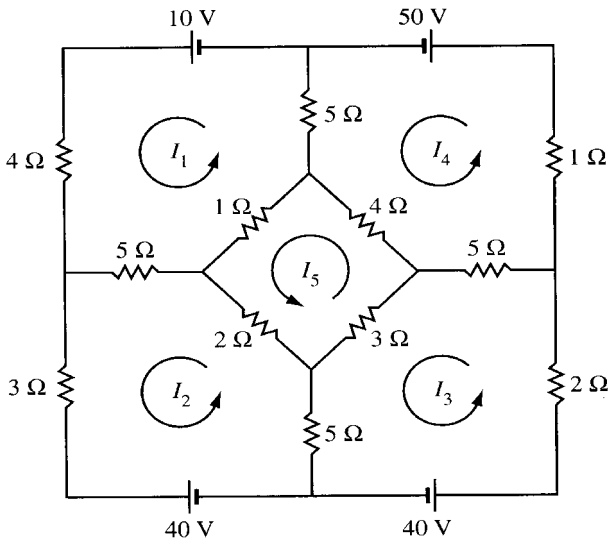
$$\begin{array}{rcrcrcrcrcr} 2x_1 & - & 3x_2 & + & x_3 & - & 2x_4 & + & 4x_5 & = & 7 \\ -4x_1 & + & 6x_2 & - & 2x_3 & + & 5x_4 & - & 6x_5 & = & -10 \\ 6x_1 & - & 9x_2 & + & 3x_3 & - & 4x_4 & + & 10x_5 & = & 17 \\ 2x_1 & - & 4x_2 & + & 3x_3 & + & 2x_4 & - & 3x_5 & = & 5 \\ -2x_1 & + & 5x_2 & - & 3x_3 & + & 2x_4 & - & x_5 & = & 1 \end{array}$$

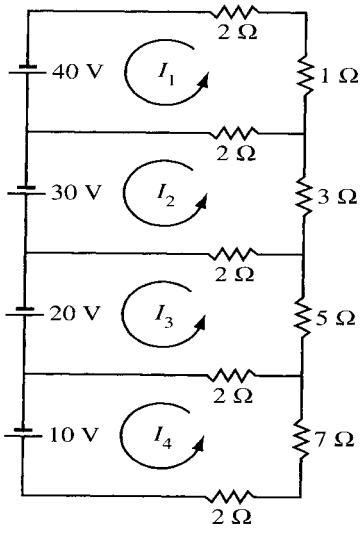
Lösung 14.2.10

11. Die Stromstärken I in den Widerständen eines elektrischen Netzwerkes sollen bestimmt werden. Zur Lösung dieses Problems untersucht man die Ströme in den einzelnen Maschen. Dabei gilt das Ohmsche Gesetz: $\mathbf{U} = \mathbf{RI}$.

Eine Gleichung innerhalb der ersten Masche sieht dann z.B. folgendermaßen aus:
$4\Omega * \mathbf{I}_1 + 4\Omega * \mathbf{I}_1 + 3\Omega * (\mathbf{I}_1 - \mathbf{I}_2) = 30V$
Dabei gehören die 4Ω-Widerstände nur zur ersten Masche, während der 3Ω-Widerstand zur ersten und zweiten Masche gehört. Die angelegte Spannung muß negativ angegeben werden, wenn der Strom in die entgegengesetzte Richtung fließt.

Durch Aufstellen der Gleichungen für jede einzelne Masche erhält man ein quadratisches lineares Gleichungssystem, daß nach den $\mathbf{I}_n$ gelöst werden kann.
Bestimmen Sie die Stromstärke in den einzelnen Maschen der nachfolgenden Netze.





Lösung 14.2.11

14.3 Determinanten

1. Angenommen $\mathbf{A}$ ist eine 3x3-Matrix mit $\det(\mathbf{A}) = 5$. Was ist

   a) $\det(-\mathbf{A})$

   b) $\det(\mathbf{A}^2)$

   c) $\det(2\mathbf{A})$

   d) $\det((\frac{1}{3}\mathbf{A})^3)$?

   Lösung 14.3.1

2. Berechnen Sie die Determinanten der folgenden Matrizen

$$\mathbf{A} = \begin{bmatrix} 3 & -2 & 1 & 6 & 0 \\ 5 & -4 & 2 & 11 & 1 \\ 1 & -6 & 3 & 10 & 4 \\ -2 & 2 & -1 & -5 & 3 \\ 7 & -7 & 3 & 16 & -2 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 2 & -3 & 1 & -2 & 4 \\ -4 & 6 & -2 & 5 & -6 \\ 6 & -9 & 3 & -4 & 10 \\ 2 & -4 & 3 & 2 & -3 \\ -2 & 5 & -3 & 2 & -1 \end{bmatrix}$$

   Lösung 14.3.2

3.► Die Zahlen 20604, 53227, 25755, 20927, 78421 sind durch 17 teilbar. Man zeige, daß der Wert der nebenstehenden Determinante ebenfalls durch 17 teilbar ist.

$$\det\begin{bmatrix} 2 & 0 & 6 & 0 & 4 \\ 5 & 3 & 2 & 2 & 7 \\ 2 & 5 & 7 & 5 & 5 \\ 2 & 0 & 9 & 2 & 7 \\ 7 & 8 & 4 & 2 & 1 \end{bmatrix}$$

   Lösung 14.3.3

4.► Berechnen Sie die folgende Determinante:

$$\det\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

   Lösung 14.3.4

5. Ein großer digitaler Computer braucht $2.9 \cdot 10^{-6}$ Sekunden für eine Multiplikation. Überprüfen Sie wie lange MATLAB braucht, um die Determinante einer 50x50-Matrix zu berechnen.

Wie viele Gleitpunktoperationen waren dazu notwendig? Allgemein braucht man für die Berechnung der Determinante einer n x n-Matrix $\frac{2}{3}\cdot n^3 - \frac{1}{2}\cdot n^2 + \frac{5}{6}\cdot n - 1$ Gleitpunktoperationen.

Lösung 14.3.5

6. Berechnen Sie $\det(\mathbf{A}^T\mathbf{A})$ und $\det(\mathbf{A}\mathbf{A}^T)$ für mehrere 4x5-Zufallsmatrizen und mehrere 5x6-Zufallsmatrizen. Was kann man über $\mathbf{A}^T\mathbf{A}$ und $\mathbf{A}\mathbf{A}^T$ sagen, wenn $\mathbf{A}$ mehr Spalten als Zeilen hat?

Lösung 14.3.6

7. Erzeugen Sie zwei 8x8-Zufallsmatrizen $\mathbf{A}$ und $\mathbf{B}$ mit ganzzahligen Einträgen zwischen –9 und 9.

   a) Stimmt die Aussage: $\det(\mathbf{A}+\mathbf{B}) = \det(\mathbf{A}) + \det(\mathbf{B})$?

   b) Stimmt die Aussage: $\det(\mathbf{A}\mathbf{B}) = \det(\mathbf{A})\cdot\det(\mathbf{B})$

   c) Vergleichen Sie $\det(\mathbf{A})$ mit: $\det(\mathbf{A}^T), \det(-\mathbf{A}), \det(2\mathbf{A})$ und $\det(10\mathbf{A})$.

Lösung 14.3.7

8. Zeigen Sie, daß für jede (quadratische) Matrix gilt: $\det(k\mathbf{A}) = k^n \det(\mathbf{A})$.

Lösung 14.3.8

14.4 Inverse

1.► Gegeben seien die Matrizen $\mathbf{A} = \begin{bmatrix} 3 & 17 \\ 5 & 2 \end{bmatrix}$ und $\mathbf{B} = \begin{bmatrix} 1 & 0 & -5 \\ 7 & 3 & 4 \\ 0 & 2 & 11 \end{bmatrix}$.

Zeigen Sie, daß die Matrizen invertierbar sind und bestimmen Sie ihre Inverse.

Lösung 14.4.1

2.► Berechnen Sie mit ***inv*** und ***rref*** die Inversen der folgenden Matrizen

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 5 & 7 & 3 \\ 7 & 11 & 2 \\ 3 & 2 & 6 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Lösung 14.4.2

3. Um eine Nachricht zu verschlüsseln werden die Buchstaben von A bis Z durch die Zahlen von 1 bis 26 ersetzt. Ein Leerzeichen entspricht der Null. Dann werden die Zahlenfolgen in Dreierblöcke zerteilt und die Verschlüsselung erfolgt durch Multiplikation jedes Blocks mit der folgenden Matrix:

$$\mathbf{A} = \begin{bmatrix} -3 & 5 & 6 \\ -1 & 2 & 2 \\ 1 & -1 & -1 \end{bmatrix}$$

Der Empfänger erhält dann eine Zahlenfolge die er mit der Inversen der Matrix $\mathbf{A}$ wieder entschlüsseln kann.

a) Verschlüsseln Sie die folgenden Nachrichten:

ANGRIFF BEI SONNENUNTERGANG

PARTY UM MITTERNACHT

b) Entschlüsseln Sie die folgenden Nachrichten:

117, 44, -11, 6, 5, 7, 76, 32, -12, 87, 32, -5, 55, 23, -9

87, 36, -9, 119, 44, -22, 31, 13, -5, 67, 27, -13

Lösung 14.4.3

4. Bestimmen Sie die Inverse der folgenden Matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & -3 & -4 \\ 1 & -2 & 3 & -4 \\ 1 & -2 & -3 & 4 \end{bmatrix}$$

Lösung 14.4.4

5. Berechnen Sie die Inversen der Matrizen und berechnen Sie jeweils $\mathbf{A}\mathbf{A}^{-1}$ und $\mathbf{A}^{-1}\mathbf{A}$.

$$\mathbf{A} = \begin{bmatrix} 2.35 & 4.12 & 3.17 & 4.31 \\ -4.27 & 8.36 & 7.15 & 9.87 \\ 5.87 & 0.00 & -5.89 & 18.00 \\ 11.30 & 14.70 & 11.20 & 5.75 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

Lösung 14.4.5

6. Ist die Matrix M invertierbar?

$$\mathbf{M} = \begin{bmatrix} 5 & 4 & 3 & 6 & 3 \\ 7 & 6 & 5 & 9 & 5 \\ 8 & 6 & 4 & 10 & 4 \\ 9 & -8 & 9 & -5 & 8 \\ 10 & 8 & 7 & -9 & 7 \end{bmatrix}$$

Lösung 14.4.6

7. $\mathbf{A}_n$ ist eine n x n-Matrix mit Nullen auf der Hauptdiagonalen und sonst Einsen. Berechnen Sie $\mathbf{A}_n^{-1}$ für n = 4, 5 und 6 und stellen Sie eine Vermutung für die allgemeine Form von $\mathbf{A}_n^{-1}$ für größere Werte von n auf.

Lösung 14.4.7

8. In MATLAB gibt es einen Befehl zum erzeugen von Hilbert-Matrizen in beliebiger Größe. (Die Elemente der Matrix berechnen sich mit der Formel: 1/(i+j-1); i = Zeilennummer, j = Spaltennummer des jeweiligen Elements). Berechnen Sie mit einen Befehl zur Inversenberechnung die Inverse einer 8x8 oder größeren Hilbert-Matrix $\mathbf{A}$. Berechnen Sie $\mathbf{A}\mathbf{A}^{-1}$. Beschreiben Sie das Ergebnis.

Lösung 14.4.8

9.► Gegeben sind die folgenden Matrizen

$$\mathbf{A}=\begin{bmatrix}2 & 1\\ 0 & -1\\ 3 & 0\end{bmatrix} \qquad \mathbf{B}=\begin{bmatrix}1 & 3\\ -1 & 5\end{bmatrix} \qquad \mathbf{C}=\begin{bmatrix}3 & 2\\ -1 & -2\\ 0 & 2\end{bmatrix} \qquad \mathbf{D}=\begin{bmatrix}1 & 2\end{bmatrix}$$

Führen Sie in MATLAB die folgenden Operationen durch, falls sie definiert sind:

a) $\mathbf{D}\mathbf{B}^{2}$

b) $\mathbf{B}\mathbf{C}^{\mathrm{T}}$

c) $(\mathbf{C}\mathbf{B})\mathbf{D}^{\mathrm{T}}$

d) $\mathbf{A}\mathbf{C}^{\mathrm{T}}$

e) $(\mathbf{A}\mathbf{C}^{\mathrm{T}})^{-1}$

f) $\det(\mathbf{B})$

g) $\det(\mathbf{A}\mathbf{C}^{\mathrm{T}})$

h) $(\mathbf{C}^{\mathrm{T}}\mathbf{A})^{-1}$

Lösung 14.4.9

10.► Zeigen Sie, daß für die Matrix $\mathbf{A}=\begin{bmatrix}1 & -3 & 4\\ -2 & 8 & -6\\ 3 & -5 & 15\end{bmatrix}$ gilt: $\det(\mathbf{A})\cdot\det(\mathbf{A}^{-1})=1$.

In welcher Beziehung stehen $\det(\mathbf{A}^{-1})$ und $\det(\mathbf{A})$ allgemein?

Lösung 14.4.10

11. Vergleichen Sie mit den Kommandos ***tic*** und ***toc*** die benötigten Zeiten für das Lösen eines linearen Gleichungssystems einmal mit $\mathbf{A}\setminus\mathbf{b}$ und zum anderen mit $\mathbf{inv}(\mathbf{A})*\mathbf{b}$. Wählen Sie dazu eine Zufallsmatrix $\mathbf{A}$ der Größe n=200 und n=500 sowie eine entsprechende Seite $\mathbf{b}$.

Lösung 14.4.11

14.5 Adjunkte und Cramersche Regel

1.► Bestimmen Sie die Determinante und die Adjunkte von $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 1 & 5 & 7 \end{bmatrix}$.

Lösung 14.5.1

2. Ermitteln Sie nur unter Verwendung von Determinanten die Inverse der Matrix

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 1.1 \\ 1 & 1.1 & 0.9 \\ 0.9 & 1 & 0.8 \end{bmatrix}$$

Lösung 14.5.2

3. Berechnen Sie die Inverse der folgenden Matrix mit Hilfe der Adjunkten.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 2 & -1 \\ 3 & -2 & 6 & 4 \\ 2 & 2 & -5 & 6 \\ 5 & 4 & 3 & 0 \end{bmatrix}$$

Lösung 14.5.3

4. Berechnen Sie die Inverse einer 4x4-Zufallsmatrix $\mathbf{A}$ mit Hilfe von Unterdeterminanten. Berechnen Sie danach die Inverse mit dem Befehl $\mathbf{inv(A)}$. Vergleichen Sie die beiden Ergebnisse und vergleichen sie die Zeit, die MATLAB braucht, um die Inverse mit den beiden unterschiedlichen Wegen zu berechnen.

   Berechnen Sie die Anzahl der Gleitpunktoperationen bei der Berechnung der Inversen einer 30x30-Zufallsmatrix. Vergleichen Sie die beiden Berechnungsmöglichkeiten von oben. Für eine n x n-Matrix gelten dabei folgende Formeln: Zur Berechnung der Determinante braucht man $\frac{2}{3} \cdot n^3 - \frac{1}{2} \cdot n^2 + \frac{5}{6} \cdot n - 1$ und bei der Berechnung der Inversen $\frac{8}{3} \cdot n^3 - \frac{1}{2} \cdot n^2 - \frac{1}{6} \cdot n$ Gleitpunktoperationen.

Lösung 14.5.4

5.► Lösen Sie das lineare Gleichungssystem nach der Cramer`schen Regel

$$\begin{aligned} 19x_1 + 34x_2 + 45x_3 &= 177 \\ 7x_1 + 11x_2 + 13x_3 &= 54 \\ 11x_1 + 20x_2 + 30x_3 &= 108 \end{aligned}$$

Lösung 14.5.5

6. Berechnen Sie $\mathbf{x}$ mit der Cramer`schen Regel

$$\begin{bmatrix} 1 & 1 & 2 & 1 \\ -1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 0 \\ 1 & 3 & 1 & 0 \end{bmatrix} \cdot \mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Lösung 14.5.6

7. Testen Sie die Cramer`sche Regel für eine 4x4-Zufallsmatrix $\mathbf{A}$ und einen 4x1-Zufallsvektor $\mathbf{b}$. Berechnen Sie die Lösung von $\mathbf{Ax} = \mathbf{b}$ und vergleichen Sie sie mit der Lösung von $\mathbf{A}^{-1}\mathbf{b}$.

Lösung 14.5.7

14.6 Unabhängigkeit, Rang

1.► Gegeben seien die Vektoren

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \qquad \mathbf{d} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Sind die folgenden Vektorsysteme linear abhängig?

a, **b** **a**, **b**, **c** **a**, **b**, **d** **a**, **b**, **c**, **d** **a**, **b**, **a** **a**

Lösung 14.6.1

2.► Kann der Vektor $\mathbf{x}$ durch eine Linearkombination der Vektoren $\mathbf{v}_1, \mathbf{v}_2$ und $\mathbf{v}_3$ dargestellt werden?

$$\mathbf{v}_1 = \begin{bmatrix} -6 \\ 4 \\ -9 \\ 4 \end{bmatrix} \qquad \mathbf{v}_2 = \begin{bmatrix} 8 \\ -3 \\ 7 \\ -3 \end{bmatrix} \qquad \mathbf{v}_3 = \begin{bmatrix} -9 \\ 5 \\ -8 \\ 3 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} 4 \\ 7 \\ -8 \\ 3 \end{bmatrix}$$

Lösung 14.6.2

3. Überprüfen Sie die Spaltenvektoren der Matrizen $\mathbf{A}$ und $\mathbf{B}$ auf lineare Unabhängigkeit.

$$\mathbf{A} = \begin{bmatrix} 8 & 4 & -1 & 6 & -1 \\ 9 & 5 & -4 & 8 & 4 \\ -3 & 1 & -9 & 4 & 11 \\ -6 & -4 & 6 & -7 & -8 \\ 0 & 4 & -7 & 10 & -7 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -8 & 8 & -8 & 1 & -9 \\ 7 & -7 & 7 & 4 & 3 \\ 6 & -9 & 4 & 9 & -4 \\ 5 & -5 & 5 & 6 & -1 \\ -7 & 7 & -7 & -7 & 0 \end{bmatrix}$$

Bilden die Vektoren jeweils eine Basis des $\Re^5$?

Lösung 14.6.3

4.► Bestimmen Sie den Rang der Matrix $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 4 \\ 2 & 3 & 1 & 5 & -5 \\ 1 & 3 & 3 & 7 & 2 \\ 5 & 6 & 5 & 8 & -2 \\ 3 & 4 & 4 & 6 & 1 \end{bmatrix}$.

Lösung 14.6.4

5. Zeigen Sie, daß der Rang eines Matrizenproduktes den Rang der einzelnen Faktoren nicht übersteigen kann. $\mathbf{B}$ ist eine n x m-Matrix, dann ist

   a) $rang(\mathbf{AB}) \leq rang(\mathbf{A})$

   b) $rang(\mathbf{AB}) \leq rang(\mathbf{B})$

   Lösung 14.6.5

6. Zeigen Sie, daß wenn $\mathbf{P}$ eine invertierbare m x m-Matrix ist, dann ist

   a) $rang(\mathbf{PA}) = rang(\mathbf{A})$

   b) $rang(\mathbf{AP}) = rang(\mathbf{A})$

   Lösung 14.6.6

14.7 Gleichungssysteme

1.► Finden Sie die allgemeine Lösung des folgenden linearen Gleichungssystems. Verwenden Sie die MATLAB-Funktion ***rref***.

$$\begin{aligned} -3x_1 - 9x_2 + 6x_3 &= 15 \\ 2x_1 + 9x_2 + x_3 &= -16 \end{aligned}$$

Lösung 14.7.1

2.► a) Zeigen Sie, daß die folgende Matrix singulär ist: $\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix}$.

b) Es sei $\mathbf{b} = [2 \quad 4 \quad 6]^T$. Wie viele Lösungen hat das System $\mathbf{Ax} = \mathbf{b}$?

Lösung 14.7.2

3.► Lösen die folgende Reihe linearer Gleichungssysteme, mit der Koeffizientenmatrix $\mathbf{A}$ und verschiedenen rechten Seiten $\mathbf{b}_1$, $\mathbf{b}_2$ und $\mathbf{b}_3$.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -2 & 0 \end{bmatrix} \quad \mathbf{b}_1 = \begin{pmatrix} 0 \\ 1 \\ -1 \\ 3 \end{pmatrix} \quad \mathbf{b}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ -5 \end{pmatrix} \quad \mathbf{b}_3 = \begin{pmatrix} 0 \\ 0 \\ -3 \\ 5 \end{pmatrix}$$

Lösung 14.7.3

4.► Prüfen Sie, ob die folgenden Gleichungssysteme lösbar sind und bestimmen Sie gegebenenfalls die Lösungsgesamtheit.

a)
$$\begin{aligned} 15x_1 \quad\quad &+ 17x_3 + 9x_4 + 4x_5 = 0 \\ 5x_1 - 2x_2 &+ 7x_3 + 3x_4 + 2x_5 = -1 \\ 3x_2 - 2x_3 \quad\quad &- x_5 = 2 \end{aligned}$$

b)
$$\begin{aligned} x_1 + x_2 - x_3 &= 1 \\ -3x_1 - x_2 + 4x_3 &= 2 \\ 2x_1 + 8x_2 + x_3 &= 17 \\ -4x_1 - 2x_2 + 5x_3 &= 1 \end{aligned}$$

c)
$$\begin{aligned} x_1 - 4x_2 + 2x_3 \quad\quad &= -1 \\ 2x_1 - 3x_2 - x_3 - 5x_4 &= -7 \\ 3x_1 - 7x_2 + x_3 - 5x_4 &= -8 \\ x_2 - x_3 - x_4 &= -1 \end{aligned}$$

Lösung 14.7.4

5.► Bestimmen Sie die allgemeine Lösung der folgenden linearen Gleichungssysteme. Welche der linearen Gleichungssysteme $\mathbf{Ax} = \mathbf{b}$ mit Koeffizientenmatrizen $\mathbf{A}$ sind für beliebige rechte Seiten $\mathbf{b}$ lösbar?

a)
$$\begin{aligned} 11x_1 + 6x_2 + 4x_3 &= 1 \\ 5x_1 + 4x_2 - 2x_3 &= 3 \\ 3x_1 + 6x_3 &= -9 \end{aligned}$$

b)
$$\begin{aligned} x_1 + 2x_2 + 3x_3 + 4x_4 &= 0 \\ x_1 + 2x_2 + 4x_3 + 3x_4 &= -2 \\ 2x_1 + 4x_2 + 7x_3 + x_4 &= -8 \end{aligned}$$

c)
$$\begin{aligned} 2x_1 + 3x_2 + x_3 - 2x_4 \quad\quad &= 5 \\ x_1 + 4x_2 - 3x_3 + 6x_4 - 3x_5 &= 5 \\ 3x_1 + 7x_2 - 2x_3 + 4x_4 - 3x_5 &= 10 \end{aligned}$$

d)
$$\begin{aligned} x_1 + 3x_2 - 2x_3 - x_4 + x_5 + 4x_6 &= -1 \\ x_1 + 4x_2 - 3x_3 \quad\quad + x_5 + 6x_6 &= -1 \\ -2x_1 - 8x_2 + 6x_3 + 3x_4 + 2x_5 - 13x_6 &= 6 \\ -x_1 \quad\quad - x_3 + 7x_4 + 3x_5 + 3x_6 &= 9 \end{aligned}$$

Lösung 14.7.5

6.► Lösen Sie das folgende Gleichungssystem mit dem Gaußschen Verfahren und bestimmen Sie seinen Rang:

$$
\begin{array}{rcrcrcrcrcl}
x_1 & + & x_2 & + & x_3 & + & x_4 & & & = & 1 \\
2x_1 & + & 2x_2 & & & + & 5x_4 & + & x_5 & = & 2 \\
x_1 & + & x_2 & - & x_3 & + & 2x_4 & + & x_5 & = & 3 \\
x_1 & + & x_2 & - & x_3 & + & x_4 & + & 2x_5 & = & 4 \\
2x_1 & + & 2x_2 & - & 2x_3 & + & 3x_4 & + & 3x_5 & = & 7
\end{array}
$$

Lösung 14.7.6

7. Die nachfolgende chemische Reaktion kann in einigen industriellen Prozessen verwendet werden, wie z.B. der Produktion von Arsen. Gleichen Sie die Reaktionsgleichung aus.

$$\mathbf{MnS + As_2Cr_{10}O_{35} + H_2SO_4 \rightarrow HMnO_4 + AsH_3 + CrS_3O_{12} + H_2O}$$

Stellen Sie für jede der 7 Verbindungen einen Vektor im $\Re^6$ auf, der die Anzahl der Atome jedes Mangan-Molekül (Mn), Schwefel-Molekül (S), Arsen-Molekül (As), Chrom-Molekül (Cr), Sauerstoff-Molekül (O) und Wasserstoff-Molekül (H) auflistet. Schreiben Sie eine Vektorgleichung, so daß die unbekannten Koeffizienten die Reaktion ausgleichen und finden Sie eine ganzzahlige Lösung.

Lösung 14.7.7

8. Aus dem Fahrplan der Bundesbahn ist ersichtlich welche Zeit man zwischen den Orten Regensburg (R), Nürnberg (N) und München (M) benötigt.

von	über	nach	Abfahrt	Ankunft	Stunden
N	R	M	14.01	18.01	4
R	M	N	14.10	18.10	4
N	M	R	14.21	18.21	4
R	N	M	15.12	18.12	3

Berechnen Sie welche Zeit man für die Strecke von Regensburg nach München benötigt.

Lösung 14.7.8

9. Ein Weinhändler verkauft die Hälfte seines Lagervorrats an Sekt und den gesamten Weißwein an A, insgesamt 70 Flaschen. Die andere Hälfte des Sektes und den ganzen Rotweinbestand verkauft er an B; das sind insgesamt 80 Flaschen. Anschließend will der Weinhändler die Verkaufszahlen so genau wie möglich haben; er weiß jedoch nur noch, dass er von jeder Sorte ca. 50 Flaschen hatte. Helfen Sie ihm.

Lösung 14.7.9

10. Durch zwei Messungen wurde festgestellt, dass 6 50-Pfenningstücke, 14 Markstücke, 8 Zweimarkstücke und 2 Fünfer zusammen 120g wiegen und dass 3 50-Pfenningstücke, 1 Markstück, 16 Zweimarkstücke und 3 Fünfer zusammen auch 120g wiegen. Nun soll das Gewicht jeder einzelnen Münze berechnet werden. Dazu wird angenommen, dass das Gewicht proportional mit dem Wert der Münzen zusammenhängt.

Lösung 14.7.10

14.8 Iteration

1. Es sei das lineare Gleichungssystem $\mathbf{Ax} = \mathbf{b}$ mit

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 8 \\ -5 \end{bmatrix}$$

Berechnen Sie $\mathbf{x}$ mit dem Gauß-Jacobi-Verfahren und dem Gauß-Seidel-Verfahren bis zur 8. Näherung. Der Startvektor ist der Nullvektor.

Lösung 14.8.1

2. Gegeben sind die Gleichungen

$$\begin{array}{rcrcrcrcrcrcr} 10x_1 & - & x_2 & & & & & & & & & = & 10 \\ -x_1 & + & 10x_2 & - & x_3 & & & & & & & = & 0 \\ & - & x_2 & + & 10x_3 & - & x_4 & & & & & = & 10 \\ & & & - & x_3 & + & 10x_4 & - & x_5 & & & = & 0 \\ & & & & & - & x_4 & + & 10x_5 & - & x_6 & = & 10 \\ & & & & & & & - & x_5 & + & 10x_6 & = & 10 \end{array}$$

Berechnen Sie den Vektor $\mathbf{x}$ bis zur 6. Näherung mit dem Gesamtschrittverfahren.

Lösung 14.8.2

14.9 Kleinste Quadrate

1. Finden Sie eine Lösung des linearen Systems mit der Methode der kleinsten Quadrate. (d.h. $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$).

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & -1 & 1 & -1 \\ 1 & -2 & 4 & -8 \\ 1 & 3 & 9 & 27 \\ 1 & -3 & 9 & -27 \end{bmatrix} \cdot \mathbf{x} = \begin{pmatrix} -2 \\ -7 \\ 8 \\ 1 \\ -4 \\ -22 \end{pmatrix}$$

Lösung 14.9.1

2. Berechnen Sie die Lösung der Gleichung $\mathbf{Ax} = \mathbf{b}$ mit der Methode der kleinsten Quadrate.

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -3 \\ -1 \\ 0 \\ 2 \\ 5 \\ 1 \end{bmatrix}$$

Lösung 14.9.2

3. Eine Portion (28g) Kellogg's Crunchy Nuts enthält 110 Kalorien, 3g Protein, 21g Kohlenhydrate und 3g Fett. Eine Portion Kellogg`s Crispis enthält 110 Kalorien, 2g Protein, 25g Kohlenhydrate und 0.4g Fett.

   a) Stellen Sie eine Matrix $\mathbf{B}$ und einen Vektor $\mathbf{u}$ auf, so daß $\mathbf{Bu}$ die Summe der Kalorien, Proteine, Kohlenhydrate und Fett ergibt, die in einer Mischung von 3 Portionen Crunchy Nuts und 2 Portionen Crispis enthalten sind.

   b) Sie wollen ein Frühstück mit mehr Protein als in Crispis aber weniger Fett als in Crunchy Nuts. Ist es möglich eine Mischung der beiden Kellogg`s-Sorten zusammenzustellen, die 110g Kalorien, 2.25g Protein, 24g Kohlenhydrate und 1g Fett enthält? Wenn ja, wie lautet die Mischung?

Lösung 14.9.3

4. Aus den gemessenen Molekulargewichten

NO	N_2O	NO_2	N_2O_3	N_2O_5	N_2O_4
30.006	44.013	46.006	76.012	108.010	92.011

Sollen die Atomgewichte von Stickstoff und Sauerstoff bestimmt werden.

Lösung 14.9.4

5. Berechnen Sie die x_i so, daß die Kurve $p(t) = x_1 \sin t + x_2 \cos t + x_3 \sin 2t + x_4 \cos 2t$ die beste Näherung für die Punkte (-2, -4), (-1, -4), (0, -2), (1, 2), (2, 8), (3, 16) ist. Zeichnen Sie das Ergebnis und die Punkte in das gleiche Koordinatensystem.

Lösung 14.9.5

6.* Nach Kepler`s erstem Gesetz hat ein Komet entweder eine Ellipse, eine Parabel oder eine Hyperbel als Umlaufbahn (Anziehungen anderer Planeten aufgrund der Gravitation werden ignoriert). Die Position (r, ϑ) in Polarkoordinaten kann mit der Formel $r = \beta + e(r \cdot \cos \vartheta)$ berechnet werden, wobei β eine Konstante und e die Exzentrität der Umlaufbahn ist: $0 \le e \le 1$ bei einer Ellipse, $e = 1$ bei einer Parabel und $e > 1$ bei einer Hyperbel. Beobachtungen eines neu entdeckten Kometen liefern die folgenden Daten:

ϑ	0.88	1.10	1.42	1.77	2.14
r	3.00	2.30	1.65	1.25	1.01

Bestimmen Sie den Typ der Umlaufbahn und sagen Sie vorher wo der Komet bei $\vartheta = 4.6$ sein wird. Zeichnen Sie die Umlaufbahn.
Hinweis: Die Formel muß zunächst linearisiert werden.

Lösung 14.9.6

14.10 Eigenwerte

1.► Überprüfen Sie ob $\mathbf{x}$ oder $\mathbf{y}$ Eigenvektoren der Matrix $\mathbf{A}$ sind. Berechnen sie dann den zugehörigen Eigenwert.

$$\mathbf{A}=\begin{bmatrix}2&1&1\\2&3&2\\1&1&2\end{bmatrix} \qquad \mathbf{x}=\begin{bmatrix}-1\\1\\0\end{bmatrix} \qquad \mathbf{y}=\begin{bmatrix}-1\\0\\1\end{bmatrix}$$

Lösung 14.10.1

2.► Gegeben seien die Matrizen $\mathbf{A}=\begin{bmatrix}2&16\\1&-4\end{bmatrix}$ und $\mathbf{B}=\frac{1}{3}\begin{bmatrix}3&2&2\\2&2&0\\2&0&4\end{bmatrix}$.

Ermitteln Sie ihre Eigenwerte und geben Sie jeweils einen dazugehörigen normierten Eigenvektor an.

Lösung 14.10.2

3.► Bestimmen Sie für die folgende Matrix die Eigenwerte und die zu den jeweiligen Eigenwerte gehörenden Eigenräume. Ist die Matrix diagonalähnlich?

$$\mathbf{A}=\begin{bmatrix}-3&1&-1\\-7&5&-1\\-6&6&-2\end{bmatrix}$$

Lösung 14.10.3

4. Finden Sie eine 4x4-Matrix, die die Eigenwerte 2, -3, 0 und 3 hat und deren zugehörige Eigenvektoren folgende sind:

$$\begin{bmatrix}1\\2\\3\\-1\end{bmatrix},\begin{bmatrix}4\\2\\-1\\0\end{bmatrix},\begin{bmatrix}3\\1\\0\\0\end{bmatrix},\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

Lösung 14.10.4

5. Berechnen Sie das charakteristische Polynom, die Eigenwerte und wenn möglich eine reguläre Diagonalisierungsmatrix.

$$\mathbf{A}=\begin{bmatrix}2&-1&0&0\\-1&2&-1&0\\0&-1&2&-1\\0&0&-1&2\end{bmatrix} \qquad \mathbf{B}=\begin{bmatrix}2&1&-1&2\\1&3&2&-3\\-1&2&1&-1\\2&-3&-1&4\end{bmatrix}$$

Lösung 14.10.5

6. Berechnen Sie die Eigenwerte der folgenden Matrizen. Finden Sie in jedem Fall auch die Matrix $\mathbf{P}$ der Eigenvektoren und berechnen Sie $\mathbf{P}^{-1}\mathbf{AP}$.

$$\mathbf{A} = \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 5 & 2 & 0 & 0 & 0 \\ 2 & -4 & 3 & 0 & 0 \\ 0 & 3 & 0 & 6 & 0 \\ 0 & 0 & 6 & 4 & 1 \\ 0 & 0 & 0 & 1 & 5 \end{bmatrix}$$

Lösung 14.10.6

7. Bestimmen Sie die Eigenwerte und die Eigenvektoren der folgenden Matrix.

$$\mathbf{B} = \begin{bmatrix} -4 & -4 & 20 & -8 & -1 \\ 14 & 12 & -46 & 18 & 2 \\ 6 & 4 & -18 & 8 & 1 \\ 11 & 7 & -37 & 17 & 2 \\ 18 & 12 & -60 & 24 & 5 \end{bmatrix}$$

Lösung 14.10.7

8. Diagonalisieren Sie die Matrix $\mathbf{A}$, d.h. finden Sie Matrizen $\mathbf{V}$ und $\mathbf{D}$ mit $\mathbf{A} = \mathbf{VDV}^{-1}$ aus den Eigenwerten und Eigenvektoren von $\mathbf{A}$.

$$\mathbf{A} = \begin{bmatrix} 11 & -6 & 4 & -10 & -4 \\ -3 & 5 & -2 & 4 & 1 \\ -8 & 12 & -3 & 12 & 4 \\ 1 & 6 & -2 & 3 & -1 \\ 8 & -18 & 8 & -14 & -1 \end{bmatrix}$$

Lösung 14.10.8

9.► Ermitteln Sie alle Eigenwerte von $\mathbf{A}$ und überprüfen Sie, ob $\mathbf{A}$ diagonalähnlich ist.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 2 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 3 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 4 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 5 & 6 & 6 \\ 7 & 7 & 7 & 7 & 7 & 7 & 6 & 7 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 \end{bmatrix}$$

Lösung 14.10.9

10. Gegeben ist die Matrix $\mathbf{A} = \begin{bmatrix} -2 & 0 & 6 & -6 \\ -5 & 3 & 5 & -5 \\ -1 & 1 & 5 & -3 \\ -1 & 1 & 1 & 1 \end{bmatrix}$.

Bestimmen Sie die Eigenwerte und dazugehörigen Eigenvektoren der folgenden Matrizen.

a) $\mathbf{A}$

b) $\mathbf{A}^{-1}$

c) $\mathbf{A}^2 = \mathbf{A} \cdot \mathbf{A}$

d) $\mathbf{A} - 0.5 \cdot \mathbf{I}$

e) $\mathbf{A}^2 + 3 \cdot \mathbf{A} - 4 \cdot \mathbf{I}$

f) $\mathbf{A}^2 - 3 \cdot \mathbf{A} + 4 \cdot \mathbf{I}$

Die Ergebnisse geben Ihnen sicher Anlass zu einigen Vermutungen.

Lösung 14.10.10

14.11 Komplexe Zahlen

1.► Berechnen Sie i^n für n = 1, 2, ..., 10

Lösung 14.11.1

2.► Bestimmen Sie alle Lösungen der Gleichung $iz^2 - (2-i)z + (1+3i) = 0$

Lösung 14.11.2

3. Lösen Sie die Gleichungen

a) $z^4 - 1 = 0$

b) $z^4 + 1 = 0$

c) $z^5 = 32i$

Lösung 14.11.3

4. Geben Sie eine Lösung der Gleichung $z^i = 1 + i$ an!

Lösung 14.11.4

5.► Berechnen Sie für die komplexen Zahlen $z = 3 + 2i$ und $w = 4 - 5i$ die folgenden Werte:

a) $z + w$

b) $2z - 3w$

c) $z + \overline{w}$

d) $\overline{z} + \overline{w}$

e) $\overline{zw}$

f) $\overline{z} \cdot \overline{w}$

g) $\overline{z} z$

h) $\begin{bmatrix} \overline{z} \\ \overline{w} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} z \\ w \end{bmatrix}$

Lösung 14.11.5

6. Überprüfen Sie die folgenden Aussagen

a) $e^{z1} \cdot e^{z2} = e^{z1+z2}$

b) $e^{i \cdot z1} = \cos(z1) + i \cdot \sin(z1)$

Lösung 14.11.6

7. Berechnen Sie mit der MATLAB-Funktion ***eig*** die Eigenwerte der Matrix

$$\mathbf{A} = \begin{bmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{bmatrix}$$

Ändern Sie das Matrixelement a_{22} zu 180.01 bzw. 179.99 und berechnen Sie erneut die Eigenwerte.

Lösung 14.11.7

8. Das charakteristische Polynom der Matrix $\mathbf{A}$ lautet: $\lambda^4 + 5\lambda^3 + 3\lambda^2 - 7\lambda + 11$ und das der Matrix $\mathbf{B}$ lautet: $\lambda^5 + 3\lambda^4 - 5\lambda^2 - 11\lambda + 5$. Berechnen Sie die Determinante und die Spur der beiden Matrizen.

Lösung 14.11.8

9. Bestimmen Sie die Eigenwerte und die Eigenvektoren der folgenden Matrix.

$$\mathbf{A} = \begin{bmatrix} 4 & -9 & -7 & 8 & 2 \\ -7 & -9 & 0 & 7 & 14 \\ 5 & 10 & 5 & -5 & -10 \\ -2 & 3 & 7 & 0 & 4 \\ -3 & -13 & -7 & 10 & 11 \end{bmatrix}$$

Lösung 14.11.9

10. Erzeugen Sie eine 4x4-Zufallsmatrix $\mathbf{A}$ und zeigen Sie, daß $\mathbf{A}$ und $\mathbf{A}^{\mathrm{T}}$ das gleiche charakteristische Polynom haben (die gleichen Eigenwerte). Haben Sie auch die gleichen Eigenvektoren? Überprüfen Sie die Annahmen auch bei größeren Zufallsmatrizen.

Lösung 14.11.10

11. Erzeugen Sie eine 6x6-Zufallsmatrix $\mathbf{A}$. Berechnen Sie $\det(\mathbf{A})$ und die Eigenwerte von $\mathbf{A}$. Vergleichen Sie den Wert der Determinante mit dem Produkt der Eigenwerte.

Lösung 14.11.11

14.12* Programmieraufgaben

1. Matrizenrechnung und lineare Gleichungssysteme

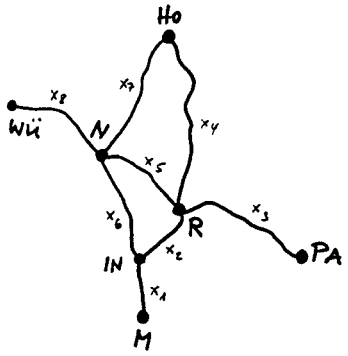
a) Vier Kurierdienste mit Sitz in Würzburg (WÜ), Regensburg (R), Hof (HO) und München (M) befördern eilige Sendungen zwischen verschiedenen Städten und zwar:

WÜ $\Rightarrow$ HO, N, M, PA (N = Nürnberg)

R $\Rightarrow$ HO, N, IN, PA (PA = Passau)

HO $\Rightarrow$ R, N (IN = Ingolstadt)

M $\Rightarrow$ WÜ, N, R, IN



Die Firmen wollen künftig zusammenarbeiten und gegenseitig Fracht übernehmen, damit jeder Kurierdienst jede der genannten Städte schnell beliefern kann. Um mit einem einheitlichen Preisniveau arbeiten zu können, sollen die Sendung nach durchschnittlicher Fahrtdauer und Gewicht berechnet werden. Für die Fahrtzeiten werden von den Firmen folgende Angaben (in Minuten) gemacht:

WÜ $\Rightarrow$ HO: 130	WÜ $\Rightarrow$ N: 100	WÜ $\Rightarrow$ M: 160	WÜ $\Rightarrow$ PA: 180
R $\Rightarrow$ HO: 125	R $\Rightarrow$ N: 90	R $\Rightarrow$ IN: 40	R $\Rightarrow$ PA: 70
HO $\Rightarrow$ R: 130	HO $\Rightarrow$ N: 80		
M $\Rightarrow$ WÜ: 150	M $\Rightarrow$ N: 100	M $\Rightarrow$ R: 80	M $\Rightarrow$ IN: 40

Unter Beachtung des skizzierten Streckennetzes kann man aus diesen 14 Angaben lineare Gleichungen für die Fahrtzeiten $x_1, x_2, ..., x_8$ auf den skizzierten Strecken aufstellen. So entsteht ein lineares Gleichungssystem $\mathbf{Bx} = \mathbf{b}$, das überbestimmt ist. Also muss eine Näherungslösung $\mathbf{x}$ gefunden werden, derart dass $|\mathbf{Bx} - \mathbf{b}| = \min$ wird. Berechnen Sie diese Lösung $\mathbf{x}$.

b) Die Städte aus Aufgabe 1 seien alphabetisch durchnumeriert: 1=HO, 2=IN, 3=M, 4=N, 5=PA, 6=R, 7=WÜ. Geben Sie die Bewertungsmatrix $\mathbf{C}$ aus, in deren Komponente $c_{m,n}$ die Fahrtdauer zwischen Stadt m und Stadt n steht.
Beispiel: München-Regensburg: $c_{3,6} = x_1 + x_2$
Beachten Sie, dass es von Hof nach Ingolstadt 2 Strecken gibt. Die kürzere von beiden ist in $\mathbf{C}$ einzutragen. Verwenden Sie an dieser Stelle die MATLAB-Funktion ***min*** und beachten Sie, dass diese nur auf Spalten wirkt.
Die Matrix $\mathbf{C}$ ist symmetrisch. Es genügt deshalb, die Komponenten im rechten oberen Dreieck von $\mathbf{C}$ zu berechnen und dann $\mathbf{C}$ mit der Transponierten $\mathbf{C'}$ zusammenzusetzen.

c) Die MATLAB-Funktion ***gplot*** zeichnet einen Graphen. Dazu wird die Adjazenzmatrix $\mathbf{A}$ des Graphen benötigt, deren Komponenten $a_{m,n} = 1$ sind, wenn Knoten m mit Knoten n direkt verbunden ist, andernfalls ist $a_{m,n} = 0$. Zweiter Parameter von ***gplot*** ist eine 2-

spaltige Matrix $\mathbf{P}$, die die xy-Koordinaten der Knoten enthält. Als dritten Parameter kann man ein Plot-Symbol angeben. Informieren Sie sich mit ***help gplot***.
Unsere 7 Städte haben die Koordinaten:

	HO	IN	M	N	PA	R	WÜ
Längengrad	11,9	11,4	11,5	11,1	13,3	12,1	9,9
Breitengrad	50,3	48,8	48,2	49,4	48,6	49,0	49,8

Zeichnen Sie das Straßennetz als Graph, die Städte darin als kleine Kreise, die Verbindungen als blaue Linien. Sie benötigen dazu zwei Aufrufe von ***gplot*** mit unterschiedliche Plotsymbolen. Dazwischen muss ***hold on*** stehen, damit das erste Bild nicht gelöscht wird.
Beachten Sie, dass die Längengrade ca. 75 km entsprechen, Breitengrade dagegen ca. 111 km. Um diesen Unterschied auszugleichen, sind alle Tabellenwerte in Kilometer umzurechnen, bevor sie in die Matrix $\mathbf{P}$ übernommen werden. Außerdem muss nach den gplot-Befehlen ***axis equal*** angegeben werden, damit beide Achsen mit gleichen Einheiten skaliert erscheinen. Mit Hilfe der Funktion ***text*** können Sie das Bild mit den Ortsnamen beschriften.

d) Die Inzidenzmatrix $\mathbf{I}$ eines Graphen kennzeichnet, welche Wege die einzelnen Knoten verbinden. Ihre Zeilennummern entsprechen den Knotennummern, ihre Spaltennummern den Nummern der Wege (vergl. Skizze). Die Elemente von $\mathbf{I}$ sind fast alle = 0. Es gilt nur $i_{mn} = 1$, falls der Weg Nr. n im Knoten m beginnt und $i_{mn} = -1$, falls er dort endet. Folglich gibt es in jeder Spalte von $\mathbf{I}$ genau eine 1 und eine –1. Stellen Sie die Inzidenzmatrix $\mathbf{I}$ zu unserem Graphen auf.
Wir wollen die Entfernungskilometer zwischen unseren Orten in Luftlinie messen und diese Werte den entsprechenden Wegen zuordnen. Dazu benötigen wir die Angaben aus der Matrix $\mathbf{P}$ (s. Aufgabenteil c). Zunächst wird mit der MATLAB-Funktion ***find*** anhand $\mathbf{I}$ festgestellt, welche Knoten jeder Weg verbindet. Speichern Sie deren Nummern in einer zweizeiligen Matrix $\mathbf{J}$. Jede Spalte k von $\mathbf{J}$ enthält damit die beiden Nummern der Endknoten des Weges k. Die Luftlinie ergibt sich nun für jeden Weg k als Abstand der Knoten $j_{1,k}$ und $j_{2,k}$:

$$d_k = \sqrt{(\mathbf{P}_{j_{1,k},1} - \mathbf{P}_{j_{2,k},1})^2 + (\mathbf{P}_{j_{1,k},2} - \mathbf{P}_{j_{2,k},2})^2}$$

Berechnen Sie diese Luftlinien und geben Sie die Entfernungen aus.
Um die Zahlenwerte auch im vorhandenen Bild mit Hilfe der Funktion ***text*** einzubringen, ist ihre vorherige Konversion in Zeichenketten nötig. Das kann z.B. mit Hilfe der Funktion ***sprintf*** bewerkstelligt werden, die nach dem Vorbild der entsprechenden C-Funktion arbeitet (vergl. ***help sprintf***). Als geeignete Position zur Anbringung der Entfernungsangabe empfiehlt sich der Mittelpunkt zwischen den Endknoten des betreffenden Weges.
Übrigens: Formatierte Ein- und Ausgabe funktioniert in MATLAB analog zu den entsprechenden C-Anweisungen ***fscanf***, ***fprintf***.

Lösung 14.12.1

2. Zufallszahlen, Grundbegriffe aus der linearen Algebra und Statistik

a) Wir wollen mit quadratischen Matrizen arbeiten. Geben Sie in Ihrem MATLAB-Code zunächst die Zahl N = 100 ein, um eine ansehnliche Matrix-Größe zu erhalten.

b) Erzeugen Sie 2 NxN-Matrizen $\mathbf{A}$ und $\mathbf{B}$, mit ganzzahligen Zufallszahlen. Die Zahlen in $\mathbf{A}$ sollen gleichverteilt sein im Intervall $[0,10)$, die in $\mathbf{B}$ sollen normalverteilt sein mit Mittelwert 0 und Standardabweichung 10. Sie benötigen dazu die MATLAB-Funktionen ***rand***, ***randn*** sowie ***fix*** und ***round***. Bei der Berechnung von $\mathbf{A}$ wird ***fix*** benötigt, bei $\mathbf{B}$ ***round***. Warum?

c) Geben Sie das minimale und das maximale Element jeder dieser beiden Matrizen aus (MATLAB-Funktionen ***max*** und ***min*** richtig anwenden). - Die Ergebnisse sollen amin, amax, bmin und bmax heißen.

d) Stellen Sie fest, wie oft jedes Matrixelement vorkommt und speichern Sie diese Häufigkeiten in Vektoren $\mathbf{a}$ bzw. $\mathbf{b}$. Für diese Aufgabe eignen sich besonders die MATLAB-Funktionen ***find*** und ***length***. Außerdem ist es zweckmäßig, mit einer for-Schleife zu arbeiten.

e) Zeichnen Sie die beiden Häufigkeitsverteilungen $\mathbf{a}$ und $\mathbf{b}$ in 2 getrennten Bildern mit Hilfe der MATLAB-Funktionen ***plot*** und ***figure***. Achten Sie auch auf die richtige Skalierung der x-Achse. Verwenden Sie als Plotsymbol für $\mathbf{b}$ einen Punkt, für $\mathbf{a}$ einen Stern.

f) Da die Daten in $\mathbf{A}$ gleichverteilt sind, sollte in allen Komponenten von $\mathbf{a}$ immer in etwa die gleiche Zahl stehen. Zeichnen Sie im Bild 1 den Mittelwert aller Komponenten von $\mathbf{a}$ als horizontale schwarze Linie ein. Um diese Linie müssen die Werte gleichmäßig streuen. MATLAB: ***hold on*** verwenden, um das Bild nicht zu zerstören.

g) Die Normalverteilung der Komponenten von $\mathbf{B}$ sollte sich im Bild 2 dadurch äußern, dass die meisten Werte in der Nähe von Null liegen und die Häufigkeiten nach außen rasch abnehmen. Mit der Standardabweichung s = 10, die wir oben eingeführt haben, wird der Kurvenverlauf von $\mathbf{b}$ in Bild 2 durch folgende Funktion charakterisiert:

$$f(t) = \frac{N^2}{s\sqrt{2\pi}} e^{\frac{-t^2}{2s^2}}$$

Zeichnen Sie diese Funktion ins vorhandene Bild als rote Linie ein. Verwenden Sie für t ein Raster mit Zehntel-Schritten zwischen den Endpunkten bmin und bmax der horizontalen Achse.

h) Prüfen Sie folgende Behauptungen durch MATLAB-Anweisungen nach:

i) 68,3% aller Elemente von $\mathbf{B}$ liegen im Intervall [-s, s],

ii) 95,5% aller Elemente von $\mathbf{B}$ liegen im Intervall [-2s, 2s],

iii) 99,7% aller Elemente von $\mathbf{B}$ liegen im Intervall [-3s, 3s].

i) Berechnen Sie die folgenden Matrizen (. bedeutet „elementweise“):

$$\mathbf{C} = \mathbf{AB} + \mathbf{BA}, \qquad \mathbf{D} = \mathbf{A.B} + \mathbf{B.A}, \qquad \mathbf{E} = \mathbf{A}^{\mathbf{T}}\mathbf{B} + \mathbf{B}^{\mathbf{T}}\mathbf{A}, \qquad \mathbf{F} = \mathbf{A}^{\mathbf{T}}\mathbf{A} + \mathbf{B}^{\mathbf{T}}\mathbf{B}$$

j) Berechnen Sie die Ränge und Determinanten dieser 4 Matrizen und geben Sie diese 8 Zahlen in Form von 2 Zeilenvektoren aus.

k) Stellen Sie fest, welche der 4 Matrizen symmetrisch sind. Benutzen Sie dazu die MATLAB-Funktion ***all*** oder ***any***.

Lösung 14.12.2

3. Graphiken, Polynome, Wertetabellen, Approximation

Die Datei pinunze.4u enthält den Kursverlauf in € einer Aktie namens "Pinunze" in den ersten 288 Tagen des Jahres 2011. Sie sollen damit einige Analysen durchführen. Im einzelnen ist folgendes zu tun:

a) Einlesen der Daten und speichern der Werte in 2 Vektoren **t** (Tag) und **w** (Wert).

b) Graphische Ausgabe des Kursverlaufs als blaue Kurve in Figur 1.

c) Die optimale Approximationsgerade "durch" die Kurvenpunkte konstruieren.

d) Zeichnen dieser Geraden als rote gestrichelte Linie ins vorhandene Bild. Die Linie soll bis zum Jahresende (365 Tage) reichen. Der Endpunkt soll als roter Sheriffstern (Pentagon) markiert werden, um die Kurs-Prognose zu verdeutlichen.

e) Das optimale Polynom 3. Grades durch die Kurvenpunkte konstruieren und als grüne gestrichelte Linie bis zum Jahresende einzeichnen. Markierung der Prognose zum Jahresende durch ein grünes Pentagon.

f) Für die Jahresendprognose wollen wir ein Polynom 2. Grades verwenden. Zeichnen Sie seinen Jahresendwert als blauen Stern ins vorhandene Bild.

Lösung 14.12.3

4. Interpolation mit Polynomen und Exponentialfunktion

Die Weltbevölkerung hat sich entwickelt wie in der nebenstehenden Tabelle gezeigt, die aus einem Nachschlagewerk entnommen ist. In der unteren Zeile steht die Bevölkerungszahl in Milliarden, d.h. 10^9 oder "Giga" im EDV-Jargon.

Jahr	1.650	1.750	1.800	1.850	1.900	1.960	1.975	2.000
Mrd.	0,545	0,728	0,906	1,171	1,608	2,995	3,838	6,272

Wir wollen einige Untersuchungen mit diesen Daten anstellen. Um nicht mit übergroßen Zahlen arbeiten zu müssen empfiehlt es sich für die Rechnung, auch die Jahreszahlen durch 1000 zu dividieren, wie in der Tabelle durch Punkte angedeutet.

a) Speichern Sie die beiden Spalten der Tabelle in 2 Vektoren **x** und **y**. Geben Sie die Daten als blaue Kringel in einem halblogarithmischen Plot aus. Dazu dient die MATLAB-Routine ***semilogy***. Sie funktioniert ansonsten wie ***plot***. Geben Sie mit ***title*** der Graphik den Titel "Weltbevölkerung".

b) Im folgenden werden uns die Jahre von 1.600 bis 2.100 interessieren. Speichern Sie diese Zahlen in einem Vektor **t** (z.B. mit ***linspace***).

c) Berechnen Sie das Interpolationspolynom p(t) (Grad beachten!) durch alle Knotenpunkte der Tabelle. Speichern Sie seine Koeffizienten in einem Vektor **a**. Berechnen Sie p(t) für die im Vektor **t** gespeicherten Jahre und zeichnen Sie damit p(t) als blaue Kurve ins vorhandene Bild. - Man erkennt ein Fehlverhalten der Funktion für Daten aus der Vergangenheit: Die Bevölkerungszahl steigt für zurückliegende Jahre vor 1650!

d) Eine weitere Möglichkeit wäre die Darstellung der Bevölkerungsentwicklung durch eine Exponentialfunktion der Form $e^{q(t)}$, worin q(t) ein Polynom gleichen Grades wie p(t) ist. Berechnen Sie seine Koeffizienten und speichern Sie diese in einem Vektor **b**. Berechnen Sie $e^{q(t)}$ für die im Vektor **t** gespeicherten Jahre und zeichnen Sie damit $e^{q(t)}$ als

grüne Kurve ins vorhandene Bild. - Man erkennt in der Vergangenheit wieder das Fehlverhalten der Funktion. Zusätzlich deutet sich in der Zukunft der Untergang der Menschheit an.

Lösung 14.12.4

5. Funktion-m-Files, inline-Funktionen, Integrale berechnen, Extremwertbestimmung

Als Beispielfunktion wollen wir die "Fakultätsfunktion" f(x) = x! betrachten. Auf den meisten Taschenrechnern ist diese Funktion nur für ganzzahlige x verfügbar. In MATLAB wird die verwandte Funktion gamma(x + 1) = x! als Standardfunktion bereitgestellt.

a) Definieren Sie die Fakultätsfunktion in einem function-m-File fakultaet.m als y = gamma (x+ 1).

b) Zeichnen Sie die Funktionen f (x) = x! rot und g(x) = 1 / x! magentafarben für $x \in [-\pi, \pi]$. Verwenden Sie dazu ***linspace*** mit 501 Rasterpunkten.

c) Schränken Sie mit ***axis*** das Bildfenster ein auf den Bereich $x \in [-\pi, \pi],\ y \in [-5,7]$ und zeichnen Sie ins vorhandene Bild die x- und y-Achsen als schwarze Linien sowie den Nullpunkt als schwarzen Kringel.

d) Verwenden Sie die MATLAB-Funktion ***quad8***, um das Integral $I_1 = \int_0^3 x! dx$ zu berechnen. Als erster Parameter ist der Funktionsname ‘ fakultaet ‘ einzutragen.

e) Berechnen Sie das Integral $I_2 = \int_{-3}^{-2} \frac{dx}{x!}$ mit Hilfe eines ***inline***-Aufrufs in ***quad8*** oder mit Hilfe eines zweiten function-m-Files für die Funktion g(x) = 1 / x!.

f) Berechnen Sie die 3 im Bild erkennbaren Extremwerte der Funktion f (x) = x! und markieren Sie sie durch blaue Sterne. Beachten Sie, dass MATLAB hierzu nur die Funktion ***fmins*** anbietet. Um ein Maximum zu bestimmen, sucht man deshalb ein Minimum der negativen Funktion. Programmiertechnisch hilft man sich hierfür zweckmäßig wieder mit ***inline*** oder einem weiteren function-m-File.

Lösung 14.12.5