MATLAB Kurzanleitung

Otto–von–Guericke–Universität
Institut für Automatisierungstechnik

11. April 1995

1 Einführung

MATLAB ist ein interaktives, matrixorientiertes Programmsystem für wissenschaftlich-technische Berechnungen. Die Lösung selbst komplexer numerischer Probleme ist meist ohne das Schreiben eigener Programme in traditionellen Hochsprachen möglich. MATLAB wurde ursprünglich entwickelt, um einfachen Zugriff auf die Algorithmen der Softwareprojekte LINPACK und EISPACK zu haben. Diese beiden Projekte repräsentieren den Standard bezüglich Software für Matrixberechnungen. MATLAB wird seit einer Reihe von Jahren ständig weiterentwickelt. Der Name MATLAB ist eine Abkürzung für MATrix LABoratory.

Diese Anleitung ist als Hilfe bei den ersten Schritten mit MATLAB gedacht. Der Leser sei dazu ermutigt, die hier angegebenen Beispiele gleich am Rechner zu erproben.

Diese Beschreibung orientiert sich an der Version 4.0 von MATLAB für Microsoft Windows 3.1. Ältere Versionen, wie die weit verbreitete Version 3.5 für MS-DOS, unterscheiden sich nur gering in der Handhabung.

Für detailliertere Informationen zu den einzelnen Kommandos wird die Nutzung der on-line-Hilfe empfohlen. Nach dem Aufruf von MATLAB, wie im Abschnitt 2 beschrieben, kann durch Eingabe des Befehls `help` eine Übersicht zu den vorhandenen Befehlsgruppen abgerufen werden. Die zu den Befehlsgruppen gehörenden Befehle sind durch `help <befehlsgruppe>` abrufbar. Möchte man nur zu einem speziellen Befehl den Hilfetext lesen, dann ist einfach `help <befehl>` einzugeben. Das Kommando `help eig` gibt zum Beispiel die Übersicht zur Funktion `eig` von MATLAB aus. Einen Eindruck von den Möglichkeiten, die MATLAB bietet, erhält man nach Eingabe des Befehls `demo`.

Die Fähigkeiten von MATLAB sind weitaus größer, als sie in den folgenden Seiten dargestellt werden können. Für weitergehende Informationen sei hier auf die Handbücher zu MATLAB verwiesen. Sie sind für gewöhnlich in den Rechnerräumen der Universitäten oder der einzelnen Institute zu finden.

Inhaltsverzeichnis

2 Aufruf von MATLAB

Nach dem Start von Microsoft Windows befindet sich das Programm gewöhnlich im Ordner MATLAB. Durch Doppelklick auf das entsprechende Symbol wird MATLAB aufgerufen. Danach meldet MATLAB seine Bereitschaft durch die Ausgabe des Kommando-Prompts in Form eines $\gg$ -Zeichens. Verlassen kann man MATLAB durch die Befehle `quit` oder `exit`.

2.1 Bearbeiten der Kommandozeile

Die aktuelle Kommandozeile ist entsprechend der folgenden Tastenkombinationen zu editieren:

↑,Ctrl-P	Aufrufen der vorherigen Kommandozeile
↓,Ctrl-N	Aufrufen der nächsten Kommandozeile
←,Ctrl-N	Ein Zeichen nach links
→,Ctrl-N	Ein Zeichen nach rechts
Delete	Rechts ein Zeichen löschen
←	Links ein Zeichen löschen
Ctrl-L	Ein Wort nach links
Ctrl-R	Ein Wort nach rechts
Ctrl-A	An den Zeilenanfang
Ctrl-E	An das Zeilenende
Ctrl-U	Zeile löschen
Ctrl-K	Bis zum Ende der Zeile löschen

Abgeschlossen wird die Zeile mit Enter. Die enthaltenen Funktionen werden ausgeführt. Wenn die Zeile einen Tipfehler enthält, gibt Matlab eine Fehlermeldung aus. Zum Beispiel, wenn sqrt in der Kommandozeile

```
log(sqt(atan(2*(3+4))))
```

falsch geschrieben wurde bringt Matlab die Fehlermeldung:

```
??? Undefined function or variable sqt.
```

Die Zeile kann mit der ↑ zurückgeholt und korrigiert werden. Die Kommandozeilen die während einer Matlabsitzung mit Enter abgeschlossen wurden sind, werden in einem Puffer gespeichert. Dies ermöglicht es, den selben Befehl mit veränderten Parametern zu beliebigen Zeitpunkten zu wiederholen. Zu erwähnen ist noch die Möglichkeit des gezielten Zurückrufens von Zeilen, durch plo↑ wird die Zeile zurückgerufen, deren erste Buchstaben plo sind.

2.2 Installationsprobleme

Nach der vollständigen Installation ist MATLAB sofort einsatzbereit. Alle Voreinstellungen wurden vom Installationsprogramm vorgenommen. Bei der Arbeit mit MATLAB kann es jedoch vorkommen, daß man zusätzliche MATLAB-Bibliotheken einrichten möchte oder daß die Struktur der Verzeichnisse auf der Festplatte geändert werden muß. In solchen Fällen sind lediglich die Suchpfade von MATLAB so zu ergänzen oder zu ändern, daß alle Toolboxen im Suchpfad stehen. Dazu ist die Datei *matlabrc.m* mit einem einfachen ASCII-Editor entsprechend abzuändern. Alternativ kann auch die Datei *startup.m*, welche beim Aufruf von MATLAB automatisch abgearbeitet wird, um entsprechende Befehle ergänzt werden (`help matlabpath`).

3 Beenden einer Matlabsitzung

Matlab kann über die Befehle `quit` und `exit` beendet werden. Variabeln die während der Matlabsitzung definiert wurden, werden aus dem Arbeitsspeicher entfernt. mit dem Befehl `save` können sie für spätere Sitzungen abgespeichert werden. Die Kommandozeile

```
save temp X Y Z
```

speichert die Variabeln `X Y Z` in dem File `temp.mat` ab, wird in einer späteren Sitzung `load temp` aufgerufen, so stehen die Variabeln wieder zur Verfügung, ohne erneut definiert werden zu müssen.

4 Anweisungszeilen, Ausdrücke und Variablen

Anweisungszeilen haben in MATLAB gewöhnlich die Form:

Variable = *Ausdruck* ,

oder

Ausdruck,

wenn das Ergebnis keiner Variablen zugewiesen werden muß. Ausdrücke setzen sich aus Operatoren, Funktionen und Variablen zusammen. Die Berechnung eines Ausdrucks erzeugt eine Matrix, welche entweder einer Variablen zugewiesen wird oder auf dem Bildschirm erscheint. Eine Anweisungszeile wird normalerweise mit einem RETURN abgeschlossen. Soll eine Anweisungszeile auf der nächsten Bildschirmzeile fortgesetzt werden, kann dies durch Angabe von drei Punkten am Ende der Zeile geschehen. Mehrere Ausdrücke lassen

sich in einer Anweisungszeile durch Komma oder Semikolon getrennt angeben. Folgt einem Ausdruck ein Semikolon, so wird die Ausgabe des Ergebnisses auf dem Bildschirm unterdrückt.

MATLAB unterscheidet streng zwischen großen und kleinen Buchstaben. So sind die Variablen `A` und `a` als völlig unabhängig voneinander zu betrachten.

Das Kommando `who` listet alle momentan verwendeten Variablen auf. Eine Variable kann mit dem Befehl `clear <variablenname>` gelöscht werden. `clear functions` löscht alle Funktionen, `clear variabels` löscht alle Variabeln und `clear all` löscht sowohl alle Funktionen als auch alle Variabeln. Einige MATLAB-interne Variablen (Konstanten) lassen sich jedoch nicht löschen.

5 Eingabe von Matrizen

MATLAB arbeitet nur mit einem Datentyp, Matrizen mit möglicherweise komplexen Elementen. Matrizen der Dimension 1×1 dienen als Ersatz für Skalare. Matrizen mit nur einer Zeile oder Spalte werden als Zeilen- bzw. Spaltenvektoren aufgefaßt.
In MATLAB können Matrizen in folgender Weise eingegeben werden:

- explizite Liste von Elementen;
- Aufruf von eingebauten Funktionen;
- Abarbeitung von M-Files (siehe Abschnitt 16);
- Laden von externen Datenfiles.

Soll z.B. die Matrix:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

eingegeben werden, dann kann das durch

```
A=[1 2 3;4 5 6;7 8 9]
```

oder

```
A=[1,2,3;4,5,6;7,8,9]
```

oder

```
A=[
1 2 3
```

```
4 5 6
7 8 9]
```

erfolgen. Das heißt, die Matrixelemente einer Zeile trennt man durch Komma oder Leerzeichen und die Zeilen untereinander durch Semikolon oder RETURN. Bei der Eingabe von Zahlen in exponentieller Form, z.B. 2.34e-9, sind Leerzeichen zwischen der Mantisse und dem Exponenten nicht zulässig.

Jedes individuelle Matrixelement kann aus beliebig komplexen Matlabausdrücken zusammengesetzt werden, z.B.

```
x=[-1.3 sqrt(3) 1+2+3*4/5]
```

erzeugt

```
x= -1.3 1.7321 4.8
```

Schreibt man diese Zeilen in ein File namens `genx.m` und ruft `genx` unter MATLAB auf hat man den selben Effekt. Mit `load` und `fread` können in anderen Matlabsitzungen abgespeicherte Matrizen wieder eingelesen werden. Das Abspeichern erfolgt mit dem Befehl `save <matrixname>`.

Die Built-in-Funktionen, z.B. `rand`, `magic` und `hilb`, gestatten es auf einfache Weise Matrizen für Experimente zu erzeugen. Der Befehl `rand(n)` oder `rand(m,n)` erzeugt eine $n \times n$ bzw. $m \times n$ Matrix mit zufälligen Einträgen; `magic(n)` berechnet eine Matrix, bei der alle Zeilen und Spalten die gleiche Summe haben (magisches Quadrat); `hilb(n)` baut eine $n \times n$ Hilbert-Matrix auf, welche als Standardbeispiel für schlecht konditionierte Matrizen dient.

Auf die Elemente einer Matrix kann durch Angabe der Indizes zugegriffen werden. Zum Beispiel bezeichnet `A(2,3)` das Element a_{23} der Matrix `A`, d.h. das Element, das in der dritten Spalte der zweiten Zeile steht. Ist `x` ein Zeilen- oder Spaltenvektor, so läßt sich das dritte Element des Vektors `x` durch `x(3)` ermitteln.

MATLAB ermöglicht auch die Verwendung komplexer Zahlen. Diese sind zu erzeugen durch multiplikation mit `i` bzw. `j`, z.B.:

```
z= 3+4*i
```

bzw.

```
z= 6+2*j
```

oder auch

```
w= r*exp(i*theta)
```

6 Matrixoperationen

Die folgenden Matrixoperationen sind in MATLAB verfügbar:

+	Addition
−	Subtraktion
*	Multiplikation
^	Potenz
′	Transponieren
\	Linksdivision
/	Rechtsdivision

Selbstverständlich sind diese Operationen auch auf Skalare anwendbar. Wenn bei einer Operation die Dimension der Matrizen nicht verträglich ist, wird eine Fehlermeldung ausgegeben.

Die Operation „Matrixdivision“ verlangt nach einer genaueren Erklärung. Wenn A eine quadratische invertierbare Matrix ist und b ein kompatibler Zeilen- bzw. Spaltenvektor, dann ist:

$x = A\backslash b$ die Lösung von $A * x = b$ und
$x = b/A$ die Lösung von $x * A = b$.

Ist A bei der Linksdivision quadratisch, so verwendet MATLAB den Gaußschen Algorithmus zur Lösung des Gleichungssystems. Wenn A nicht quadratisch ist, wird das Gleichungssystem im Sinne des kleinsten quadratischen Fehlers gelöst. Die Rechtsdivision entspricht einer Linksdivision der Form $b/A = (A'\backslash b')'$.

Die Operationen *, ^, \ und / können auch elementweise erfolgen. Dazu ist dem Operator ein Punkt voranzustellen. Zum Beispiel erhält man sowohl für den Ausdruck [1,2,3,4].*[1,2,3,4] als auch für [1,2,3,4].^2 das Ergebnis [1,4,9,16].

7 Funktionen zur Matrixerzeugung

Einige spezielle Matrizen werden während der Arbeit häufig benötigt. Zur Erzeugung dieser stellt MATLAB verschiedene Funktionen zur Verfügung. Das sind z.B.:

eye	Einheitsmatrix
zeros	Nullmatrix

ones	Matrix aus Einsen
diag	Diagonalmatrix
triu	obere Dreiecksmatrix
tril	untere Dreiecksmatrix
rand	Matrix mit zufälligen Elementen
hilb	Hilbert-Matrix
magic	magisches Quadrat
toeplitz	siehe `help toeplitz`

Zum Beispiel erzeugt `zeros(m,n)` eine $m \times n$ Matrix aus Nullen. Wenn `A` eine bereits existierende Matrix ist, dann erzeugt `zeros(size(A))` ein Nullmatrix der Dimension der Matrix `A`. Für einen Vektor `x` erzeugt `diag(x)` eine quadratische Matrix, bei der die Hauptdiagonale aus den Elementen von `x` aufgebaut ist. Die Hauptdiagonalelemente einer quadratischen Matrix `A` können mit `diag(A)` ermittelt werden.

Eine Matrix kann wiederum aus Matrizen aufgebaut sein (Aufbau von Hypermatrizen). Ist z.B. A eine 3×3 Matrix, dann generiert der Ausdruck:

```
B=[A,zeros(3,2);zeros(2,3),eye(2)]
```

eine 5×5 Matrix.

8 Matrixfunktionen

Einen Großteil seiner Leistungsfähigkeit besitzt MATLAB durch seine Matrixfunktionen. Einige Funktionen sind Bestandteile des MATLAB - Prozessors selbst, andere sind als mit MATLAB mitgelieferte M-Files verfügbar und dann gibt es noch eine Unmenge durch individuelle Nutzer geschriebene M-Files. Einige der nützlichsten sind:

eig	Eigenwerte und Eigenvektoren
chol	Cholesky-Faktorisierung
svd	Singulärwertzerlegung
inv	Matrixinversion
lu	LU-Faktorisierung
qr	QR-Faktorisierung
hess	Hessenbergform
schur	Schur-Zerlegung
expm	Matrixexponentialfunktion
sqrtm	Quadratwurzel einer Matrix

poly	charakteristisches Polynom
det	Determinante
size	Dimension einer Matrix
norm	verschiedene Matrixnormen
cond	Konditionszahl einer Matrix
rank	Rang einer Matrix

In diesem Abschnitt werden die vier unten aufgelisteten Faktorisierungen und Zerlegungen erläutert.

- Trilingular-Faktorisierung
- Orthogonal-Faktorisierung
- Singulärwert-Zerlegung
- Eigenwert-Zerlegung

Kurz beschrieben werden auch die Funktionen `cond`,`rank` und `norm`. Tiefergründige Informationen sind den Nutzer-Handbüchern der Softwarepakete LINPACK und EISPACK zu entnehmen.

8.1 Trilingular-Faktorisierung

Die einfachste Faktorisierung drückt jede quadratische Matrix als ein Produkt aus zwei Dreiecksmatrizen, einer speziell veränderten unteren und einer oberen Dreiecksmatrix, aus. Die Faktorisierung wird meistens als LU- oder auch LR-Faktorisierung bezeichnet. Im allgemeinen wird der Gaußsche Algorithmus als Grundlage benutzt.

Die Faktoren selbst werden durch die Fuktion `lu` bereitgestellt. Die Faktorisierung wird wiederum in den Funktionen zur Ermittlung der Inversen `inv` und der Determinaten `det` einer Matrix benutzt, ebenfalls ist sie die Grundlage zum Lösen linearer Gleichungssysteme sowie der Division quadratischer Matrizen mit \ und /.

An einem Beispiel sei die LU-Faktorisierung erläutert.

```
A =
        1        2        3
        4        5        6
        7        8        0
```

Bei der LU-Faktorisierung nutzt man die Fähigkeit von MATLAB mehrer Matrizen als Ergebnis zu übergeben.

```
[L,U] = lu(A)

L =
    0.1429    1.0000         0
    0.5714    0.5000    1.0000
    1.0000         0         0


U =
    7.0000    8.0000         0
         0    0.8571    3.0000
         0         0    4.5000
```

Zu Beachten ist, daß L eine speziell veränderte untere Dreiecksmatrix ist, welche Einsen in der veränderten Diagonale über der Hauptdiagonale aufweist. Die U Matix ist eine normale obere Dreiecksmatrix. Es ist noch zu prüfen ob die Faktorisierung funktioniert hat.

```
L*U

ans =
        1        2        3
        4        5        6
        7        8        0
```

Die Inverse ebenso wie die Determinante von `A` kann ebenfalls über die Dreiecksmatrizen ermittelt werden.

```
X = inv(A)

X =
   -1.7778    0.8889   -0.1111
    1.5556   -0.7778    0.2222
   -0.1111    0.2222   -0.1111

X = inv(U)*inv(L)

X =
```

```
   -1.7778    0.8889   -0.1111
    1.5556   -0.7778    0.2222
   -0.1111    0.2222   -0.1111

d = det(A)

d =
    27

d = det(L)*det(U)

d =
   27.0000
```

Warum sind nun aber die Formate der beiden `d`'s unterschiedlich? Beim berechnen von `det(A)` sind alle Elemente von `A` Integerwerte. MATLAB stellt das Ergenis demnach auch als Integer dar, während beim Berechnen des zweiten `d`'s die Elemente von `U` Realzahlen sind.

Hier noch ein Beispiel zum Lösen eines unabhängigen linearen Gleichungssystems, `A` ist noch aus dem letzten Beispiel gegeben.

```
b =
        1
        3
        5
```

Zum Lösen der Gleichung `Ax = b` wird der MATLAB - Divisionsoperator genutzt.

```
x = A\b

x =
    0.3333
    0.3333
    0.0000
```

MATLAB ermittelt das Ergebnis durch Lösen des Systems der Dreiecksmatrizen.

```
y = L\b, x = U\y
```

```
y =
    5.0000
    0.2857
    0.0000

x =
    0.3333
    0.3333
    0.0000
```

Die spezielle Funktion **rcond** nutzt die LU-Faktorisierung, sie besteht hauptsächlich aus LINPACK Unterroutinen und dient dem Schätzen der reziproken Konditionszahl einer quadratischen Matrix.

Zwei weitere Funktionen **chol** und **rref** gehören ebenfalls mit in diese Gruppe, da ihre Algotithmen dem der LU-Faktorisierung verwandt sind. Die Funktion **chol** erzeugt die Cholesky Faktorisierung einer symmetrischen, positiv definiten Matrix. Die reduzierte Zeilenstaffelform einer rechteckigen Matrix. Die **rref** Funktion ist von Bedeutung für die theoretische Algebra, für praktische Probleme wird sie kaum verwendet. MATLAB enthält sie aus pädagogischen Gründen.

8.2 Orthogonal-Faktorisierung

Die QR-Faktorisierung ist nützlich für quadratiche und auch rechteckige Matrizen. Sie stellt die Ausgangsmatrix als ein Produkt aus einer orthonormalen Matrix und einer oberen Dreiecksmatrix dar. Nehmen wir z.B.

```
A =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
```

Diese Marix hat einen mangelhaften Rang, die mittlere Spalte ist der Durchschnitt der anderen beiden Spalten. Der mangelhafte Rang wird durch die QR-Faktorisierung offensichtlich.

```
[Q,R] = qr(A)

Q =
```

```
   -0.0776   -0.8331    0.5444    0.0605
   -0.3105   -0.4512   -0.7709    0.3251
   -0.5433   -0.0694   -0.0913   -0.8317
   -0.7762    0.3124    0.3178    0.4461

R =
  -12.8841  -14.5916  -16.2992
         0   -1.0413   -2.0826
         0         0    0.0000
         0         0         0
```

Die Dreiecksstruktur von `R` hat eigentlich nur Nullen unter der Hauptdiagonalen. Die Nullen auf der Hauptdiagonalen impliziert, daß `R` und folglich auch `A` nicht den vollen Rang aufweisen.

Die QR Faktorisierung wird benutzt, um Gleichungssysteme zu lösen, mit mehr Gleichungen als Unbekannten. Verändert man `b` des letzten Abschnitts zu

```
b =
        1
        3
        5
        7
```

so hat das lineare System `Ax = b` vier Gleichungen aber nur drei Unbekannte. Die beste Lösung wird mittels der kleinsten Fehlerquadrate berechnet.

```
x = A\b

Warning: rank deficient, rank = 2  tol =   1.4594e-014
x =
    0.5000
         0
    0.1667
```

Die Meldung warnt, daß ein mangelhafter Rang vorliegt. Der Wert von `tol` ist eine Toleranz, die genutzt wird, um zu entscheiden ob ein Element von `R` unwesentlich ist. Obige Lösung wurde unter Nutzung der QR-Faktorisierung und folgender zwei Schritte ermittelt

```
y = Q'*b;
x = R\y
```

Prüft man nun die Lösung von A*x, stellt man fest, daß sie genau gleich mit `b` ist, ohne jegliche Rundungsfehler aufzuweisen. Obgleich das Gleichungssystem überbestimmt und mit verminderten Rang ist, stimmen sie überein. Es sind unendlich viele Lösungsvektoren `x` möglich und die QR Faktorisierung hat einen davon gefunden.

Die QR-Faktorisierung ist auch die Basis für die Funktionen `null` und `orth`, welche die orthonormale Grundlage für den `Null`-raum und -bereich einer gebenen rechteckigen Matrix erzeugen.

8.3 Singulärwert-Zerlegung

In diesem Heft wird nicht versucht die Singulärwert-Zerlegung zu beschreiben, es genügt zu sagen, daß sie ein leistungsfähiges Werkzeug beim Untersuchen von Matrixproblemen ist. Tiefergründige Erläuterungen sind dem LINPACK Nutzerhanbuch oder dem Buch *Matrix Computations* von Golub und Van Loan zu entnehmen.

Unter MATLAB erzeugt die Dreifachzuweisung

```
[U,S,V] = svd(A)
```

die drei Faktoren der Singulärwertzerlezung (singular value decomposition)

```
A = U*S*V'
```

Die Matrizen `U` und `V` sind orthogonal, und `S` ist eine Diagonalmatrix. Die Funktion `svd(A)` ohne Übernahmevektor liefert als Ergebnis nur die Singulärwerte von `A`.

Verschiedene andere Funktionen nutzen die Singulärwert-Zerlegung, wie z.B. die Pseudoinverse `pinv(A)`, der Rang `rank(A)`, die Euklidische Matrix `norm`, `norm(A,2)` und die Konditionszahl `cond(A)`.

8.4 Eigenwert-Zerlegung

Es sei A eine Matrix der Dimension $n \times n$. Man bezeichnet die n Werte λ_i, $i = 1, \ldots, n$ welche die Gleichung $Ax = \lambda_i x$ erfüllen als die Eigenwerte der Matrix A. Zu ermitteln sind sie mit

```
eig(A)
```

wobei die Eigenwerte in einem Spaltenvektor zurückliefert werden. Wenn A aus reellen Zahlen besteht und symmetrisch ist, dann sind die Eigenwerte ebenfalls reelle Zahlen. Wenn A nicht symmetrisch ist, so sind die Eigenwerte häufig komplex.

```
A =
     0     1
    -1     0

eig(A)
ans =
        0 + 1.0000i
        0 - 1.0000i
```

Beides, sowohl die Eigenwerte als auch die Eigenvektoren kann man mit einem Ergebnisübernahmevektor erhalten.

```
[X,D] = eig(A)
```

In der Diagonalen von `D` sind die Eigenwerte enthalten, und die Spalten von `X` sind die korrespondierenden Eigenvektoren, so daß $A * X = X * D$ ist.

Zwei Zwischenergebnisse beim Berechnen der Eigenwerte sind die Hessenberg Matrix `hess(A)` und die Schur Matrix `schur(A)`. Die Schur Form wird benutzt um tranzendente mathematische Funktionen von Matrizen zu berechnen, wie z.B. `sqrtm(A)` und `logm(A)`.

Sind `A` und `B` quadratische Matrizen, so liefert die Funktion `eig(A,B)` einen Vektor zurück, welcher die generalisierten Eigenwerte folgenden Systems liefert

$$Ax = \lambda * Bx$$

Um auch die Eigenvektoren zu erhalten, wird die Doppelzuweisung genutzt.

```
[X,D] = eig(A,B)
```

In der Diagonalen von `D` sind die Eigenwerte enthalten, und Spalten von X sind die korrespondierenden Eigenvektoren, so daß $A * X = B * X * D$ ist. Die Zwischenergebnisse der Berechnung sind verfügbar über `qz(A,B)`.

8.5 Norm-, Rang- und Konditionszahl

Folgende Funktionen stellt MATLAB im Zusammenhang mit Matrixnorm, -rang und -kondition bereit.

cond	Konditionzahl in 2-Norm
norm	1-Norm, 2-Norm, Frobenius-norm, ∞-Norm
rank	Rang
rcond	geschätzte Kondition

Der Rang einer Matrix wird in vielen anderen M-Files als Unterroutine benutzt.

- in `rref(A)`
- in `A\B` für nicht quadratische Matrizen
- in `orth(A)` und `null(A)`
- in der Pseudoinversen `pinv(A)`

Drei verschiedene Algorithmen mit drei unwesentlich verschiedenen Kriterien, so ist es jedoch möglich, drei verschiedene Werte für ein und die selbe Matrix zu erhalten.

Bei `rref(A)` ist der Rang von `A` die Anzahl der von Null verschiedenen Zeilen. Der Eliminationsalgorithmus von `rref(A)` ist das schnellste der drei Rangbestimmungsverfahren, es ist aber auch das numerisch unausgegorenste sowie unzuverlässigste Verfahren.

Mit `A\B`, `orth(A)` und `null(A)` nutzt MATLAB die QR-Faktorisierung, wie im Kapitel 9 des LINPACK Nutershandbuchs beschrieben.

Der Algorithmus von `pinv(A)` basiert auf der Singulärwertzerlegung und ist im Kapitel 11 des LINPACK Nutershandbuchs beschrieben. Der `svd`-Algorithmus benötigt die meiste Zeit, arbeit dafür aber am zuverlässigsten und wird auch für die explizite Rangberechnung `rank(A)` genutzt.

9 Zugriff auf Untermatrizen

Vektoren und Untermatrizen werden in MATLAB häufig für komplexe Datenmanipulationen benutzt. Die dazu notwendigen Schleifen lassen sich in vielen Fällen durch die Verwendung einer speziellen Notationsform, der Doppelpunktnotation, umgehen. Der Vorteil liegt in der wesentlich schnelleren Abarbeitung solcher Anweisungen durch MATLAB. Die Verwendung dieses Ausdrucksmittels sei hier nachdrücklich empfohlen.

9.1 Verwendung des Doppelpunktes in Vektoren

Bereits bei der Behandlung der Schleifenkonstruktionen von MATLAB (for-Befehl) wurde der Doppelpunkt zum Aufbau eines Vektors verwendet. Soll z.B. ein Vektor y mit natürlichen Zahlen von eins bis fünf aufgebaut werden, so läßt sich das wie folgt notieren:

```
y = 1:5
```

Allgemein gilt für solche Ausdrücke:

Variable = *Anfangswert* : *Endwert*

Möchte man eine andere Schrittweite verwenden, so muß diese als drittes Argument mit angegeben werden. Die allgemeine Form lautet dann:

Variable = *Anfangswert* : *Schrittweite* : *Endwert*

Zum Beispiel erzeugt der Ausdruck

```
0.2:0.2:1.2
```

den Vektor [0.2 0.4 0.6 0.8 1.0 1.2].

Die Schrittweite darf auch negativ gewählt werden.

Ein weiteres Beispiel für die Verwendung dieser Notationsform ist die Berechnung des Graphen der Sinusfunktion:

```
t=0:0.1:2*pi;
y=sin(t);
plot(t,y);
```

Die im folgenden Abschnitt vorgestellten Möglichkeiten gelten ebenso für Vektoren, die als Sonderfall der Matrix anzusehen sind.

9.2 Verwendung des Doppelpunktes mit Matrizen

Die Doppelpunktnotation kann für den Zugriff auf Untermatrizen verwendet werden. Ist z.B. die Matrix `A` gegeben und man möchte lediglich die ersten vier Elemente der dritten Spalte von `A` weiterverarbeiten, dann läßt sich das wie folgt notieren:

```
A(1:4,3)
```

Ein Doppelpunkt allein bezeichnet eine ganze Zeile oder Spalte.

```
A(:,3)
```

würde die gesamte dritte Spalte der Matrix `A` liefern.

Ebenso ist es möglich, mehrere Spalten oder Zeilen einer Matrix anzugeben.

```
A(:,[2 3 5])
```

würde eine Matrix ausgeben, die aus der zweiten, dritten und fünften Spalte der Matrix `A` zusammengesetzt ist. Doppelpunkte dürfen auch auf der linken Seite einer Zuweisung vorkommen.

```
A(:,[2 4 5])=B(:,1:3)
```

ersetzt die zweite, vierte und fünfte Spalte der Matrix `A` durch die erste bis dritte Spalte der Matrix `B`. Voraussetzung ist hier, daß die Matrix `B` die gleiche Anzahl von Zeilen hat wie die Matrix `A`.

9.3 Binäre Vektoren

Man kann binäre Vektoren wie sie gewöhnlich durch Vergleichsoperationen erzeugt werden, zur Bezugnahme auf Untermatrizen benutzen. Es sei `A` eine m x n Matrix und `L` ein binärer Vektor der Länge m. Dann spezifiziert

A(L,:)

die Zeilen von A deren Elemente verschieden von Null sind.

Alle Elemente die größer als die dreifache Standardabweichung sind, werden durch folgenden Term herausgefunden:

```
x=x(x<=3*std(x));
```

Die nächsten beiden Terme

```
L= X(:,3)>100;
```

und

```
x= (L;:);
```

ersetzen `X` durch die Zeilen von `X`, deren drittes Element größer als 100 ist.

9.4 Leere Matrizen

Der Ausdruck

```
X = [ ]
```

erzeugt eine Matrix der Dimension Null mal Null. Dieses ist klar von der Anweisung `clear x` zu unterscheiden, da diese die Variable `X` aus der Liste der aktuellen Variablen entfernt. Man kann die Funktion `exist` verwenden, um zu testen ob eine Matrix existiert oder nicht und mit `empty` ist zu erfragen

ob es eine Matrix der Dimension Null mal Null ist. Mittels leerer Matrizen lassen sich effizient Zeilen und Spalten löschen.

```
A(:,[2 4]) = [ ];
```

löscht die zweite und vierte Spalte von A.

9.5 Spezielle Matrizen

Eine Kollektion spezieller Matrizen wird unter Matlab bereitgestellt.

compan	-	Begleitmatrix
diag	-	Diagonalmatrix
gallery	-	Testmatrizen
hadamard	-	Hadamard
hankel	-	Hankel
hilb	-	Hilbert
invhilb	-	inverse Hilbert
kron	-	Kronecker tensor Produkt
magic	-	magisches Quadrat
pascal	-	Pascalsches Dreieck
toelpitz	-	Toelpitz
vander	-	Vandermonde

Zum Beispiel läßt sich eine Begleitmatrix zum Polynom $x^3 - 7x + 6$ fogendermaßen erzeugen:

```
p= [1 0 -7 6];
A= compain(p);
```

Somit sind die Eigenwerte von `A` gleich den Wurzeln des Polynoms.

```
eig(A) =
        -3.000
         2.000
         1.000
```

Weiter Funktionen, die nicht ganz so interessante Matrizen, dafür aber um so nützlichere Matrizen erzeugen, sind:

zeros	-	alle Elemente sind Null
ones	-	alle Elemente sind Eins

rand - gleichverteilte Zufallszahlen
randn - normalverteilte Zufallszahlen
eye - Einheitsmatrix
linspace - Vektor äquidistanter Elemente
logspace - Vektor logarithmisch angeordneter Elemente
meshgrid - Erzeugen des X und Y Feldes für 3-D Graghiken

Eine dieser Funktionen, ist die Funktion `eye(A)` , diese liefert als Ergebnis eine Einheitsmatrix der Dimension, wie A sie aufweist. In den neueren MATLAB-Versionen 4.x wird eine neue Syntax für das Erzeugen von gleichgroßen Matrizen unterstützt. Diese Synatxänderung betrifft die Funktionen `rand`, `ones`, `eye` und `zeros`. Ist z.B. die Matrix `A` bereits definiert und möchte man eine gleichgroße Matrix `B` aufbauen, die nur aus Nullen besteht, dann schrieb man in älteren MATLAB-Versionen `B=zeros(A)` dafür. In neueren MATLAB-Versionen sollte man dafür `B=zeros(size(A))`.

9.6 Erzeugen größerer Matrizen

Es ist einfach größere Matrizen zu erzeugen, indem man Untermatrizen in eckige Klammern einschließt. Z.B.:

```
C = [A A'; ones(size(A) A^2]
```

erzeugt eine Matrix, zweimal so groß wie `A`.

9.7 Matrixmanipulationen

Einige Funktionen zum Drehen, Kippen, Umformen sowie zum Gewinnen spezieller Teile von Matrizen.

rot90 - Rotation
fliplr - vertikales Kippen
flipud - horizontales Kippen
diag - Entnehmen/Erzeugen einer Diagonalmatrix
tril - untere Dreiecksmatrix
triu - obere Dreiecksmatrix
reshape - Dimensionsänderung
' - Transponieren

: - generelles Neudefinieren
(siehe Abschnitt 9.2)

Beispiel zum Ändern einer Matrix der Dimension 3 mal 4 in eine der Dimension 2 mal 6

```
A =
        1       4       7       10
        2       5       8       11
        3       6       9       12

B = reshape(A,2,6)

B =
        1       3       5       7       9       11
        2       4       6       8       10      12
```

Die drei Funktionen **diag**, **triu** und **tril** erlauben den Zugriff auf die obere und untere Dreiecksmatrix sowie auf die Diagonalelemente einer Matrix. Zum Beispiel

```
tril(A)
```

erzeugt

```
ans =
        1       0       0       0
        2       5       0       0
        3       6       9       0
```

Weitere nützliche Funktionen sind **size** und **length**. Die Funktion **size(A)** liefert einen Zweielmemtevektor mit der Dimension von A. Bei einem Vektorargument liefert **length(V)** die Länge des Vektors, dasselbe Ergebnis erreicht man mit **max(size(V))**.

Möchte man von einer Matrix lediglich die Anzahl der Zeilen oder Spalten ermitteln, dann kann dafür auch der **size**-Befehl verwendet werden.

`m=size(A,1)`	Anzahl der Zeilen von **A**
`n=size(A,2)`	Anzahl der Spalten von **A**

10 Skalare Funktionen

Einige MATLAB-Funktionen arbeiten hauptsächlich mit skalaren Werten als Argument. Werden Vektoren oder Matrizen als Argument übergeben, so erfolgt die Berechnung elementweise.
Trigonometrische Funktionen:

sin	asin	sinh	asinh	
cos	acos	cosh	acosh	
tan	atan	tanh	atanh	atan2

Grundfunktionen:

abs	-	Betrag
angle	-	Phasenwinkel
sqrt	-	Quadratwurzel
real	-	Realteil
imag	-	Imaginärteil
conj	-	konjugiert komplexe Zahl
round	-	zur nächsten ganzen Zahl runden
fix	-	gegen Null runden
floor	-	abrunden
ceil	-	aufrunden
sign	-	Vorzeichen bestimmen
rem	-	Divisionsrest bzw. Modula
gcd	-	größter gemeinsamer Teiler
lcm	-	kleinster gemeinsamer Teiler
exp	-	enxponentiell zur Basis e
log	-	natürlicher Logarithmus
log10	-	Logarithmus zur Basis 10

Einige spezielle Funktionen:

bessel	-	Bessel-Funktion
gamma	-	vollständige und unvollständige Gammafunktion
erf	-	Fehlerfunktion
ervinv	-	inverse Fehlerfunktion

Ebenso wie die Grundfunktionen arbeiten die speziellen Funktionen elementweise, wenn ihnen als Argument eine Matrix übergeben wurde.

11 Feldoperationen

Feldoperationen, sind Operationen, die auf jedes Matrixelement einzeln angewandt werden, im Gegensatz zu den normalen algebraischen Operatoren `*, /, \, ^` und `'` , welche sich auf die Matrix als Ganzes beziehen. Ein vor den entsprechenden Operator gesetzter Punkt (.) zeigt an, daß die Operation elementweise auszuführen ist. Mögliche Feldoperatoren sind: `.*` für elementweise Multplikation, `.\` und `./` für elementweise Division und `.^` für die elementweise Potenzierung.

12 Programmablaufsteuerung

MATLAB hat, wie die meisten Programmiersprachen, Befehle zur Steuerung des Programmablaufes. Dazu gehören sowohl Befehle zur Schleifenkonstruktion als auch zur bedingten Verzweigung.

12.1 For-Schleifen

For-Schleifen verwendet man um eine definierte Anzahl von Schleifendurchläufen zu erreichen. Die allgemeine Form einer For-Schleife in MATLAB lautet:

```
for Index = Ausdruck},
        Anweisungszeile}
end
```

Die Anzahl der Spalten der Matrix *Ausdruck* ist verantwortlich für die Anzahl der Schleifendurchläufe. Bei jedem Schleifendurchlauf wird die nächste Spalte dieser Matrix der Laufvariablen *Index* zugewiesen, solange bis alle Spalten abgearbeitet sind. Daraus folgt, daß genau so viele Schleifendurchläufe erfolgen, wie *Ausdruck* Spalten hat. Das Kommando

```
for i=1:3, x(i)=i^2, end
```

belegt z.B. den Vektor x mit den Werten x=[1,4,9].

Es ist möglich Schleifen zu verschachteln.

```
for i = 1:m
        for j = 1:n
                A(i,j) = 1/(i+j-1);
        end
end
```

Das Semikolon verhindert, daß `A` während jedes Schleifendurchlaufs erneut dargestellt wird.

Für ein weiteres Beispiel sei

```
t =          A =

    -1                1     -1      1     -1      1
     0                0      0      0      0      1
     1                1      1      1      1      1
     3               81     27      9      3      1
     5              625    125     25      5      1
```

gegeben. Es ist `t` der Ausgangsvektor und `A` die gesuchte Vandermonde Matrix, d.h. eine Matrix deren Spalten Potenzen der Elemente von `t` sind.

Zuerst die naheliegenste Doppelschleifenkonstruktion

```
n = length(t);
for j = 1:n
        for i = 1:n
                A(i,j) = t(i)^(n-j);
        end
end
```

Die nachfolgende einfache Schleife ist um einiges schneller und zeigt gleichzeitig, daß *for* Schleifen rückwärts abgearbeitet werden können.

```
A(:,n) = ones(n,1);
for j = n-1:-1:1
        A(:,j) = t.*A(:,j+1);
end
```

12.2 While-Schleifen

While-Schleifen dienen zur Wiederholung einer Anweisungsfolge, solange eine bestimmte Bedingung erfüllt ist. Die allgemeine Form einer While-Schleife in MATLAB ist:

```
while Bedingung
        Anweisungszeile(n)
end
```

Die *Anweisungszeilen* werden solange wiederholt, wie die *Bedingung* zutrifft. Möchte man z.B. für eine gegebene Zahl `a` diejenige nichtnegative ganze Zahl `n` ermitteln, für die $2^n \geq a$ gilt, dann ist das durch die folgende Anweisungsfolge möglich:

```
n=0;
while 2^n < a,
n=n+1;
end
n
}
```

12.3 Programmverzweigungen

Programmverzweigungen werden benötigt, um abhängig von bestimmten Bedingungen nur ausgewählte Programmteile abzuarbeiten. Die allgemeine Form des in MATLAB verwendeten `if`-Befehls lautet:
`if` *Bedingung*1,
Anweisungszeile(n)
`elseif` *Bedingung*2
Anweisungszeile(n)
⋮
`elseif` *Bedingung*n
Anweisungszeile(n)
`else`
Anweisungszeile(n)
`end`
Der `else`-Zweig und der `elseif`-Zweig sind dabei optional. Das folgende Beispiel demonstriert, wie eine einfache Bestimmung des Vorzeichens einer Zahl n erfolgen kann. Das Vorzeichen wird dazu in der Variablen `sig` vermerkt.

```
if n == 0,
sig=0;
elseif n > 0,
sig=1;
else
sig=-1;
end;
```

Als zweites Beispiel sei ein faszinierendes Problem der Zahlentheorie verwendet. Man nehme eine beliebige positive ganze Zahl, ist sie gerade, teile man

sie durch 2 und ist sie ungerade, multipliziere man sie mit 3 und addiere noch 1. Das Faszinierende ist nun, daß es einige Zahlen gibt, für die dieser Prozeß nich abbricht. In diesem Beispiel wird die Funktion von *while* und *input* sowie von *if* und *break* gezeigt. Mit *break* existiert eine Möglichkeit gegeben Schleifen vorzeitig zu beenden.

```
% Classic "3n+1" problem from number theory.
while 1
   n = input('Enter n, negative quits. ');
   if n <= 0, break, end
   while n > 1
      if rem(n,2) == 0
         n = n/2
         else
            n = 3*n+1
         end
    end
end
```

12.4 Logischen Ausdrücken

Die von MATLAB unterstützten Vergleichsoperatoren sind:

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

Ein häufiger Fehler in MATLAB-Anweisungen ist die Verwendung des „=“ Zeichens für Vergleiche. Das „=“ darf nur für **Zuweisungen** an Variablen verwendet werden. Für **Vergleiche** ist das „==“ zu verwenden. Wenn in einer Bedingung mehrere Teilbedingungen miteinander verknüpft werden müssen, dann verwendet man dazu logische Operatoren:

&	und
|	oder
~	nicht

Bedingungen werden in MATLAB, wie auch in den meisten anderen Programmiersprachen, als spezielle Form von Ausdrücken behandelt. Das heißt, daß die

Vergleichsoperatoren als normale Operatoren in allen Ausdrücken (nicht nur in Verbindung mit den Befehlen zur Programmablaufsteuerung) verwendet werden können. Die folgende Zeile ist somit eine gültige MATLAB-Anweisung:

```
c=a>b
```

Der Vergleich wird hierbei für jedes Element vorgenommen. Die Elemente von c werden dann nach folgender Vorschrift bestimmt:

$$c_{ij} = \begin{cases} 1 & \textit{für } a_{ij} > b_{ij} \\ 0 & \textit{für } a_{ij} < b_{ij} \end{cases} \qquad \forall\, i = 1 \ldots m, \quad \forall\, j = 1 \ldots n$$

Wird bei der Auswertung von logischen Ausdrücken ein Vektor oder eine Matrix verwendet, dann ist der Wahrheitswert des Gesamtausdrucks gleich der Und-Verknüpfung aller Elemente der Matrix. Beispiel:

```
if A,
```
Anweisung1
```
else
```
Anweisung2
```
end
```
Ist bei der Abarbeitung des If-Befehls ein Element der Matrix A ungleich Null, dann wird *Anweisung2* abgearbeitet. Sind dagegen alle Elemente von A verschieden von Null, so wird *Anweisung1* abgearbeitet.

Zur Formulierung logischer Ausdrücke werden auch noch die Funktionen `any` und `all` zur Verfügung gestellt. Für einen Vektor `x` liefert `any(x)` den Wert Eins zurück, wenn irgendein Element des Vektors ungleich Null war. Null wird demzufolge zurückgeliefert, wenn alle Elemente des Vektors `x` Null waren. Für eine Matrix `X` erfolgt die Auswertung von `any(X)` spaltenweise, und das Ergebnis wird in einem Zeilenvektor zurückgegeben. Der Befehl `all` ist das logische Gegenstück zu `any`. Er liefert den Wert Eins zurück, wenn alle Elemente ungleich Null sind, in allen anderen Fällen ist das Ergebnis Null.

13 Datenanalyse und Statistik

Dieser Abschnitt ist eine Einführung in die Datenanalyse mit MATLAB und beschreibt einige grundlegende statistische Werkzeuge.

Die hier behandelten MATLAB-Funktionen arbeiten grundsätzlich mit Vektorelementen. Übergibt man als Argument eine Matrix, dann erfolgt die Bearbeitung der Matrix spaltenweise, wobei der Rückgabewert ein Zeilenvektor ist.

13.1 Spaltenorientierte Analyse

Die Urdaten werden, wie zu erwarten, in Matrizen abgelegt, offen ist noch die Frage des Aufbaus der Matrizen. Unter MATLAB werden die Meßwerte einer Stichprobe in einer Zeile abgelegt, die einzelnen Stichproben werden untereinander angeordnet und ergeben eine Matrix. Somit wird ein Satz von 50 Stichproben über 13 Variablen in einer Matrix der Dimension 50×13 abgespeichert.

Beispiel:

Ein Stichprobensatz wird mit dem UNIX Wortzählbefehl `wc` erzeugt. Zur Zeit als das Verzeichnis untersucht wird, befinden sich im Verzeichnis 21 Dateien. Der Befehl `wc` liefert 3 Beobachtungswerte:

- Zeilenzahl
- Wortzahl
- Anzahl der benötigten Bytes

Das Ergebnis wird in einer ASCII-datei count.dat abgespeichert.

```
11      57     291
43     178    1011
38     163    1095
61     420    2407
12      59     287
12      61     346
28     132     841
18     106     541
11      56     282
17      69     394
19     131     754
38     287    1558
42     218    1272
11      56     284
35     197     996
13      51     364
57     222    1990
44     189    1258
32     164     921
114    459    3261
10      51     268
```

Nach dem die Datei erzeugt ist, kann sie mit `load count.dat` in den Arbeitsspeicher geladen werden. Dort wird eine Matrix mit der Bezeichnung `count` angelegt. Für unser Beispiel von Stichproben mit jeweils 3 Variablen enthüllen wir nun die Dimension der Matrix.

```
[n,m] = size(count)
n =
       21
m =
        3
```

Eine Gruppe von grundlegenden statistische Funktionen sind:

max	-	gößter Wert
min	-	kleinster Wert
mean	-	Mittelwert
median	-	Median- bzw. Zentralwert
std	-	Standardabweichung
sort	-	Sortierfunktion
sum	-	Summe der Elemente
prod	-	Produkt der Elemente
cumsum	-	kumultative Summe der Elemente
diff	-	Differenz aufeinanderfolgender Elemente
hist	-	Histogramm
corrcoef	-	Korrelationskoeffizient
cov	-	Kovarianz einer Matrix
any	-	logisches ODER der Elemente
all	-	logisches UND der Elemente

Wird ein Vektor als Argument übergeben, spielt es keine Rolle, ob es sich um einen Zeilen- oder Spaltenvektor handelt. Bei Matrixargumenten, werden diese spaltenweise ausgewertet, so liefert `max` eines Feldes einen Zeilenvektor als Ergebnis, der die Maximalwerte der einzelnen Spalten als Elemente enthält.

Es können beliebig viele neue Funktionen definiert werden. Man muß jedoch immer daran denken das andere Nutzer von einer spaltenweisen Auswertung ausgehen, wenn man seine M-Files nicht aus puren Eigennutz schreiben will. Vor dem schreiben eigener Statistikfunktionen, ist es angebracht, zuerst mitgelieferte M-Files der Datenanalyse zu studieren.

Zurück zu der Beispieldatei `count.dat`

```
mx = max[count]
mu = mean(count)
sigma = std(count)
```

führt zu

```
mx =
            114         459        3261
mu =
        31.7143    158.3810    972.4286
sigma =
        24.7066    116.5480    793.5427
```

Man kann den Mittelwert von jeder Spalte abziehen.

```
e = ones(n,1)
X = count - e*mu
```

13.2 Fehlende Werte

Der spezielle Wert NaN, steht für keine Zahl (Not a Number) in MATLAB. Normalerweise liefern nicht definierte Ausdrücke, wie `0/0`, NaN anstelle einer Fehlermeldung. Dank der Definition des IEEE Gleitpunktarithmetikstandards, ist dies möglich. Für statistische Zwecke können NaNs ebenfalls verwendet werden, um fehlende Werte darzustellen. Der gleiche Effekt wird mit NA, was für nicht verfügbar (not available) steht, erreicht.

Die korrekte Behandlung von NAs ist ein kompliziertes Problem, oft abhängig von der speziellen Situation. Matlab ist in der Behandlung von NAs einheitlich hart, wenn ein Teilterm einer Berechnung NaN enthält, so ist das Endergebnis ebenfalls NaN, auch wenn das Gesamtergebnis nicht von dem Wert, der NaN ist, abhängt.

Sie müssen vor der Berechnung statistischer Werte zu erst alle NaNs entfernen. Zu finden sind die NaNs mit

```
i = find(isnan(x));
```

und zu entfernen mit

```
x = x(find(~isnan(x)));
```

Das gleiche Ergebnis erreicht man mit

```
i = x(~isnan(x));
```

und

```
x(isnan(x)) = [];
```

Die letzte Version ist hierbei die unkomplizirteste. Die spezielle Funktion `isnan` wird bereitgestellt, da

```
x(x == NaN) = [];
```

nicht das Ergebnis liefert, wie man es erwartet, sondern den Vektor unverändert läßt.

Handelt es sich nicht um einen Vektor, sondern um die Zeilen einer Matrix, so kann man mit

```
X(any(isnan(X)'),:) = [];
```

die Matrix von allen Zeilen mit NaNs befreien.

Wenn man häufiger NaNs zu entfernen hat ist es günstig

```
function X = excise(X)
X(any(isnan(X)'),:) = [];
```

in einer Datei namens `excise.m` abzulengen.

Mit der Eingabe von

```
X = excise(X);
```

vereinfacht sich somit das Elemenieren von NaNs für spätere Aufgaben. Siehe auch Abschnitt 16 .

13.3 Enfernen von Ausreißern

Ausreißer-Werte können auf die selbe Art und Weise wie NaNs entfernt werden. Am Beispiel `count.dat` sei dies erläutert. Der Mittelwert und die Standardabweichung jeder Spalte sind

```
mu = mean(count)
sigma = std(count)
```

Die Anzahl der Ausreißer, in diesem Fall, die Werte die größer als die dreifache Standardabweichung sind, wird ermittelt mit Hilfe eines Vektors `e`, der n Einsen enthält.

```
outliers = abs(count - e*mu) > 3*e*sigma;
nout = sum(outliers)

nout =
        1       0       0
```

Somit befindet sich ein Ausreißer in der ersten Spalte, man kann seine Zeile mit

```
count(any(outliers'),:) = [] ;
```

entfernen.

13.4 Regression

Viele Datenanalysen benötigen normierte Eingangsdaten. So wird z.B. der Mittelwert Null gesetzt und die Standardabweichung zu Eins. Mit MATLAB erreicht man dies mit folgenden Zeilen.

```
e = ones(length(count),1);
X = count - e*mean(count);
X = X ./ (e*std(X));
```

Für die Beispieldaten ist eine streng lineare Abhängigkeit der drei Variablen einer Beobachtung: Linien,Wörter und Byteanzahl gegeben. Zu erkennen ist dies an den Korrelationskoeffizienten, die alle nahe eins sind. Für normierte Daten sind die Korrelationskoeffizienten einfach

```
X'*X/(n-1)

ans =
        1.0000  0.9242  0.9704
        0.9242  1.0000  0.9710
        0.9704  0.9710  1.0000
```

Für nicht normierte Daten, kann man die Funktion `corrcoef` verwenden, welche zuerst eine Normierung durchführt und dann denn Koerralationskoeffizienten bestimmt.

Um die letzte Spalte Byteanzahl zurückzuführen auf die anderen beiden Wort- und Zeilenanzahl, nutzt man die Rohdaten. Zuerst wird ein Vektor erstellt der die Daten der lezten Spalte Byteanzahl enthält, dann wird eine Matrix erzeugt deren erste Spalte aus Einsen besteht gefolgt von der ersten und zweiten Spalte der Matrix mit den Urdaten.

```
y = count(:,3);
A = [e count(:,1:2)];
```

Jetzt kann man mit der MATLAB Linksdivision die Regressionskoeffizienten dieses überbestimmten Systems errechnen.

```
beta = A\y

beta =
        -85.6334
         16.0876
          3.4591
```

Eine Matrix-Vektor Multiplikation liefert uns das Regressionsresultat.

```
fit = A*beta;
```

Nun kann man die Orginaldaten mit den ermittelten Regressionsdaten in `fit` vergleichen. Hierzu ordnen wir die Urdaten in aufsteigender Reihenfolge, die in `fit` enthaltenen Daten ordenen wir in der selben Anordnung (Abb. 1).

```
[y,k] = sort(y);
fit = fit(k);
plot(1:n,y,'o',1:n,fit,'-')
```

Da die Urdaten ziemlich gut aber nicht genau linear korrelierten, ist der Graph eine gute aber nicht ganz genaue Verbindungslinie zwischen den Punkten der Rohdaten.

Oft wird ein Polynom $p(x) = c_1 x^d + c_2 x^{d-1} + ... + c_n x^0$ gesucht mit dem die Daten der Vektoren `x` und `y` möglichst gut getroffen werden.

Die Ordnung des Polynoms ist `d` und die Anzahl der Koeffizienten ist $n = d+1$. Die Koeffizienten $c_1, c_2, ...c_n$ sind durch Lösen eines Systems unabhängiger linearer Gleichungen $A * c = y$ zu bestimmen.

Die Spalten von `A` sind die geordneten Potenzen des Vektors `x`, ein Weg `A` zu erzeugen ist

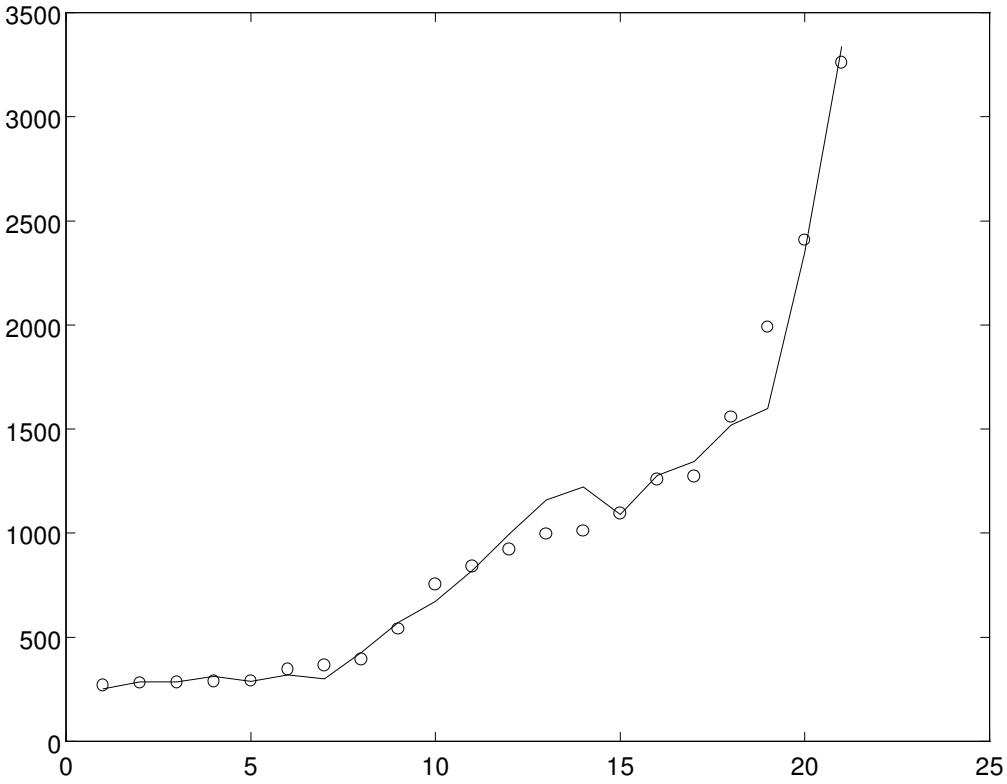

Abbildung 1: Regression

```
for j=1:n
        A(:,j) = x.^(n-j);
end
```

Gelöst werden kann das Gleichungssystem mit dem Divisionsoperator von MATLAB `c = A\y` .

Die Funktion `polyfit.m` aus der MATLAB Toolbox automatisiert diese Prozedur.

Bei Regressionsproblemen sind oft auch andere Funktionen nützlich um die Daten der Spalten der Datenmatrix zu aproximieren, z.B.

```
A = [count(:,1) count(:,2).^2 sin(count(:,3)) ...
ones(n,1)];
coef = A\y;
```

findet die Regressionskoeffizienten eines komplizierteren Modells.

14 Polynome und Signalverarbeitung

MATLAB hat Funktionen um mit Polynomen zu arbeiten und zur digitalen Signalverarbeitung. Diese operieren hauptsächlich mit Vektoren.

14.1 Darstellung von Polynomen

MATLAB stellt Polynome als Zeilenvektoren dar, welche die Koeffizienten in fallender Ordung enthalten, z.B. die charakteristische Gleichung der Matrix

```
A =
     1     2     3
     4     5     6
     7     8     0
```

wird berechnet mit

```
p = poly(A)

p =
    1.0000   -6.0000  -72.0000  -27.0000
```

Dieses ist die MATLAB - Darstellung des Polynoms $s^3 - 6s^2 - 72s - 27$. Die Wurzeln der Gleichung sind

```
r = roots(p)

r =
   12.1229
   -5.7345
   -0.3884
```

Diese Wurzeln sind selbstverständlich gleich den Eigenwerten der Matrix `A`. Aus den Wurzeln kann das Polynom mit der Funktion `poly` zurückgewonnen werden.

```
p2 = poly(r)

p2 =
    1.0000   -6.0000  -72.0000  -27.0000
```

Zur Bildung des Produktes zweier Polynome nutzen man die Polynommultiplikation. Gegeben seien $a(s) = s^2 + 2s + 3$ und $b(s) = 4s^2 + 5s + 6$, gesucht ist $c = a * b$.

```
a = [1 2 3]; b = [4 5 6];
c = conv(a,b)

c =
     4    13    28    27    18
```

Mit der Polynomdivision kann a bzw. b wieder herausdividiert werden,

```
[q,r] = deconv(c,b)

q =
     1     2     3
r =
     0     0     0     0     0
```

`r` ist der Divisionsrest. Hier noch eine Liste aller Polynomfunktionen

poly	-	charakteristisches Polynom
roots	-	Wurzeln des Polynoms
polyval	-	Funktionswertberechnung
polyvalm	-	Funktionswertberechnung bei Matrixargument
conv	-	Multiplikation
deconv	-	Division
residue	-	Partialbruchzerlegung
polyder	-	Ableitung eines Polynoms
polyfit	-	Regression mit Polynomzielfunktion

14.2 Signalverarbeitung

Vektoren werden genutzt um Abtastdaten zu speichern, handelt es sich um Systeme mit mehreren Eigangsdaten, so entspricht je Zeile einem Abtastzeitpunkt und jede Spalte einem Signaleingang.

Das Grund - MATLAB - System besitzt nur einige wenige Signalverarbeitungsfunktionen.

abs	-	Betrag
angele	-	Phasenwinkel
cov	-	Kovarianz
conv	-	Polynommultiplikation
deconv	-	Polynomdivision
fft	-	schnelle Fourier-Transform
ifft	-	inverse FFT
fftshift	-	tauschen der Quadranten einer Matrix

Zu einigen dieser Funktionen gibt es zweidimensionale Entsprechungen, wenn das Signal als Matrix vorliegt.

fft2	-	zweidimensionale FFT
ifft2	-	inverse zweidimensionale FFT
fftshift	-	umstellen der FFT Resultate
conv2	-	zweidimensionale Polynomenmultiplikation

Weiter Signalverarbeitungsfunktionen sind dem Handbuch zur *Signal Processing Toolbox* zu entnehmen.

14.3 Filterung von Daten

Die Funktion

```
Y = filter(b, a, x)
```

filtert die Daten in dem Vektor `x` nach der Filterbeschreibung durch die Vektoren `a` und `b` und legt den Ergebnisvektor in `y` ab. Filter sind die als Differenzengleichung oder gebrochen rationale Funktionen in z dargestellten Übertragungsglieder.

$$\begin{aligned} y(n) &= b(1)x(n) + b(2)x(n-1) + ... + b(nb)x(n-nb+1) \\ &\quad -a(2)y(n-1) - ... - a(na)y(n-na+1) \end{aligned} \tag{1}$$

oder als Z-Transformierte

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b(1) + b(2)z^{-1} + ... + b(nb)z^{-(nb-1)}}{1 + a(2)y(n-1) - ... - a(na)z^{-(na-1)}}$$

Hier noch ein Beispiel, wie die Impulsantwort eines Digitalfilters zu finden und zu zeichnen ist.

```
x = [1 zeros(1,n-1)];
y = filter(b,a,x);
plot(y,'o')
```

Die Funktion `freqz` der *Signal Processing Toolbox* liefert die komplexe Frequenzantwort des Digitalfilters. Die Frequenzantwort H(z) ist definiert in der Umgebgung des Einheitskreises der komplexen Ebene, $z = e^{j\omega}$. Man kann `freqz` nutzen um die Frequenzantwort zu finden und zu zeichen.

```
[h,w] = freqz(b,a,n);
mag = abs(h);
phase = angle(h);
semilogy(w,mag)
plot(w,phase)
```

Die *Signal Processing Toolbox* beinhaltet eine Vielzahl von Funktionen zum Entwerfen von Digitalfiltern. Mit einigen Kenntnissen im Filterentwurf sind viele Verfahren verfügbar. Zur Unterstützung des Filterentwurfs stehen in MATLAB Funktionen zur Transformation kontinuierlicher Systeme in zeitdiskrete zur Verfügung (z.B. `c2d` und `c2dm`).

14.4 Nutzung des FFT-Algorithmus

Der FFT-Algorithmus zum Berechen der diskreten Fouriertransformierten einer Sequenz ist das Arbeitspferd der digitalen Signalverarbeitung schlechthin. Er nutzt Funktionen wie Filterung, Polynommultiplikation, Frequenzantwortberechnung, Gruppenlaufzeiten bis hin zur Ermittlung des Dichtespektrums.

`fft(x)` ist die diskrete Fouriertransformierte des Vektors `x`, berechnet mit der radix-2 fast Fouriertransformation, wenn die Länge des Vektors x eine Potenz von 2 ist und mit dem mixed radix Algorithmus, wenn Die Länge keine Potenz von zwei ist. Handelt es sich bei `X` um eine Matrix, so ist `fft(X)` die Fouriertransformierte jeder der einzelnen Spalten.

`fft(x,n)` ist die n-Punkt FFT. Wenn die Länge von `x` kleiner als `n` ist, so wird `x` bis zur vollen Länge `n` mit Nullen aufgefüllt. Wenn die Länge von `x` größer als `n` ist, so werden die überzähligen Elemente abgeschnitten. Bei einer Matrix wird die Länge der einzelnen Spalten in der selben Weise verkürzt.

`ifft(x)` ist die inverse Fouriertransformierte des Vektors `x`, `ifft(x,n)` ist die n-Punkt inverse FFT.

Die beiden Funktionen implementieren das Transformationspaar durch

$$X(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)}$$

$$x(j) = \sum_{k=1}^{N} X(k)\omega_N^{-(j-1)(k-1)}$$

wobei

$$\omega_N = e^{-2\pi/N}$$

die n-te Wurzel der komlexen Einheit ist.

Die FFT eines Spaltenvektors `x` sei im Beispiel gezeigt.

```
x = [4 3 7 -9 1 0 0 0]';
y = fft(x)

y =
   6.0000
```

```
  11.4853 - 2.7574i
  -2.0000 -12.0000i
  -5.4853 +11.2426i
  18.0000
  -5.4853 -11.2426i
  -2.0000 +12.0000i
  11.4853 + 2.7574i
```

Obwohl `x` eine Sequenz reeller Zahlen beinhaltet, ist `y` komplex. Die erste Komponente der transformierten Daten ist der Gleichanteil und das fünfte Element korrespondiert mit der Nyquist-Frequenz. Die letzten drei Werte von y korrespondieren zu den negative Frequenzen und für eine Folge reeller Zahlen `x` sind sie die konjungiertkomplexen Zahlen der drei komplexen Zahlen der ersten Hälfte des Vektors `y`.

15 Funktionsfunktionen

Eine Klasse von MATLAB - Funtionen arbeitet nicht mit numerischen Matrizen dafür aber mit mathematischen Funktionen. Dies sind

- numerische Integration
- nichlineare Gleichungen und Optimierung
- Differentialgleichungslösung

MATLAB stellt Funktionen mit Hilfe von M-Files dar, z.B.

$$f(x) = \frac{1}{(x-0,3)^2+0,01} + \frac{1}{(x-0,9)^2+0,04} - 6$$

wird für MATLAB mit folgenden M-File verfügbar gemacht

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

Ein Graph der Funktion ist zu zeichnen mit (Abb. 2)

```
x = -1:.01:2;
plot(x,humps(x))
```

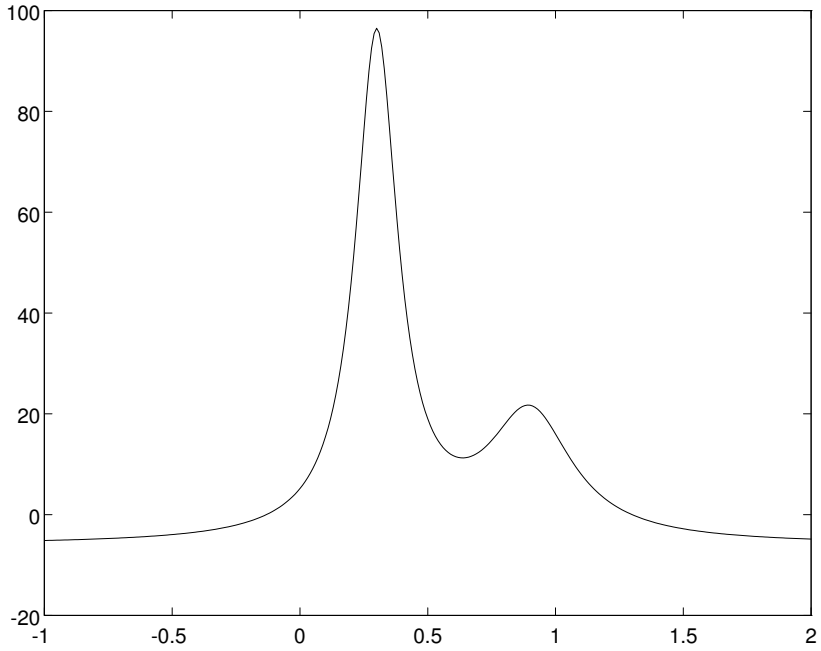

Abbildung 2: Buckelfunktion

15.1 Numerische Integration (rechteck)

Die Fläche unter einer Funktion `f(x)` kann über die numerische Integration von `f(x)` bestimmt werden. Mit der folgenden Zeile wird die Funktion `humps.m` im Bereich von 0 bis 1 integriert.

```
q = quad('humps',0,1)

q =
   29.8583
```

Die zwei MATLAB - Funktionen zur Rechteckintegration sind:

quad - adaptive Simpsonregel
quad8 - adaptive Newton Cotes 8 Feld Regel

Als erstes Argument von `quad` steht ein String, welcher der Name der zu integrierenden Funktion ist. Die beiden nachfolgenden Argumente geben den Anfang und das Ende des integrationsintervalls an.

15.2 Nichlineare Gleichungen und Optimierungsfunktionen

Die Funktionsfunktionen zu nichtlinearen Gleichungen und zur Optimierung sind:

fmin - Minimum einer Funktion einer Variablen
fmins - Minimum einer multivariablen Funktion (Simplexverfahren)
fzero - Nullstelle einer Funktion einer Veränderlichen

Fortfahrend mit dem Beispiel `humps.m` wird das Minimum des Bereiches von 0,5 bis 1 bestimmt.

```
xm = fmin('humps',.5,1)

xm =
    0.6370
```

Somit ist der y-Wert des Minimums

```
y = humps(xm)

y =
   11.2528
```

Betrachtet man den Graphen, so sieht man, daß die Funktion **humps** zwei Nullstellen besitzt. Die Nullstelle der Umgebung von x = 0 ist

```
xz1 = fzero('humps',0)

xz1 =
   -0.1316
```

Die Nullstelle nahe x = 1 ist

```
xz2 = fzero('humps',1)

xz2 =
    1.2995
```

Die Optimierungs-Toolbox enthält sieben weitere Funktionsfunktionen für nichtlineare Gleichungen und zur Optimierung:

attgoal - Mehrkriterielle Optimierung
constr - Optimierung mit Nebenbedingungen
fminu - unbeschränkte Optimierung
fsolve - Lösung nichtlinearer Gleichungen
leastsq - Minimierung quadratischer Funktionen
seminf - Optimierung mit Nebenbedingungen als Intervall

15.3 Differentialgleichungsfunktionen

MATLAB hat folgende Funktionen zum Lösen gewöhnlicher Differentialgleichungen.

ode23 - Runge-Kutta Verfahren, 2er/3er Ordnung
ode45 - Runge-Kutta-Fehlberg Verfahren, 4er/5er Ordnung

Die Differentialgleichung zweiter Ordnung, welche als Van der Pol Gleichung bekannt ist,

$$\ddot{x} + (x^2 - 1)\dot{x} + x = 0$$

wird für nachfolgendes Beispiel verwendet. Diese Differentialgleichung kann auch als System zweier gekopplter Differentialgleichungen erster Ordnung beschrieben werden.

$$\begin{aligned} \dot{x}_1 &= x_1(1 - x_2^2) - x_2 \\ \dot{x}_2 &= x_1 \end{aligned}$$

Der erste Schritt in Richtung Simulation des Systems, ist ein M-File zu schreiben, welches das Differentialgleichungssystem enthält. Die Datei wird **vdpol** genannt.

```
function xdot = vdpol(t,x)
xdot = zeros(2,1);
xdot(1) = x(1).*(1-x(2).^2)-x(2);
xdot(2) = x(1);
```

Um das System von Differentialgleichungen, das in **vdpol** beschrieben ist, im Interval $0 \leq t \leq 20$ zu simulieren, wird ode23 verwendet. Mit **plot** läßt sich das Ergebnis darstellen (Abb. 3).

```
t0 = 0; te = 20;
x0 = [0 0.25]'; % Startwerte
[t,x] = ode23('vdpol',t0,te,x0);
plot(t,x(:,1),'-.',t,x(:,2))
```

Für weitergehende Simulationen, sei auf das *Simulink* - Handbuch verwiesen. SIMULINK ist eine umfassende auch grafische Erweiterung von MATLAB zur Simulation von Differentialgleichungssystemen.

16 M-Files

Bei der Arbeit mit MATLAB werden bestimmte Befehlsfolgen immer wieder verwendet. MATLAB gestattet es, Befehlssequenzen in Files abzulegen. Solche Files **müssen** die Endung `.m` haben (daher der Name M-File). M-Files

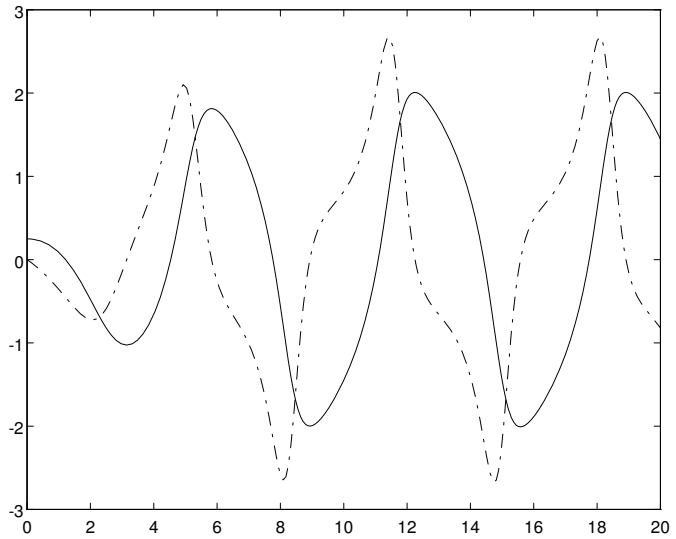

Abbildung 3: Van der Pol

sind gewöhnliche ASCII-Files, wie sie mit jedem normalen Editor (z.B. Editor des Norton-Commanders, Notepad von Windows, Turbo-Pascal-Editor u.a.) erzeugt werden können. Es werden zwei Arten von M-Files unterschieden: Script-Files und Funktionen.

16.1 Script-Files

Script-Files, auch als Macros bezeichnet, bestehen aus der gewöhnlichen Aneinanderreihungen von MATLAB-Befehlen, so wie man sie auch in der Kommandozeile eingeben würde. Sie werden zur Erzeugung großer Matrizen oder

langer Kommandosequenzen verwendet. Erzeugt man ein File namens `data.m`, das die folgende Zeilen enthält

```
A=[
1 2 3 4
5 6 7 8
];
```

und ruft dann von MATLAB aus das File auf (eingeben des Filenamens, hier `date`, gefolgt von der Enter-Taste), so wird die Matix `A` in MATLAB erzeugt. Der Vorteil dieser Vorgehensweise ist die leichte Editierbarkeit der Matrix, wenn im Nachhinein Änderungen erforderlich sind.

Als weiteres Beispiel dient die Scriptdatei `fibno.m`

```
% An M-file to calculate Fibonacci numbers.
f = [1 1]; i = 1;
while f(i) + f(i+1) < 1000
        f(i+2) = f(i) + f(i+1);
        i = i + 1;
end
plot(f)
```

Nach Eingabe von `fibno` in der Kommandozeile und Abschluß mit ENTER werden die ersten 16 Fibonacci Zahlen berechnet und gezeichnet. Die Variabeln `f` und `i` verbleiben danach im Arbeitsspeicher verfügbar.

Die Demos die mit MATLAB mitgeliefert werden, sind gute Beispiele, wie Script-Files zu verwenden sind.

Beim Aufrufen von MATLAB wird automatisch das Script-File `matlabrc.m` ausgeführt, welches wiederum das File `startup.m` aufruft. Dieses File ist ein geeigneter Platz für häufig verwendete Konstanten.

16.2 Funktionsfiles

Funktionen bieten die Möglichkeit MATLAB um eigene Befehl zu erweitern. Sie sind ebenso wie Script-Files gewöhnliche ASCII-Files. Der Unterschied besteht nur in der ersten Zeile des Files, welche den „Funktionskopf“ enthält. Beispiele für MATLAB-Funktionen sind reichlich vorhanden, da der überwiegende Teil von MATLAB aus solchen Funktionen besteht. Selbstgeschriebene Funktionen unterscheiden sich demzufolge nicht von „normalen“ MATLAB-Funktionen.

Zur Erläuterung folgt ein einfaches Beispiel für eine MATLAB-Funktion:

```
function P = prodsqr(A,B)\\
% PRODSQR Produkt der Quadrate zweier Matrizen\\
P=A^2 * B^2;
```

Werden diese Zeilen unter dem Namen `prodsqr.m` gespeichert, so ist die Funktion `prodsqr`, genau wie andere MATLAB-Funktionen, verfügbar. Dem Schlüsselwort `function` in der ersten Zeile folgt die Bezeichnung der Variablen, die den Rückgabewert enthält, gefolgt vom Namen der Funktion sowie deren Argumenten. Es ist auch möglich, mehr als einen Rückgabewert zu definieren. Dazu werden die durch Komma getrennten Rückgabewerte in eckige Klammern eingeschlossen. Ein Beispiel hierfür ist die Funktion `stat`, welche den Mittelwert und die Standardabweichung des Arguments berechnet.

```
function [mean, stdev] = stat(x)
% STAT Mittelwert und Standardabweichung
% Für einen Vektor x, berechnet stat(x) den
% Mittelwert und die Standardabweichung von x.
% Für eine Matrix x, berechnet stat(x) zwei
% Zeilenvektoren, welche den Mittelwert und
% die Standardabweichung für jede Spalte von x
% enthalten.
[m,n] = size(x);
if m == 1,
   m=n; % Behandeln von Zeilenvektoren
end;
mean = sum(x);
stdev = sqrt(sum(x.^2)/m-mean.^2);
```

Wenn diese Zeilen in einem File mit dem Namen `stat.m` abgelegt werden, liefert das Kommando `[xm,xd] = stat(x)` den Mittelwert des Vektors `x` in der Variablen `xm` und die Standardabweichung in der Variablen `xd` zurück. Wenn nur der Mittelwert benötigt wird, dann ist es **nicht** notwendig, beide Rückgabewerte zu bearbeiten. Der Aufruf `xm=stat(x)` würde nur den Mittelwert von `x` an `xm` zuweisen. Allgemein gilt, daß von den möglichen Rückgabewerten die weiter rechts stehenden weggelassen werden dürfen.

Das % Symbol kennzeichnet in MATLAB Kommentare. Alle Kommentare, die sofort an den Funktionskopf anschließen, werden bei Verwendung des `help`-Befehls angezeigt. Dadurch lassen sich Funktionen mit Hilfetexten versehen. Möchte man über den Hilfetext hinaus auch noch den Rest der Funktion sehen, kann das durch `type <Funktionsname>` erfolgen.

Der ersten Hilfe-Zeile kommt eine besondere Bedeutung zu, sie wird in dem Verzeichnis-File `contents.m` vermerkt. Sucht man z.B. alle Funktionen, die

etwas mit FFT zu tun haben, dann läßt sich das durch `lookfor FFT` erreichen. Der Befehl `lookfor` durchsucht dann von jedem M-File die erste Hilfetextzeile nach dem Auftreten des Suchbegriffs `FFT`. Wird der Suchbegriff gefunden, dann wird der Name des M-Files ausgegeben. Demnach sollte die erste Kommentarzeile möglichst prägnant den Inhalt und die Aufgabe der Funktion erläutern.

16.3 Befehle für M-Files

Einige Befehle sind nur in Verbindung mit M-Files sinnvoll. Dazu gehören die Befehle `nargin` und `nargout`. Sie ermöglichen die Abfrage der Anzahl der übergebenen und zurückzugebenden Variablen. Ein großer Teil der Flexibilität von MATLAB beruht auf der Nutzung dieser Befehle. Ihre Verwendung läßt sich leicht aus den vorhandenen M-Files ableiten. Zum Beispiel aus der Funktion zur Berechnung des Ranges einer Matrix.

```
function r = rank(X,tol)
%Rang,  Anzahl der linear unabhängigen Zeilen bzw. Spalten.
%       K = RANK(X) ist die Anzahl der Singulärwerte von X
%       die größer sind als MAX(SIZE(X)) * NORM(X) * EPS.
%       K = RANK(X,tol)  ist die Anzahl der Singulärwerte
%       von X die größer sind als tol.
s = svd(X);
if (nargin == 1)
    tol = max(size(X)) * s(1) * eps;
end
r = sum(s > tol);
```

Zu Beachten ist, daß `nargin` und `nargout` permanente Variablen von MATLAB sind.

16.4 Ein paar Infos zur internen Verarbeitung

Infolge des ersten Aufrufs eines M-Funktions-Files wird dieses kompiliert und im Arbeitsspeicher abgelegt. Es ist somit für den erneuten Gebrauch verfügbar ohne erneut kompiliert werden zu müssen. Es verbleibt im Arbeitsspeicher für die Dauer der MATLAB - Sitzung, es sei den der Speicherplatz wird knapp, dann werden solche Funktionen automatisch aus dem Arbeitsspeicher entfernt.

Allgemein geht MATLAB folgendermaßen vor, wenn z.B. `whoopie` eingegeben wird.

1. Ist `whoopie` eine Variable?
2. Ist `whoopie` eine interne Matlabfunktion?
3. Ist `whoopie` eine kompilierte Funktion im Arbeitsspeicher?
4. Ist `whoopie` ein M-File in einem Verzeichnis des MATLAB - Suchpfades?

Hatte MATLAB Erfolg, wird den übrigen Alternativen nicht mehr nachgegangen. Möchte man dafür sorgen, daß eine Funktion von MATLAB neu übersetzt wird, so ist sie zuerst aus dem Arbeitsspeicher zu entfernen. Dazu wird das Kommando `clear <funktionsname>` verwendet. Will man alle Funktionen löschen, dann geht dies mit `clear all`.

16.5 Echo, Input, Keyboard und Pause

Normalerweise werden die Kommandos eines M-Funktions-Files bei seiner Ausführung nicht angezeigt. Mit dem Kommando `echo` kann dies verändert werden, nützlich ist dies bei der Fehlersuche.

Die Funktion `input` gestattet Eingaben durch den Nutzer.

```
n = input('Wie viele Birnen')
```

erzeugt die Zeichenkette als Eingabe-Prompt auf dem Bildschirm und wartet auf die Eingane, welche in `n` abgespeichert wird.

Ähnlich zu `input` ist die jedoch leistungsfähigere Funktion `keyboard`. Die Funktion `keyboard` ermöglicht es die Abarbeitung eines M-Files zu stoppen und die Kontrolle an die Tastatur des Nutzers zu übergeben. Angezeigt wird dieser Status durch ein Doppel-Prompt, nun können Variablen erzeugt, berechnet oder einfach nur verändert werden. Alle MATLAB - Funktionen sind in diesem Status verfügbar. Beendet wird dieser Status mit RETURN (D.h. eingeben von R-E-T-U-R-N und anschließendes betätigen der RETURN-Taste), daraufhin wird das unterbrochene M-File weiter abgearbeitet.

Das Kommando `pause` stoppt die abzuarbeitende Prozedur und setzt diese nach der Betätigung einer beliebigen Taste fort. Hingegen `pause(n)` stoppt den laufenden Prozeß für n Sekunden.

16.6 Globale Variabeln

Normalerweise nutzt jede MATLAB - Funktion seine eigenen lokalen Variabeln, welche nur innerhalb dieser Funktion zur Verfügung stehen. Mit dem Attribut `global` kann dieser Grundsatz durchbrochen werden.

laufenden Prozeß für n Sekunden.

16.7 Globale Variabeln

Normalerweise nutzt jede MATLAB - Funktion seine eigenen lokalen Variabeln, welche nur innerhalb dieser Funktion zur Verfügung stehen. Mit dem Attribut `global` kann dieser Grundsatz durchbrochen werden.

Aus stilistischen Gründen sollten für globale Variabeln nicht zu kurze Bezeichnungen aus Großbuchstaben verwendet werden.

Als Beispiel dient der Effekt der Interaktion der Koeffizienten α und β des Lotka-Volterra Räuber/Opfer - Modells.

$$\begin{aligned} \dot{y}_1 &= y_1 - \alpha y_1 y_2 \\ \dot{y}_2 &= -y_2 + \beta + y_1 y_2 \end{aligned}$$

1. Erzeugen eines Files `lotka.m`:

```
funktion yp = lotka(t,y)
% LOTKA   Lotka-Volterra predator-prey model.
global ALPHA BETA
yp = [y(1) - ALPHA*y(1)*y(2); -y(2) + BETA*y(1)*y(2)];
```

2. Ineraktive Eigabe der folgenden Zeilen.

```
global ALPHA BETA
ALPHA = .01
BETA = .02
[t,y] = ode23('lotka',0,10,[1; 1]);
plot(t,y)
```

16.8 Zeichenketten

MATLAB ermöglicht auch die Verwendung von Zeichenketten. Soll einer Variablen eine Zeichenkette zugewiesen werden, so wird die Zeichenkette in einfache Hochkommas eingeschlossen.

```
s= 'Das ist eine Zeichenkette';
```

Zeichenketten werden gewöhnlich zur Benachrichtigung des Nutzers über bestimmte Ergebnise oder Probleme verwendet. Wenn während der Abarbeitung

eines M-Files Fehler auftreten, so kann die Bearbeitung mit einer Nachricht abgebrochen werden.

```
error('Es müssen drei Argumente übergeben werden');
```

Soll einfach nur Text ausgegeben werden, so können die Befehle `disp` und `fprintf` verwendet werden.

```
disp('Bitte warten, die Rechnung kann einige Minuten dauern')
fprintf('Mittelwert:%f Standardabweichung:%f \n',xm,xd)
```

Die Syntax des fprintf-Befehls stimmt mit der des gleichnamigen C-Befehls überein.

Abschließend noch einige Befehle zum Bearbeiten und zur Analyse von Zeichenketten.

```
Name = 'Carsten'
```

führt zu

```
Name =
Carsten
```

Der Text ist in einem Vektor gespeichert. Die Buchstaben sind als ASCII Werte abgebildet und `abs` zeigt diese Werte auf dem Bildschirm. Mit `size` kann die Anzahl der ASCII Zeichen ermiitelt werden.

```
size(Name)

ans =
     1     7

abs(Name)

ans =
    67    97   114   115   116   101   110
```

Nützliche Funktionen sind auch `isstr` zum Prüfen ob eine Zeichenkette vorliegt und `strcmp` zum Vergleichen zweier Zeichenketten.

Unter Zuhilfenahme eckiger Klammern können größere Zeichenketten zusammengesetzt werden.

```
s = [Name,' ','Schulze']
s =
Carsten Schulze
```

Numerische Werte können mit **sprintf**, **num2str** und **int2str** in Zeichenketten umgewandelt werden, häufig wird dies angewandt um Titelzeilen mit numerischen Werten zu erzeugen.

```
f = 70; c = (f-32)/1.8;
title(['Die Raumtemperatur ist ',num2str(c),' Grad Celsius'])
```

16.9 Die Funktion eval

Die Funktion **eval** ist ein leistungsfähiges Makrowerkzeug für Zeichenketten. Bei der Eigabe von **t** erscheint die Zeichenkette, die unter **t** abgespeichert ist, auf dem Bildschirm, mit **eval(t)** wird dieselbe Zeichenkette jedoch als mathematischer Ausdruck oder als Faktor interpretiert. Das folgende Beispiel zur Berechnung der Hilbert-Matrix diene der Veranschaulichung.

```
t = '1/(i+j-1)';
for i = 1:n
        for j = 1:n
                A(i,j) = eval(t);
        end
end
```

Mit Hilfe von **eval** und **input** ist es möglich zwischen Algorithmen, die in verschiedenen M-Files abgelegt sind, auszuwählen. Im Beispiel sind es die Dateien **punt.m**, **kick.m**, etc.

```
plays = ['punt'; 'pass'; 'run '; 'kick'];
k = input('Spielnummer: ');
game = eval(plays(k,:))
```

Da die Zeichenketten als Zeilen einer Matrix abgelegt sind, müssen sie alle die selbe Länge aufweisen, darum wurde an **'run '** ein Leerzeichen angefügt.

Abschließend noch ein Beispiel um zehn aufeinanderfolgend nummerierte Dateien zu laden.

```
fname = 'mydata';
for i = 1:10
  eval(['load ',fname,int2str(i)])
end
```

Das Makrowerkzeug **eval** ist insbesondere nützlich um Funktionsnamen an M-Files zu übergeben. Siehe auch **funm.m**.

16.10 Verbessern der Geschwindigkeit und des Speichermanagements

Die internen Operationen von MATLAB zur Matrix- und Vektorbearbeitung sind um mehr als eine Größenordnung schneller als die von selbstgeschriebenen Funktionen. Man sollte deshalb Schleifen durch Matrixoperationen ersetzen, soweit es das Problem zuläßt. Am Beispiel der Berechnung des Sinus von 1001 Zahlen zwischen 1 und 10 sei dies verdeutlicht.

```
i = 0;
for t = 0:.01:10
        i = i+1;
     y(i) = sin(t);
end
```

Dieser Algorithmus ist um mehr als das Zehnfache langsamer als die Vektorform des Algorithmus zur selben Aufgabenstellung.

```
t = 0:.01:10;
y = sin(t);
```

Wenn es nicht möglich ist die `for` Schleife durch einen Vektoralgorithmus zu ersetzen, so ist es zumindest möglich, den Vektor, in welchen die Ergebnisse abgelegt werden, zuvor mit einer Vektoroperation anzulegen. So läuft

```
i = 0; y = zeros(1,1001);
for t = 0:.01:10
        i = i+1;
     y(i) = sin(t);
end
```

um einiges schneller als die erste Version des Algorithmus, aber immernoch langsamer als die Vektorform. Der Grund hierfür ist, daß der Platz für den Ergebnisvektor nur einmal im Arbeitsspeicher festzulegen ist, so muß dieser nicht ständig in seinem Ausmaß und seiner Anordnung im Arbeitsspeicher nach jedem Iterationsschritt erneut bestimmt werden.

Ein zweiter Vorteil der vorzeitigen Dekleration aller Elemente von Vektoren und Matrizen besteht im Vorbeugen einer Speicherfragmentierung.

17 M-Files

17.1 Verwaltung von M-Files

Unter Windows-Oberflächen erfolgt die Erstellung von M-Files am besten in einem eigenen Fenster. Es ist aber auch möglich, von der Kommandozeile von MATLAB einen Editor aufzurufen. Dazu wird nach einem Ausrufezeichen der Befehl für den Editoraufruf genauso eingegeben, wie in einer Kommando-Shell. Es können auch beliebige andere Befehle des Betriebssystems auf diese Weise abgearbeitet werden.

```
! edit data.m
```

```
! dir
```

MATLAB stellt auch einige Befehle für die File-Verwaltung bereit:

dir	Anzeigen aller Files im momentanen Verzeichnis
what	Anzeigen aller M-Files im momentanen Verzeichnis
type	Ausgeben eines M-Files auf dem Bildschirm
delete	Löschen eines Files
chdir	Wechseln des momentanen Arbeitsverzeichnisses
diary	Protokollfile der MATLAB-Sitzung
lookfor	Schlüsselwortsuche in MATLAB-Files

17.2 Datenimport und -export

Es gibt verschiedene Wege zum Datenaustausch mit anderen Programmen. Möglicherweise können andere Programme bereits M-Files erzeugen oder verarbeiten. Dieser Abschnitt beschreibt aber Möglichkeiten des Datenimports und -exports aus der MATLAB - Umgebung heraus.

Weitergehende Informationen sind dem *External Interface to* MATLAB *guide* zu entnehmen.

17.2.1 Datenimport

Die beste Methode Daten zu importieren hängt ab: von der Menge der Daten, ob die Daten bereits rechentechnisch erfaßt sind. Hier ist eine Auswahl, aus der die geeignetste Methode auszuwählen ist.

- Eingabe als explizite Datenliste umschlossen von eckigen Klammern (als normaler Vektor unter MATLAB), nur für Datenmengen kleiner zwanzig Elemente akzeptabel.

- Eingabe wie beim vorherigen Punkt, jedoch nicht in der MATLAB - eingabezeile sondern mit Hilfe eines Editors in eine Datei. Geeignet für eine größere Anzahl von Elementen, die aber noch nicht rechentechnisch erfaßt sind.

- Laden der Daten von einem ASCII-File, d.h. die Zeilen sind durch Zeilenendezeichen ($0x0d0x0a$) gekennzeichnet und die einzelnen Elemente einer Zeile sind durch Leerzeichen getrennt. ASCII-Files können mit `load <Filename.*>` direkt in MATLAB importiert werden. Die Matrix in der die Daten abgelegt werden erhält die Bezeichnung *Filename*.

- Daten können auch mit den Grund E/A Funktionen `fopen` und `fread` eingelesen werden. Diese Methode ist nützlich, wenn die Daten von anderen Programmen erstellt wurden, welche eigene Datenformate verwenden.

- Entwickeln eigener MEX-Files zum Lesen der Daten bzw. schreiben eigener Programme in Fortran oder C ist sicherlich der aufwendigste Weg, jedoch nicht immer zu umgehen. Ab der Version 4.2 steht auch ene DDE-Schnittstelle für den Datenaustausch mit anderen Programmen zur Verfügung. Informationen hierzu sind der *MAT-file Subroutine Library* und dem *External Interface to* MATLAB *guide* zu entnehmen.

17.2.2 Datenexport

Hat man mit MATLAB Daten erzeugt, die in anderen Programmen weiterbearbeitet werden sollen, dann gibt es verschiedene Wege, diese Daten zu exportieren. Einige Methoden sind:

- Um kleine Datenmengen zu exportieren ist der `diary` Befehl geeignet, jedoch werden auch sämliche MATLAB-Befehle der Sitzung mit archiviert, was für eine spätere Dokumentation der Daten aber auch nütlich sein kann.

- Speichern der Daten im ASCII-Format mit Hilfe des `save` Kommandos und der Option `-ascii`, z.B.

```
A = rand(4,3);
save temp.dat A -ascii
```

kreiert ein ASCII-File mit dem Namen temp.dat, der Gestalt

```
0.0077    0.6868    0.5269
0.3834    0.5890    0.0920
0.0668    0.9304    0.6539
0.4175    0.8462    0.4160
```

- Mit den Grundfunktionen **fopen** und **fwrite** ist es möglich spezielle Formate der Datenspeicherung zu erzeugen, wie sie durch andere Programme benötigt werden.
- Entwickeln eigener MEX-Files zum Exportieren der Daten von MATLAB aus oder aber schreiben eigener Programme in Fortran oder C, welche die mit dem **save** Kommando abgespeicherten Daten so konvertieren, daß andere Programme auf sie zugreifen können, sind ziemlich aufwendig aber für große Datenmengen oder bei häufig anfallenden Datentransfer die einzig akzeptable Lösung. Informationen hierzu sind der *MAT-file Subroutine Library* und dem *External Interface to* MATLAB *guide* zu entnehmen.

17.3 File Ein/-Ausgabe

Die MATLAB - File Ein/-Ausgabe - Funktionen ermöglicht das Lesen und Schreiben von Daten in einem anderen als dem MATLAB eigenen Format.

Die MATLAB - File Ein/-Ausgabe - Funktionen basieren auf den Ein-/ Ausgabe - Funktionen der Programmiersprache C. Wenn sie mit C vertraut sind, werden sie sich schnell mit den E/A Funktionen von MATLAB zurechtfinden, anderenfalls ist ein C-Handbuch sicherlich hilfreich.

17.4 Öffnen und Schliessen von Files

Bevor Dateien beschrieben oder gelesen werden können, müssen diese erstmal geöffnet werden. dies geschieht mit **fopen** und einer Zugriffsrechtdefinition, z.B.

```
fid = fopen('pen.dat','r')
```

öffnet das Daten-File **pen.dat** zum Lesen. Mögliche Zugriffsrechte sind:

- 'r' für Lesen
- 'w' für Schreiben

- 'a' für Anhängen
- 'r+' für beides Lesen und Schreiben

Systeme wie VMS, welche zwischen Text und binären Dateien unterscheiden, benötigen noch zusätzliche Parameter bzgl. des Zugriffsrechtes, wie z.B. 'rb' zum Öffnen eines binären Files.

Die `fopen` Funktion liefert einen positiven Integerwert zurück, welcher als File-ID dient. MATLAB File Ein-/ Ausgabe - Funktionen nutzen diese ID um Files zu beschreiben, zu lesen oder zu schließen.

Wenn ein File nicht geöffnet werden kann, liefert `fopen` -1 als Ergebnis zurück. Bei einigen Systemen wird noch eine zweite zusätzliche Fehlerinformation bereitgestellt, folgender Aufruf

```
[fid, message] = fopen('pen.dat','r')
```

setzt `fid` zu -1 und belegt `message` mit einem String wie z.B.

```
No such file or directory.
```

Diese Meldung ist systemabhängig und wird auch nicht für alle Fehlerarten bereitgestellt. Eine dritte Möglichkeit ist die Informationen der `ferror` Funktion abzurufen.

Einmal geöffnet, ist das File verfügbar zum Schreiben oder Lesen. Nach dem Beenden von Lesen oder Schreiben ist es mit `fclose` wieder zu schließen.

```
status = fclose('fid')
```

schließt das File mit der File-ID `fid` und

```
status = fclose('all')
```

schließt alle offenen Files. Beide Aktionen liefern 0, wenn sie erfolgreich waren, anderenfalls wird `status` zu -1 gesetzt.

17.5 Lesen binärer Daten

Die `fread` Funktion liest binäre Daten-Files. In der einfachsten Form wird das gesamte Daten-File in einer Matrix abgelegt.

```
fid = fopen('penny.mat','r');
A = fread(fid);
status = fclose(fid);
```

liest alle Daten des Files `penny.mat` als ASCII Zeichen (Typ char) in die Matrix `A` ein.

Zwei optionale Argumente von `fread` geben Kontrolle über die Anzahl der zu übernehmenden Werte und ihre Genauigkeit.

```
fid = fopen('penny.mat','r');
A = fread(fid,100);
status = fclose(fid);
```

liest nur die ersten 100 Elemente in den Spaltenvektor A. Ersetzt man die Zahl 100 durch die Matrixdimension [10,10], werden die selben 100 Elemente in einer Matrix der Dimension 10 x 10 abgelegt.

Die Argumente zur numerischen Genauigkeit bestimmen die Anzahl der Bits die pro Wert gelesen werden, sowie die Interpretation dieser Bits als ASCII-Zeichen, Integer- oder Gleitpunktzahl. Die numerische Genauigkeit ist hardwarespezifisch, und ist dem Handbuch zum Rechner zu entnehmen.

MATLAB offeriert einen großen Bereich verschieden genauer Wertebereiche. Einige sind

- `char` und `uchar` für Daten des Typs `char` mit oder ohne Vorzeichen (typisch sind 8 Bit)
- `short` und `long` für kurze bzw. lange Integerwerte (typisch sind 16 und 32 Bit)
- `float` und `double` für einfach- und doppeltlange Gleitpunktzahlen (typisch sind 32 bzw. 64 Bit)

Diese Zahlenkonventionen sind den Programmiersprachen Fortran und C entnommen. Eine komplette Liste ist dem MATLAB *Reference Guide* oder jedem besseren C Buch zu entnehmen.

Wenn `fid` die ID Zahl eines offenen Files mit Gleitpunktzahlen ist, so liest

```
A = fread(fid,10,'float')
```

die ersten 10 Werte dieser Datei ein.

Das folgende Beispiel öffnet das File, welches den Hilfetext der `fread` Funktion enthält, und zeigt diesen anschließend auf dem Bildschirm an.

```
freadid = fopen('fread.m','r');
F = fread(freadid,'uchar');
disp(setstr(F'))
status = fclose(freadid);
```

17.6 Schreiben binärer Daten-Files

Die `fwrite` Funktion schreibt die Elemente einer Matrix mit der spezifizierten Genauigkeit in ein File, der Rückgabewerte der Funktion ist die Anzahl der geschriebenen Werte, z.B.

```
fwriteid = fopen('magic5.bin','w');
count = fwrite(fwriteid,magic(5),'integer*4');
status = fclose(fwriteid);
```

erzeugt ein 100 Byte großes binäres Daten-File, das die 25 Integerwerte der 5 x 5 Matrix speichert. Die Integergröße von 4 Byte wurde durch die Angabe 'integer*4' erzwungen.

17.7 Zugriff auf eine bestimmte Position eines Files

Die `fseek` und `ftell` Funktionen erlauben es, eine ganz bestimmte Position festzulegen oder zu erfragen, an welcher die nächste Eingabe- bzw. Ausgabeoperation stattzufinden hat.

Die `ftell` Funktion gibt den Offset in Bytes an, der die Position im File kennzeichnet, wie sie durch die Argumente spezifiziert ist, die an `ftell` übergeben wurden. Die `fseek` Funktion positioniert den File-Positions-Zeiger neu. So ist es möglich Daten zu überspringen und zu einem früheren Teil des Files zu gelangen. Die Argumente sind: die File-ID und ein positiver oder negativer Offset, um zu kennzeichnen ob vor- oder zurückzuspringen ist, sowie ein Ausgangspunkt von welchem aus der Sprung startet. Der Ausgangspunkt kann die aktuelle Position ('cof') oder das Ende des Files ('bof' oder 'eof') sein.

In dem folgenden Beispiel schreiben die ersten 4 Anweisungszeilen die Zahlen 1 bis 5 als 2-Byte-Integers in eine File namens `five.bin`. Dann wird mit `fseek` über die ersten 6 Byte gesprungen, welche die Werte 1,2 und 3 enthalten, und `fid` auf den vierten Wert gesetzt, so daß `fread` den Wert 4 einliest. Hierdurch wird die Position des Filezeigers um 2 Bytes verschoben, die Position die die Funktion `fseek` zurückliefert ist somit 8, gemessen vom Anfang des Files. Der zweite Aufruf von `fseek` setzt die Position um 4 Byte zurück, so daß er auf das erste Byte des Wertes 3 zeigt.

```
A =[1:5];
fid = fopen('five.bin','w');
fwrite(fid,A,'short');
status = fclose(fid);
```

```
fid = fopen('five.bin','r');
status = fseek(fid,6,'bof');
four = fread(fid,1,'short');
position = ftell(fid);
status = fseek(fid,-4,'cof');
three = fread(fid,1,'short');
status = fclose(fid);
```

17.8 Schreiben formatierter Texte

Die `fprintf` Funktion konvertiert Daten zu ASCII-Zeichen und speichert diese in Zeichenkettenvariabeln oder Files ab. Eine Formatierungszeichenkette mit Text und Formatierungszeichen wird der `fprintf` Funktion als erstes Argument übergeben, es erzeugt die Textpassagen der Zeichenkette und regelt über die Formatierungszeichen, die Einbindung der folgenden Argumente (Skalare, Matrizen) in die Zeichenkette.

Formatierungszeichen sind

- %e für exponetionelle Darstelungsweise
- %f für Festkommanotation
- %g für automatische Auswahl des kürzeren Notation %e bzw. %f

Optionale Zahlenangaben sind möglich um die minimale Feldbreite sowie die Genauigkeit zu definieren, z.B.

```
x = 0:.1:1;
y = [x; exp(x)];
fid = fopen('exptable.txt','w');
fprintf(fid,'Exponentialfunktion\n\n');
fprintf(fid,'%6.2f  %12.8f \n',y);
status = fclose(fid)
```

erzeugt ein Text-File mit einer kurzen Tabelle der Exponentialfunktion. Zuerst wird die Überschrift mit `fprintf(fid,'Exponentialfunktion\n\n');` erzeugt, `\n` steht für den Zeilenumbruch. Der zweite Aufruf von `fprintf` erzeugt die Tabelle selbst, die Formatdefinitionen sind hierbei

- ein sechsstelliger Festkommawert inklusive zweier Dezimalstellen
- Zwei Leerzeichen

- ein zwölfstelliger Festkommawert inklusive acht Dezimalstellen

Die Elemente der Matrix `y` werden so in Spalten konvertiert, daß je eine Spalte zu einem Zahlenwert-Platzhalter der Zeichenkette zur Formatdefinition korrespondiert.

Die Abgeleitete Funktion `sprintf` übergibt die erzeugte Zeichenkette an eine Variable anstatt in ein File oder auf den Bilschirm, z.B.

```
root2=sprintf('Die Quadratwurzel von %f ist %10.8e. \n', ...
        2,sqrt(2));
```

17.9 Lesen formatierter Texte

MATLAB's Texteinlesefunktion `fscanf` ist ähnlich `fprintf`, `fscanf` nimmt als Argumente die File-ID einer geöffneten Datei sowie eine Zeichenkette zur Formatdefinition, welche einfache ASCII- Zeichen und Konversionsvereinbarungen enthält, mögliche Vereinbarungen sind

- %s um eine Zeichenkette aufzunehmen -> sucht alle ASCII-Zeichen aus der Zeichenkette heraus
- %d um Dezimalzahlen aufzunehmen
- %d um Gleitpunktzahlen aufzunehmen

Die im letzten Abschnitt erzeugt Datei `exptable.txt` kann mit den folgenden Befehlen eingelesen werden:

```
fid = fopen('exptable.txt','r');
title = fscanf(fid,'%s');
[table,count] = fscanf(fid,'%d %f');
status = fclose(fid)
```

Die erste Zeile liefert die File-ID. Der zweite Aufruf von `fscanf` entnimmt abwechselnd eine Dezimal- und eine Gleitpunktzahl bis zum Ende des Files. Die Anzahl der übernommenen Werte wird in `count` gespeichert.

Optionale Argumente ermöglichen es die Anzahl der einzulesenden Matrixelemente zu begrenzen, z.B.

```
A = fscanf(fid,'%5d',100);
```

liest die ersten 100 Dezimalwerte in einen Spaltenvektor ein und

```
A = fscanf(fid,'%5d',[10,10]);
```

liest die ersten 100 Dezimalwerte in eine 10 x 10 Matrix ein.

An die abgeleitete Funktion **sscanf**, wird eine Zeichenkette übergeben und nicht der Name einer Textdatei, z.B.

```
S = '2.7183  3.1416';
A = sscanf(S,'%f')
```

erzeugt einen Zweielementevektor, welcher genähert e und pi enthält.

18 Ausgabeformate

Berechnungen in MATLAB erfolgen immer mit doppelter Genauigkeit. Der Nutzer hat jedoch Einfluß auf die Darstellungsweise der Ergebnisse. Die folgenden Kommandos beeinflussen die Ausgabe:

format short	Festkomma mit 4 Dezimalstellen (Voreinstellung)
format long	Festkomma mit 14 Dezimalstellen
format short e	Fließkomma mit 4 Dezimalstellen
format long e	Fließkomma mit 15 Dezimalstellen

Das eingestellte Format bleibt bis zur nächsten Änderung erhalten. Der Befehl **format compact** unterdrückt die zusätzliche Ausgabe von Leerzeilen. Er beeinflußt die anderen Darstellungsformen nicht.

19 Graphiken

MATLAB bietet umfangreiche Möglichkeiten zur Darstellung von 2D- und 3D-Plots. Wie leistungsfähig MATLAB hierbei ist, läßt sich aus den Demonstrationsprogrammen ersehen (**demo**).

Das **plot**-Kommando erzeugt x-y-Graphiken mit linearer Achsenteilung. Die Vektoren x und y müssen dabei die gleiche Länge haben.

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

Der Plot-Befehl eröffnet dazu ein Graphikfenster und zeichnet den Graphen ein. Soll der Linientyp des Graphen geändert werden, so ist ein drittes Argument erforderlich. Für eine Strichpunktlinie ist

```
plot(x,y,'-.')
```

als Kommando einzugeben. Sollen mehrere Graphen in einem Bild dargestellt werden, so sind weitere Argumente erforderlich:

```
plot(x,y1,'b-.',x,y2,'-');
```

Dieser Befehl zeichnet einen Graphen mit einer blauen Strich-Punkt-Linie und einen mit einer durchgehenden gelben Linie ein. Gelb ist die Standardfarbe, die verwendet wird wenn kein anderes Farbattribut gesetzt ist. Möchte man die Graphen in unterschiedlichen Fenstern darstellen, so kann mit dem Befehl `figure(n)` das n-te Fenster als aktuelles Fenster gesetzt werden.

```
figure(1);
plot(x,y1);
figure(2);
plot(x,y2);
```

MATLAB gestattet auch die Darstellung von 3D-Plots. Mit Hilfe des Befehls `mesh(z)` erzeugt man einen 3D-Plot der Elemente der Matrix `z`. Die x- bzw. y-Koordinaten sind dabei die Indizes der Matrix. Die z-Koordinate ist der entsprechende Wert der Matrix für gegebene Indizes.

Um den Graphen einer Funktion der Form $z = f(x, y)$ über einem rechteckigen Gebiet zu zeichnen, definiert man zuerst Vektoren `xx` und `yy`, welche die Einteilung der Seiten des Rechtecks (Gitterpunkte) definieren. Mit Hilfe der Funktion `meshdom` berechnet man dann eine Matrix `x`, deren Zeilen aus dem Vektor `xx` aufgebaut sind und die so viele Spalten besitzt wie `yy` Elemente hat. Und zusätzlich noch eine Matrix `y`, deren Spalten aus dem Vektor `yy` aufgebaut sind und die so viele Zeilen hat wie der Vektor `xx` Elemente:

```
[x,y]=meshdom(xx,yy);.
```

Dann bestimmt man die Matrix z, indem die Funktion f elementweise für die Matrizen `x` und `y` berechnet wird. Um z.B. die Funktion $z = e^{-x^2-y^2}$ über dem Gebiet [-2,2]×[-2,2] zu berechnen, können die folgenden Zeilen benutzt werden:

```
xx=-2:.1:2;
yy=xx;
[x,y] = meshdom(xx,yy);
z = exp(-x.^2 - y.^2);
mesh(z);
```

Die drei ersten Zeilen können auch durch die folgende ersetzt werden:

```
[x,y] = meshdom(-2:.1:2, -2:.1:2);
```

Weitere Details zu den hier erwähnten Befehlen sind den Hilfetexten zu entnehmen.

20 Fehlersuche

MATLAB findet Syntaxfehler während des Kompilierens. Diese Fehler sind relativ problemlos zu beheben. Problematischer sind die Fehler die erst während der Laufzeit eines Programms auftreten. Ein Hauptproblem ist, wenn fehlerbedingt eine Funktion abgebrochen wird und zum MATLAB - Prompt zurück gesprungen wird, so sind die lokalen Variabeln dieser Funktion im Arbeitsspeicher nicht mehr verfügbar und stehen zur Fehleranalyse nicht mehr zur Verfügung.

Mögliche Methoden zur Fehlersuche sind:

- Entfernen der Semikolons zur Unterdrückung der Zwischenergebnisausgabe
- Einfügen von Tastaturabfragen zum gezielten Programmabbruch
- Herauskommentieren der Funktionsdeklarationszeile des M-Files, so daß die Funktion wie ein normales Skript-File abgearbeitet wird und die Variabeln auch nach dem fehlerbedingten Abbruch noch im Arbeitsspeicher von MATLAB vorhanden sind.
- Nutzen des MATLAB - Debuggers.

20.1 Debugger - Kommandos

dbstop	Unterbrechungspunkt setzen
dbclear	Unterbrechungspunkt entfernen
dbcont	Fortsetzen mit der Abarbeitung
dbdown	Verändern des lokalen Arbeitsspeicherbereiches
dbstack	Auflisten: Wer wen aufruft
dbstatus	Auflisten aller Unterbrechungspunkte
dbstep	Ausführen einer oder mehrerer Zeilen
dbtype	Auflisten des M-Files mit Zeilennummern
dbup	Verändern des lokalen Arbeitsspeicherbereiches
dbquit	Beenden des Debuggermodus

20.2 Arbeiten mit dem Debugger

Wenn bei der Abarbeitung eines M-Files ein Fehler auftritt, sind mit Hilfe der Debugger - Kommandos Unterbrechungspunkte zu setzen. Nach der Unterbrechung der Programmabarbeitung wird die Zeile, bei der abgebrochen wurde, angezeigt, gefolgt vom Tastatur-Prompt. Es können alle zulässigen MATLAB - Kommandos eingegeben werden.

Bei der Arbeit mit dem MATLAB - Debugger ist zu beachten:

- Debugger - Kommandos arbeiten nur mit Funktions-Files nicht mit M-Skripts
- Die M-File Unterbrechungsinformationen sind eng verknüpft mit dem kompilierten M-File. Wenn das M-File aus dem Arbeitsspeicher entfernt wird, egal ob durch ein Clear - Kommando oder durch Editieren, sind auch alle Unterbrechungspunkte gelöscht.

20.3 Eine Debugging - Sitzung

1. Erstellen zweier M-Files `test.m` und `test1.m`

```
test.m
function a = test(b)
c = sqrt(b)*cos(b);
a = test1(b,c)
test1.m
function a = test1(b,c)
q = cond(b);
[w,e] = eig(c);
a = w*q;
```

2. Mit Hilfe des `dbtype` Kommandos sind nun die beiden obigen sowie die MATLAB - Funktion `cond.m` mit Zeilennummern auszugeben

```
dbtype test
1   function a = test(b)
2   c = sqrt(b)*cos(b);
3   a = test1(b,c)

dbtype test1
1   function a = test1(b,c)
```

```
2   q = cond(b);
3   [w,e] = eig(c);
4   a = w*q;

dbtype cond
1   function y = cond(x)
2   %COND       Matrix condition number.
3   %   COND(X) is the ratio of the largest singular
        value of X
4   %   to the smallest, which is the condition number
        of X in 2-norm.
5   %
6   %   See also RCOND, NORM, CONDEST, NORMEST.
7
8   %   J.N. Little 11-15-85
9   %   Revised 3-9-87 JNL, 2-11-92 LS.
10  %   Copyright (c) 1984-93 by The MathWorks, Inc.
11
12  if length(x) == 0  % Handle null matrix
13      y = NaN;
14      return
15  end
16  if issparse(x)
17      error('Matrix must be non-sparse.')
18  end
19  s = svd(x);
20  if any(s == 0)   % Handle singular matrix
21      disp('Condition is infinite')
22      y = Inf;
23      return
24  end
25  y = max(s)./min(s);
```

3. Erzeugen einer Zeichenkettenvariabeln

```
hi = 'hello again'
```

20.3.1 Unterbrechungspunkte festlegen

Als erstes wird das `dbstop` - Kommando auf das File `test.m` angewandt.

```
dbstop in test
```

Dieser Ausdruck sorgt dafür, daß die Ausführung von `test` vor der ersten Zeile stoppt, die ausführbare MATLAB - Kommandos enthält, der selbe Effekt ist mit `dbstop at 2 in test` zu erreichen. Nun wird ein Unterbrechungspunkt in die Zeile 19 des Files `cond.m` gesetzt.

```
dbstop at 19 in cond
```

20.3.2 Anzeigen des Stacks während des Programmablaufs

1. Nach dem setzen der Unterbrechungspunkte ist `test` zu starten, wie festgelegt stoppt die Abarbeitung bei der Zeile 2

   ```
   test(magic(3))
   2   c = sqrt(b)*cos(b);
   ```

2. Nach dem Stop der Funktion, können mit `dbstack` die Funktionsaufrufe bis zur Unterbrechung angezeigt werden.

   `dbstack In` *Pfadname*`\test.m at line 2`

3. Fortsetzen der Abarbeitung der Funktion mit `cond` bis zum nächsten Unterbrechungspunkt.

   ```
   dbcont
   19  s = svd(x);
   ```

4. Erneutes Anzeigen des Stacks.

   ```
   dbstack
   In Pfadname\cond.m at line 19
   In Pfadname\test1.m at line 2
   In Pfadname\test.m at line 3
   ```

20.3.3 Prüfen der Variabeln des Arbeitsspeichers

1. Prüfen der Variabeln des aktuellen Arbeitsspeichers von `cond`

   ```
   who
   Your variables are:
   x         y
   ```

2. Ausgabe des Inhaltes von x.

```
x
x =
     8     1     6
     3     5     7
     4     9     2
```

20.3.4 Prüfen der Variablen nach Ausführung der nächsten Zeile

Ausführen der nächsten Zeile mit Hilfe von `dbstep`, sowie abrufen von Informationen über den Arbeitsspeicher, wie gehabt.

```
dbstep
20  if any(s == 0)   % Handle singular matrix

s
s =
   15.0000
    6.9282
    3.4641

who
Your variables are:
s         x         y
```

20.3.5 Verändern des Arbeitsspeichers

1. Ändern des Arbeitsspeichers der Funktion `cond`

   ```
   dbup
   In workspace belonging to Pfadname\test1.m.
   ```

2. Prüfen des Arbeitsspeichers von `test1`.

   ```
   who
   Your variables are:
   a         b         c
   ```

3. Testen der Inhalte der Variablen.

   ```
   a
   a =
   ```

```
     []

b
b =

     8     1     6
     3     5     7
     4     9     2

c
c =
   -3.0026   -0.4199    2.4503
   -4.1951   -0.8405    2.2478
   -4.1854    0.6431    3.5935
```

4. Wechseln in den Arbeitsspeicher der Funktion `test`, die aufrufende Funktion der Funktion `test1`.

```
dbup
In workspace belonging to Pfadname\test.m.
```

5. Wechseln in den Hauptarbeitsspeicherbereich von MATLAB , und Anzeigen der aktuellen Variabeln.

```
dbup
In base workspace.

who
Your variables are:
hi
```

20.3.6 Erzeugen neuer Variablen

1. Eingeben einer neuen Variable.

```
new_var = 123
new_var =
   123
```

2. Wechseln in den Arbeitsspeicherbereich von `test`, und Prüfen des Inhaltes.

```
dbdown
In workspace belonging to Pfandname\test.m.
whos
 Name    Size    Elements  Bytes  Density  Complex
   a    0 by 0       0        0      Full       No
   b    3 by 3       9       72      Full       No
   c    3 by 3       9       72      Full       No
Grand total is 18 elements using 144 bytes
```

3. Anzeigen der Variable `b`

```
b
b =
8 1 6
3 5 7
4 9 2
```

4. Wechseln zu `test1` und dann zu `cond`

```
dbdown
In workspace belonging to Pfandname\ test1.m.
dbdown
In workspace belonging to Pfandname\ cond.m.
```

20.3.7 Schrittweises Abarbeiten einer Funktion

1. Abarbeiten der einzelnen Schritte der Funktion `cond` und anschließend der Funktion `test1`.

```
dbstep
25  y = max(s)./min(s);

dbstep
End of M-file function cond.

dbstep
3   [w,e] = eig(c);
```

2. Anzeigen des Stacks

```
dbstack
In Pfandname\test1.m at line 3
In Pfandname\test.m at line 3
```

3. Nächster ausführbarer Schritt von `test1`

```
dbstep
4   a = w*q;
```

4. Fortführen der Abarbeitung bis zur nächsten Unterbrechungsmarke bzw. bis zum Rücksprung der Funktion zum MATLAB-Prompt.

```
dbcont
ans =
   2.0428            -1.5040 - 1.5431i  -1.5040 + 1.5431i
   3.6832             0.7259 - 0.3702i   0.7259 + 0.3702i
   1.0056            -2.0706 - 3.0258i  -2.0706 + 3.0258i
```

20.3.8 Anzeigen des MATLAB - Arbeitsspeichers

Prüfen des Speichers nach Beendigung der Funktion.

```
whos
   Name      Size   Elements     Bytes  Density  Complex
    ans    3 by 3         9       144     Full     Yes
     hi    1 by 11       11        88     Full     No
new_var    1 by 1         1         8     Full     No
Grand total is 21 elements using 240 bytes
```

Betrachten wir noch die während des Debuggens erzeugte Variable `new_var`.

```
new_var
new_var =
123
```

20.3.9 Stoppen des Debuggingprozesses

Der Debugging-Prozeß kann jederzeit mit `dbquit` abgebrochen werden.

1. Prüfen der Unterbrechungspunkte von `test`.

```
dbstatus test
Breakpoints for test are on lines 2.
```

2. Erneutes Starten des Programmes `test`

```
test(magic(3))
2   c = sqrt(b)*cos(b);
```

3. Abrechen der Fehler suche mit `dbquit`

Die Unterbrechungspunkte verbleiben auch nach `dbquit` im Arbeitsspeicher.

```
dbstatus test
Breakpoints for test are on lines 2.
```

Literatur

[1] Sigmon, K.; MATLAB Primer; Department of Mathematics, University of Florida, 1989

[2] MATLAB Reference Guide, The MathWorks Inc., 1992

[3] Control System Toolbox, The MathWorks Inc., 1992

[4] MATLAB User's Guide, The MathWorks Inc., 1992

[5] Jumar, U.; Vorlesungsunterlagen zu „Grundlagen technischer Systeme“ (für Studenten des 5. Semesters)

A Befehlsübersicht

Es gibt noch viele Möglichkeiten von MATLAB, die in dieser Anleitung nicht enthalten sind. Um wenigstens einen schnellen Überblick zu den gängigsten Funktionen zu geben, folgt hier eine nach Themengruppen geordnete Zusammenstellung von wichtigen MATLAB-Befehlen.

Allgemeine Funktionen	
help	Hilfebefehl
demo	Demoprogramm
info	Informationen über MATLAB
lookfor	Schlüsselwortsuche
path	Suchpfad von MATLAB
type	Auflisten von M-Files
which	Verzeichnis eins M-Files finden

Verwaltung von Variablen	
clear	löschen von Variablen
disp	Matrix oder Text anzeigen
length	Länge eines Vektors
load	Variablen aus File laden
pack	Heap defragmentieren
save	Speichern von Variablen
size	Dimension einer Matrix
who	Anzeigen aller Variablen
whos	wie who, mit Angabe der Größe

Dateiverwaltung und Betriebssystem	
cd	Aktuelles Verzeichnis wechseln
delete	File löschen
diary	Protokoll der MATLAB-Sitzung
dir	Verzeichniseinträge listen
getenv	Umgebungsvariablen auslesen
!	DOS-Kommando ausführen

Kommandofenster	
clc	Fenster löschen
echo	Befehle in Script-Files anzeigen
format	Ausgabeformat setzen
home	Cursor in linke obere Ecke
more	seitenweise Ausgabe einstellen

Starten und Beenden von MATLAB	
matlabrc	Master startup M-File
quit	Beenden von MATLAB
exit	Beenden von MATLAB
startup	wird beim Start aufgerufen

Operatoren und Sonderzeichen	
+	Plus
-	Minus
	Matrixmultiplikation
.*	elementweise Multiplikation
^	Matrixpotenz
^	elementweises Potenzieren
kron	Kronecker tensor Produkt
\	Linksdivision
/	Rechsdivision
./	elementweise Division
...	Fortsetzung einer Zeile
'	Transponieren
=	Zuweisung
==	gleich
<	kleiner als
>	größer als
&	logisches Und
|	logisches Oder
~	logisches Nicht
xor	logisches exklusives Oder

Logische Funktionen	
all	Wahr, wenn alle Elemente des Vektors wahr sind
any	Wahr, wenn ein Element des Vektors wahr ist
exist	Prüft, ob Variable existiert
find	Findet Indices von Elementen ungleich Null
finite	Wahr für endliche Elemente
isempty	Wahr für leere Matrizen
isieee	Wahr für IEEE Fließkomma Arithmetik
isinf	Wahr für unendliche Werte
isnan	Wahr für Not-A-Number
issparse	Wahr für sparse Matrizen
isstr	Wahr für Textmatrizen

MATLAB als Programmiersprache	
eval	Berechnen einer Zeichenkette mit MATLAB-Ausdrücken
feval	Funktion in Zeichenkette berechnen
function	neue Funktion definieren
global	Definiert globale Variablen
nargchk	Anzahl der Input-Argumente überprüfen

Programmablaufsteuerung	
break	beendet Schleife
else	in Verbindung mit IF benutzen
elseif	in Verbindung mit IF benutzen
end	Ende des Bereichs von IF, WHILE und FOR
error	Fehlermeldung anzeigen und Funktion beenden
for	bestimmte Anzahl von Schleifendurchläufen
if	bedingte Programmverzweigung
return	Rückkehr zur aufrufenden Funktion
while	Wiederholungsanweisung

Interaktive Eingabe	
input	Prompt für Eingabe
keyboard	behandelt Tastatur wie Script-File
menu	Nutzermenu erstellen
pause	Warten auf Nutzereingabe

Elementare Matrizen	
eye	Einheitsmatrix
linspace	linear gestaffelter Vektor
logspace	logarithmisch gestaffelter Vektor
meshgrid	X und Y Felder für 3D-Plots
ones	Matrix aus Einsen
rand	gleichverteilte Zufallswerte
randn	normalverteilte Zufallswerte
zeros	Matrix aus Nullen

Spezielle Variablen und Konstanten	
ans	letzter Rückgabewert
computer	Computertyp
eps	relative Genauigkeit von Fließkommawerten
flops	Zahl der Fließkommaoperationen
i, j	imaginäre Einheit
inf	Unendlich
NaN	Not-A-Number
nargin	Zahl der Inputargumente einer Funktion
nargout	Zahl der Outputargumente einer Funktion
pi	3.1415926535897
realmax	größte Fließkommazahl
realmin	kleinste Fließkommazahl

Zeit und Datum	
clock	Uhrzeit
cputime	verbrauchte CPU-Zeit
date	Datum
etime	Zeitdifferenz berechnen

Elementare Math. Funktionen	
abs	absoluter Wert
acos	inverser Kosinus
acosh	inverser Kosinushyperbolikus
angle	Phasenwinkel
asin	inverser Sinus
asinh	inverser Sinushyperbolikus
atan	inverser Tangens
atan2	inverser Tangens (4 Quadranten)
atanh	inverser Tangenshyperbolikus
ceil	Aufrunden
conj	konjugierte komplexe Zahl
cos	Kosinus
cosh	Kosinushyperbolikus
exp	Exponential
fix	in Richtung Null runden
floor	Abrunden
imag	Imaginärteil
log	natürlicher Logarithmus
log10	dekadischer Logarithmus
real	Realteil
rem	Divisionsrest
round	zur nächsten ganzen Zahl runden
sign	Signum-Funktion
sin	Sinus
sqrt	Quadratwurzel
tan	Tangens
tanh	Tangenshyperbolikus

Matrix Analysis	
cond	Konditionszahl einer Matrix
det	Determinante
norm	Matrix- oder Vektornorm
null	Null-Raum
orth	Orthogonalisierung
rcond	LINPACK's reziproker Konditionsschätzer
rank	Rang einer Matrix
trace	Spur einer Matrix

Zeitantworten von zeitdiskreten Systemen	
dimpulse	Impulsantwort
dinitial	Reaktion auf Anfangswertvorgabe
dlsim	Simulation für beliebige Eingangsgröße
dstep	Sprungantwort
filter	vergl. dlsim

Zeitantworten kontinuierlicher Systeme	
impulse	Impulsantwort
initial	Reaktion auf Anfangswertvorgabe
lsim	Simulation für beliebige Eingangsgröße
step	Sprungantwort

Frequenzbereichsbefehle	
bode	Bode-Diagramm
nyquist	Ortskurve
nichols	Nichols-Diagramm
margin	Verstärkungs- und Phasenrand
ngrid	Gitter für Nichols-Diagramm

Modelleigenschaften	
ctrb	Steuerbarkeitsmatrix
obsv	Beobachtbarkeitsmatrix
printsys	Darstellung von Übertragungsfunktionen
tzero	Übertragungsnullstellen
esort	sortieren von Eigenwerten

Modellbildung	
parallel	Parallelschaltung
series	Reihenschaltung
feedback	Rückkopplung
cloop	Antiparallelschaltung
blkbuild	Zustandsraummodell aus Blockdiagramm
connect	Blockdiagramme verbinden
ssselect	Auswahl von Subsystemen
conv	Polynommultiplikation
rmodel	zufälliges Modell erzeugen
drmodel	zufälliges Modell (zeitdiskret)
pade	Pade-Approximation von Totzeiten

Modellformen	
canon	kanonische Modellformen
ctrbf	Regelungsnormalform
obsvf	Steuerungsnormalform
ss2ss	Ähnlichkeitstransformation

Pole und Nullstellen	
pzmap	PN-Bild
rlocus	Wurzelortskurve
sgrid	Gitter für rlocus

B Kontrollfragen

Die Fragen sind eine Zusammenstellung von Prüfungsfragen, die im Zusammenhang mit der Einführung in die Nutzung regelungstechnischer Programmsysteme gestellt wurden.

1. Nennen Sie einige renommierte Softwarebibliotheken, die für die Systemanalyse und -technik (weltweit) Bedeutung erlangt haben! Gehen Sie anhand der historischen Entwicklung auf den ursprünglichen Leistungsumfamg dieser Bibliotheken und die vorzugsweise verwendeten Programmiersprachen ein!

2. Unternehmen Sie den Versuch einer Einordnung des Programmsystems MATLAB in den Softwarefundus des Gebietes „CAE von Systemen“! Gehen Sie auf die Intention der Entwickler von MATLAB ein!

3. Nennen und beschreiben Sie die verschiedenen von MATLAB gebotenen Varianten zur Inversion einer Matrix bzw. zur Lösung linearer Gleichungssysteme!

4. Was versteht man unter der Kondition eines mathematischen Problems? Erläutern Sie die Frage anhand des Standardproblems der linearen Algebra, der Lösung von linearen Gleichungssystemen! Ziehen Sie Schlußfolgerungen!

5. Erläutern Sie die Problematik schlecht konditionierter Aufgabenstellungen anhand des berühmten Polynombeispiels von WILKINSON (Polynom mit den Wurzeln 1,2,...20)! Welche Folgerungen ergeben sich für die Arbeit mit Polynomen?

6. Was verstehen Sie unter der Konditionszahl einer Matrix? Wie ist sie üblicherweise definiert, was bringt sie zum Ausdruck? Wie wird die Konditionszahl in MATLAB berechnet?

7. Welche Bedeutung haben Vektor- und Matrixnormen? Nennen Sie geläufige Normdefinitionen für Vektoren! Wann sind eine Vektor- und eine Matrixnorm miteinander verträglich?

8. Wann bezeichnet man einen numerischen Algorithmus als numerisch stabil (als guten Algorithmus)? Inwiefern sind die Kondition eines mathematischen Problems und die Güte eines Algorithmus' zu seiner Lösung zwei wohl voneinander zu unterscheidende Aspekte?

9. Welche prinzipiell unterschiedlichen Varianten zur Nullstellenbestimmung eines Polynoms haben in der numerischen Mathematik Verbreitung gefunden? Charakterisieren Sie diese Varianten und nennen Sie jeweils Vor- und Nachteile.

10. Was verstehen Sie unter der sog. Begleitmatrix eines Polynoms? Welche Gestalt hat diese Matrix? Worin liegt ihre Bedeutung?

11. Beschreiben Sie den Weg der Bestimmung von Nullstellen eines Polynoms über die Eigenwerte seiner Begleitmatrix! Klären Sie die Begriffe „charakteristisches Polynom einer Matrix“ und „charakteristische Gleichung“!

12. Welche Rolle spielen die Eigenwertberechnungen von Matrizen bei der Analyse von Systemen? Gehen Sie auf die Frage der Stabilität ein!

13. Begründen Sie die Zweckmäßigkeit der Verwendung einer Zustandsraumbeschreibung für die Analyse und Synthese von Systemen! Gehen Sie dabei auch auf Fragen der programmtechnischen Unterstützung durch MATLAB ein!

14. Welche Möglichkeiten gibt es, aus einer gegebenen Übertragungsfunktion auf einfachem Wege ein Zustandsraummodell mit gleichem Klemmenverhalten zu erhalten? Nennen Sie wichtige Normalformen der Zustandsbeschreibung! Geben Sie für eine ausgewählte Normalform das Aussehen der Matrizen und Vektoren des Zustandsraummodells an!

15. Beschreiben Sie das prinzipielle Vorgehen zur Berechnung der Zeitantworten eines gegebenen Systems in Zustandsraumbeschreibung! Geben Sie die allgemeine Lösung der Vektor-DGL an!

16. Welcher Weg wird in der MATLAB-Funktion „C2D“ beschritten, um ein kontinuierliches Zustandsraummodell in ein zeitdiskretes zu überführen? Nennen Sie (ohne die vollständige Ableitung wiederzugeben) die Vorteile dieses Weges!