Einführung in MATLAB

W. HUYER
2006

Die folgende Beschreibung bezieht sich auf MATLAB 7.1 (R14), Service Pack 3, unter Linux.

Inhaltsverzeichnis

1 Grundlegende MATLAB-Befehle

1.1 Starten und Beenden von MATLAB und Beschreibung der MATLAB-Oberfläche

MATLAB bietet zwei Nutzungsebenen: einfache Aufgaben löst man am besten interaktiv. In diesem Modus wird jede Anweisung unmittelbar nach der Eingabe vom MATLAB-Kern interpretiert und ausgeführt. Für komplexere Probleme kann man Programme und Unterprogramme in der Art von strukturierten Programmiersprachen schreiben mit dem Unterschied, dass sie nicht compiliert werden müssen, sondern während der Ausführung interpretiert werden.

MATLAB wird dadurch gestartet, dass man sich eine *Shell* (ein Fenster analog der „Eingabeaufforderung“ in Windows) öffnet und darin `matlab&` eingibt. (Der Abschluss des Befehls mit `&` hat den Vorteil, dass die Shell, aus der MATLAB gestartet wurde, für andere Dinge verwendet werden kann – der Prozess wird „in den Hintergrund geschoben“ –, während sie mit dem Befehl `matlab` blockiert bleibt.) Nach dem Erscheinen des MATLAB-Logos und dessen Verschwinden öffnet sich die MATLAB-Oberfläche in einem separaten Fenster.

Die MATLAB-Oberfläche sieht im Default-Format folgendermaßen aus. Unterhalb der „Überschrift“ MATLAB hat die MATLAB-Oberfläche drei Menüzeilen, und darunter enthält sie weitere Fenster. Die erste Menüzeile besteht aus Wörtern, beginnend mit dem Wort *File*, die zweite enthält v.a. Symbole, beginnend mit einem weißen Blatt, und die dritte, beginnend mit *Shortcuts*, enthält direkte Links zur Help-Facility.

Das rechte Fenster ist das *Command Window*, der wichtigste Teil von MATLAB, in dem die Ein- und Ausgabe erfolgen. Wenn die Kopfzeile „Command Window“ blau hinterlegt ist, ist das Command Window aktiviert, und man kann etwas eintippen. Im Command Window sind nach dem Start ein paar Zeilen mit der Versionsnummer von MATLAB und dem Copyright zu sehen, es wird auf die Help-Facility verwiesen, und der Doppelpfeil `>>` (MATLAB-Systemprompt) zeigt an, dass das Command Window zur Verarbeitung einer Eingabe bereit ist.

Die linke Seite der Oberfläche ist horizontal geteilt und enthält die Fenster *Current Directory* (oben) und *Command History* (unten). Durch Anklicken kann man diese Fenster aktivieren. Das obere Fenster ist doppelt belegt. Am unteren Rand des Current Directory-Fensters kann man „Workspace“ anklicken und das Current Directory-Fenster durch das Workspace-Fenster ersetzen. Analog kann man wieder zum Current Directory-Fenster zurückkehren.

Das Fenster *Current Directory* zeigt nicht nur das aktuelle Verzeichnis an (dieses ist ja auch in der zweiten Menüzeile über den Teilfenstern zu sehen), sondern auch die darin enthaltenen Unterverzeichnisse und Dateien.

Im Fenster *Workspace* stehen alle gerade aktiven Variablen, ihr Wert (bei Zahlen und kurzen Vektoren der numerische Wert, bei Character-Strings der Wert in Hochkommas, sonst die Größe, z.B. $< 3 \times 4$ double$>$ für eine 3×4-Matrix) und ihr Typ (z.B. *double* oder *char*). Nach dem Start ist dieses Fenster leer, aber es füllt sich, sobald man im Command Window

Operationen ausführt.

Das Fenster *Command History* zeigt die letzten in MATLAB eingegebenen Befehle an, und zwar nicht nur in der laufenden Sitzung, sondern nach dem Start werden die letzten Befehle der vorigen MATLAB-Sitzung angezeigt. Beim Start einer neuen Sitzung erscheint im Command History-Fenster eine mit Kommentarzeichen `%` versehene Zeile mit Datum und Zeit.

Geht man mit der Maus auf Trennlinien zwischen den Teilfenstern, so kann man die Breite und Höhe der Fenster verstellen. Durch Klicken auf das geschwungene Pfeilsymbol in der Kopfzeile eines Fensters (*Undock from Desktop*) kann man dieses Fenster aus dem Rahmen aller Fenster herausnehmen und in die gewünschte Form bringen. Durch das Anklicken des Kreuzes schließt man ein Teilfenster. (*Achtung*: Das Kreuz rechts oben in der Kopfzeile des MATLAB-Fensters sollte man nicht anklicken, weil sonst MATLAB beendet wird!) Die voreingestellte MATLAB-Oberfläche erhält man mit *Desktop* → *Desktop Layout* → *Default* zurück. Der Menüpunkt *Desktop* dient der Einstellung der MATLAB-Oberfläche. Es ist Geschmackssache, was man in der MATLAB-Oberfläche haben will, aber das Command Window ist auf jeden Fall nötig und sollte lang und breit genug sein, damit man viel ohne Scrollen sehen kann. Das Current Directory-Fenster ist hingegen weniger wichtig. MATLAB speichert Änderungen der Oberfläche, und beim nächsten Aufruf von MATLAB erhält man die zuletzt eingestellte Oberfläche. Auch eine Veränderung der Größe des ganzen Fensters wird gespeichert.

Die gebräuchlichsten Methoden, um aus MATLAB auszusteigen, sind das Anwählen von *File* → *Exit MATLAB*, dessen Shortcut `Strg-Q` (Strg = Ctrl) oder die Eingabe von `quit` im Command Window.

MATLAB enthält eine ausführliche Online-Dokumentation. Wenn man im Help nur schmökern will, ist es am besten, in der obersten Menüleiste *Help* und eines seiner Unterthemen anzuwählen, und man erhält ein Help-Fenster mit Hyperlinks zum Navigieren. Gezielte Information sucht man jedoch schneller durch Eingabe in das Command Window. Wenn man den Namen `functionname` einer Funktion kennt, über die man etwas wissen will, kann man im Workspace

```
>> help functionname
```

eingeben. Der Befehl

```
>> help
```

produziert eine Liste aller `help`-Themen.

MATLAB unterhält einen Befehlszeichenspeicher, der es ermöglicht, mit Hilfe der Pfeiltasten ↑ und ↓ im Laufe der Sitzung eingegebene Zeilen zu suchen. Gibt man eine charakteristische Zeichenkette des Zeilenbeginns an, werden mit den Pfeiltasten nur jene Befehlszeilen angezeigt, welche mit exakt dieser Zeichenkette beginnen. Ist der gesuchte Befehl gefunden, kann er mit den Pfeiltasten ← und → modifiziert werden, wobei zu beachten ist, dass der MATLAB-Editor normalerweise im Einfügemodus arbeitet.

Das aktuelle Verzeichnis ändert man am oben in der Menüleiste (2. Zeile) oder durch Eingabe von `cd` im Command Window, z.B. wechselt man durch den Befehl

```
>> cd matlab
```

ins Unterverzeichnis `matlab` des aktuellen Verzeichnisses.

Ein paar andere Menüpunkte in der Menüleiste werden noch im folgenden Text beschrieben.

1.2 Eingabe einfacher arithmetischer Ausdrücke

Die MATLAB-Befehle für die vier elementaren Operationen sind `+` für Addition, `-` für Subtraktion, `*` für Multiplikation und `/` für Division, z.B.:

```
>> 2+3
ans =
     5
```

Die Bedeutung von `ans` werden wir in Abschnitt 1.4 näher kennenlernen.

Der Potenzoperator wird mit `^` aktiviert und kann mit jedem Exponenten benutzt werden, z.B. `5^2.5` oder `5^(-3)`. Der Wurzeloperator wird mit `sqrt` aufgerufen, z.B. `sqrt(2)`. Der Dezimalpunkt muss immer als Punkt (und nicht als Komma!) eingegeben werden. Es gelten die gewohnten Rechenregeln (z.B. „Punkt- vor Strichrechnung“), und um Prioritäten zu setzen, muss man Klammern `( )` setzen, z.B.:

```
>> 3*(23+14.7-2/3)/3.5
ans =
   31.7429
```

MATLAB kann so wie ein „Taschenrechner“ verwendet werden und kennt auch eine Vielzahl von Standardfunktionen, z.B. `sin`, `cos` und `exp` (siehe Abschnitt 1.7). Dabei ist zu beachten, dass die Argumente der Winkelfunktionen stets im Bogenmaß eingegeben werden müssen.

1.3 Zahlenformate

Per Default (`format short`) werden 4 Nachkommastellen angegeben. Z.B. erhalten wir bei Eingabe der MATLAB-Konstante `pi` (π):

```
>> pi
ans =
    3.1416
```

Der im Computer gespeicherte Wert ist jedoch viel exakter. Das Format, mit dem die Zahlen dargestellt werden, wird geändert mit

```
>> format long
>> pi
ans =
   3.14159265358979
```

MATLAB-Formate für exponentielle Notation sind `format short e` und `format long e`. Mit `format` oder `format short` kehrt man zum Standardformat zurück. Ein nützliches Kommando ist auch `format compact`, mit dem Leerzeilen in der Ausgabe unterdrückt werden. In diesem Skriptum sind die Ausgaben alle in `format compact` dargestellt. Der Default ist `format loose`.

Das Format kann auch (dauerhaft) mit der Menüleiste eingestellt werden, und zwar wird mit *File* → *Preferences* ein neues Fenster geöffnet, mit Hilfe dessen man diverse Einstellungen

vornehmen kann. Mit der Auswahl *Command Window* erhält man 8 Auswahlmöglichkeiten für das numerische Format (*Numerical format*) und 2 Möglichkeiten für die Darstellung (*Numeric display*), nämlich `format compact` und `format loose`. Es empfiehlt sich, `format compact` einzustellen, um Platz zu sparen. Der Befehl `help format` liefert eine Übersicht über die Formate, die jedoch nicht ganz richtig ist. `format short` liefert nicht 5 signifikante Stellen, sondern 4 Nachkommastellen, und `format long` liefert 14 Nachkommastellen und geht gern zur Gleitkommadarstellung über (z.B. noch nicht bei `100/3`, aber schon bei `1000/3`).

Eine weitere MATLAB-Konstante (außer `pi`), die nicht neu definiert werden soll, ist `eps`, das sogenannte Maschinenepsilon, die kleinste positive Zahl, welche auf dem Computer die Ungleichung `1 < 1 + eps` erfüllt. Im PC-Labor erhalten wir mit `format short`

```
>> eps
ans =
   2.2204e-16
```

Unbestimmte Ausdrücke (z.B. `0/0`) werden mit `NaN` („not a number“) bezeichnet, und `Inf` und `-Inf` sind ∞ und $-\infty$. Auch die Bezeichnungen `nan`, `inf` und `-inf` liefern dasselbe, obwohl MATLAB bei Befehlen und selbst definierten Variablen zwischen Groß- und Kleinschreibung unterscheidet (siehe nächster Abschnitt).

1.4 Matrizen

Seit MATLAB 5 (d.h. also auch in MATLAB 7.1) ist der grundlegende Datentyp ein allgemeines mehrdimensionales Feld (Englisch *array*) mit komplexwertigen `double precision`-Einträgen, d.h. ein Element von $\mathbb{C}^{n_1 \times \cdots \times n_k}$ für $k \in \mathbb{N}$ und $n_1, \ldots, n_k \in \mathbb{N}_0$. (In früheren MATLAB-Versionen war jede Variable eine Matrix, mit Vektoren und Skalaren als Spezialfälle.) Es gibt also keinen Unterschied zwischen `integer`, `single precision` und `double precision`, und daher muss man die Variablen nicht deklarieren. Der einzige andere Typ sind `character`-Variablen, die mit Hochkommas eingegeben werden, z.B.:

```
>> s='Ich liebe Matlab'
s =
Ich liebe Matlab
```

Variablennamen in MATLAB müssen mit einem Buchstaben anfangen und dürfen nur Buchstaben, Ziffern und _ enthalten, und sie sollten nicht mit dem Namen eines MATLAB-Befehls, einer MATLAB-Variable oder einer MATLAB-Funktion zusammenfallen. MATLAB unterscheidet zwischen Groß- und Kleinschreibung, also ist z.B. `data` und `Data` nicht dieselbe Variable. Die Indizierung der Feldelemente beginnt immer mit 1.

Im Gegensatz zur mathematischen Konvention bedeutet `=` in MATLAB (und in einigen anderen Programmiersprachen) nicht Gleichheit, sondern Wertzuweisung. Wird eine Anweisung `variable =` Ausdruck durch Drücken der `<Return>`-Taste an den MATLAB-Kern gesandt, wird zuerst der Ausdruck ausgewertet und dann sein numerischer Wert in den Speicherplatz mit dem Namen `variable` geschrieben. Ein etwaiger ursprünglicher Wert von `variable` geht dabei verloren. Jede rechts vom Gleichheitszeichen `=` stehende Größe muss zum Zeitpunkt ihrer Verwendung einen Wert besitzen, d.h., es dürfen im Ausdruck nur Variablen vorkommen, denen schon ein Wert zugewiesen wurde.

Wir betrachten als Beispiel die Eingabe einer 3×3-Matrix:

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

Die Matrix wird also durch eckige Klammern eingeschlossen und zeilenweise eingegeben, wobei die einzelnen Elemente durch Leerschritte getrennt sind und der Beginn einer neuen Zeile mit einem Strichpunkt gekennzeichnet wird. Man kann aber auch die Elemente einer Zeile durch Beistriche trennen und die Eingabe einer neuen Zeile mit `<Return>` beginnen (und Kombinationen dieser zwei Methoden zur Eingabe dieser Matrix); die Eingabe von

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

liefert also dasselbe Resultat. Das Element in der i-ten Zeile und j-ten Spalte erhält man auf nahe liegende Weise:

```
>> A(2,3)
ans =
     6
```

In diesem Beispiel wurde der Ausdruck nicht explizit einer Variable zugewiesen, und in diesem Fall weist MATLAB automatisch das Resultat der Variable `ans` zu (wie wir schon in den Abschnitten 1.2 und 1.3 gesehen haben). Der Inhalt der Variable `ans` ändert sich im Laufe der Berechnungen, aber man kann mit `ans` immer auf den letzten in `ans` gespeicherten Wert zugreifen. Versuchen Sie z.B. `2*9`, gefolgt von `ans^2` und abgeschlossen mit `sqrt(ans)/9`. Was ist das Endergebnis?

Man kann Matrizen leicht modifizieren:

```
>> A(3,2) = 10
A =
       1      2      3
       4      5      6
       7     10      9
```

Wenn man den Strichpunkt `;` am Ende eines Ausdrucks schreibt, gibt der Computer das Ergebnis des Kommandos nicht aus, d.h., er wiederholt in diesem Fall nicht die Eingabe. Strichpunkte und Beistriche können auch verwendet werden, um mehrere in derselben Zeile eingegebene Befehle zu trennen, je nachdem, ob man eine Ausgabe wünscht oder nicht. Passt eine Anweisung nicht in eine Bildschirmzeile oder will man die Eingabe besser strukturieren, können mit drei Punkten `...` Fortsetzungszeilen geöffnet werden, z.B.

```
>> x = ...
3*(23 + 24.7 - 2.3)/3.5
```

Die drei Punkte müssen dabei am Ende der abgebrochenen Zeile stehen, nicht am Beginn der Fortsetzungszeile!

Wichtig ist auch das leere Feld `[]`, z.B.:

```
>> x = [7 4 6 2]; x(2) = []
x =
     7     6     2
```

Die zweite Komponente wird zu „leer“ gesetzt, also wird diese Komponente aus `x` herausgenommen, und die anderen Komponenten rücken nach.

1.5 Matrix- und Feldoperationen

MATLAB unterstützt die folgenden arithmetischen Operationen:

`+`	Addition	`-`	Subtraktion
`*`	Multiplikation	`^`	Potenz
`/`	Rechtsdivision	`\`	Linksdivision
`'`	konjugiert Transponierte	`.'`	Transponierte

Um die Wirkung dieser Operationen zu zeigen, nehmen wir an, dass die folgenden Matrizen eingegeben werden:

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

Die (konjugiert) Transponierte wird in MATLAB mit dem Apostroph bezeichnet:

```
>> B = A'
B =
     1     4     7
     2     5     8
     3     6     9
```

Multiplikation einer Matrix mit einem Skalar liefert das erwartete Resultat, und zwei Matrizen gleicher Größe werden elementweise addiert (oder subtrahiert); probieren Sie z.B. die folgenden drei Befehle aus:

```
>> 2*A
>> A/3
>> A + [b,b,b]
```

Im dritten Beispiel wird der Spaltenvektor `b`, dreimal nebeneinander geschrieben, zu einer 3×3-Matrix zusammengesetzt. Wenn man versucht, Matrizen verschiedener Dimension zu addieren (subtrahieren), erhält man eine Fehlermeldung. Eine Ausnahme bildet die Addition einer Matrix und eines Skalars. In diesem Fall wird der Skalar zu jedem Matrixelement addiert:

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

Matrixmultiplikation ist nur möglich, wenn die Größen zusammenpassen; andernfalls wird eine Fehlermeldung erzeugt:

```
>> A*b
>> b'*A
>> A*A', A'*A
>> b'*b, b*b'
```

MATLAB hat zwei Operatoren für die Division: einen für die Division von rechts und einen für die Division von links. Sind die Operanden Zahlen, führen beide Divisionsarten `6/3` und `3\6` zum gleichen Ergebnis. Links- und Rechtsdivision können aber auch auf Vektoren und Matrizen angewendet werden und liefern dann nicht das gleiche Ergebnis (siehe Abschnitt 1.9).

Elementweise (punktweise) Feldoperationen werden mit einem Punkt angedeutet. Z.B., wenn `A` und `B` Matrizen gleicher Größe sind, erhält man mit `A.*B` die Matrix, die durch elementweise Multiplikation der Matrizen `A` und `B` entsteht. Ebenso bezeichnen `./` und `.^` die elementweise Division bzw. Exponentiation, z.B.:

```
>> A^2, A.^2
>> A.*A, b.*b
>> 1./A
>> 1./A.^2
```

1.6 Funktionen zum Erzeugen von Matrizen

MATLAB enthält eine Anzahl von Funktionen zur Erzeugung von Matrizen (bzw. allgemein Feldern):

`eye`	Einheitsmatrix
`zeros`	Nullmatrix
`ones`	Matrix mit Einsern
`diag`	Diagonalmatrix
`triu`	oberer Dreieckteil einer Matrix
`tril`	unterer Dreieckteil einer Matrix
`rand`	Matrix aus Zufallszahlen (gleichverteilt in $[0, 1]$)
`randn`	Matrix aus Zufallszahlen (normalverteilt mit Mittel 0 und Varianz 1)

Z.B. produziert `zeros(2,3)` eine 2×3-Nullmatrix, `zeros(3)` eine 3×3-Nullmatrix und `zeros(2,3,4)` ein $2 \times 3 \times 4$-Feld mit Nullen. Dasselbe gilt für `ones`. `eye(n)` ist die `n`×`n`-Einheitsmatrix.

Wenn `b` ein (Zeilen- oder Spalten-)Vektor ist, ist `diag(b)` die Matrix mit den Elementen von `b` in der Diagonale; wenn `A` eine quadratische Matrix ist, ist `diag(A)` ein Vektor, der aus der Diagonale von `A` besteht. Was ist `diag(diag(A))`?

Für eine Matrix `A` ist `triu(A)` der obere Dreieckteil von `A` (d.h. die Matrix, die man aus `A` erhält, indem man die Elemente unter der Diagonale zu 0 setzt) und `tril(A)` der untere Dreieckteil von `A`. `rand` und `randn` funktionieren ähnlich wie `zeros` und `ones`, nur werden die Elemente mit Zufallszahlen und nicht mit Nullen bzw. Einsern belegt. `rand` und `randn` ohne Argument liefern eine gleichverteilte bzw. normalverteilte Zufallszahl.

Matrizen können aus Blöcken zusammengesetzt werden. Z.B., wenn `A` eine 3×3-Matrix ist, dann produziert

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

eine gewisse 5×5-Matrix.

1.7 Skalar-, Vektor- und Matrixfunktionen

In MATLAB gibt es einige Funktionen, die im Wesentlichen auf Skalare wirken und auf Felder elementweise angewendet werden. Die wichtigsten skalaren Funktionen sind:

`sin`	`cos`	`tan`	`cot`	Winkelfunktionen
`asin`	`acos`	`atan`	`acot`	inverse Winkelfunktionen
`sinh`	`cosh`	`tanh`	`coth`	Hyperbelfunktionen
`asinh`	`acosh`	`atanh`	`acoth`	inverse Hyperbelfunktionen
`exp`	`log`	`log10`	`log2`	Exponentialfunktion, Logarithmus zur Basis e, 10 und 2
`abs`	`sign`	`mod`	`rem`	$\|\ \|$, sign, vorzeichenbehafteter Divisionsrest, Divisionsrest
`fix`	`floor`	`ceil`	`round`	Runden nach 0, unten, oben, zur nächsten ganzen Zahl
`sqrt`				Quadratwurzel

Andere nützliche MATLAB-Funktionen operieren auf (Zeilen- und Spalten-)Vektoren. Wenn sie auf Matrizen angewendet werden, werden sie spaltenweise berechnet. Zu diesen Funktionen gehören die folgenden:

`max`	`min`	`sum`	`prod`
`mean`	`median`	`std`	`var`
`cumsum`	`cumprod`	`sort`	

Wenn die Matrix `A` wie in Abschnitt 1.5 definiert ist, erhalten wir wegen der spaltenweisen Anwendung von den obigen Operationen z.B. für `sum`

```
>> sum(A)
ans =
    12     15     18
```

Man kann die Operation aber mit `sum(A,dim)` auf jede beliebige Dimension anwenden, und Analoges gibt es auch (mit eventuell etwas anderer Syntax; siehe `help`) für die anderen Funktionen. Der obige Output entspricht `sum(A,1)`, und für `sum(A,2)` erhalten wir:

```
>> sum(A,2)
ans =
     5
    15
    24
```

Das größte Element der Matrix `A` findet man mit `max(max(A))`. Wenn man zwei Ausgabeparameter verwendet, erhält man mit `max` und `min` auch den Index des minimalen bzw. maximalen Wertes. Z.B. erhalten wir für den Vektor `b` aus Abschnitt 1.5:

```
>> [bmax,index] = max(b)
bmax =
     6
index =
     3
```

Man kann `max` und `min` aber auch auf zwei Argumente anwenden, z.B.:

```
>> max(2,3)
ans =
     3
```

Die nützlichsten Matrixfunktionen sind die folgenden:

`inv`	Matrixinverse
`det`	Determinante
`trace`	Spur
`norm`	Matrix- oder Vektornorm
`size`	Größe einer Matrix
`rank`	Rang
`eig`	Eigenwerte und Eigenvektoren
`null`	Nullraum
`svd`	Singulärwertezerlegung
`poly`	charakteristisches Polynom
`cond`	Matrixkonditionszahl
`expm`	Matrixexponentialfunktion
`logm`	Matrixlogarithmus
`sqrtm`	Matrixquadratwurzel
`lu`	LR-Zerlegung (Faktoren der Gauß-Elimination)
`qr`	QR-Zerlegung
`chol`	Cholesky-Zerlegung

Der Befehl `size(x)` gibt für ein Feld `x` beliebiger Dimension den Vektor der Dimensionslängen an, z.B.:

```
>> x = ones(2,3,4);
>> size(x)
ans =
     2     3     4
```

`size(x,dim)` gibt die Länge der durch den Skalar `dim` spezifizierten Dimension `dim` aus; z.B. erhält man mit `size(A,1)` die Anzahl der Zeilen der Matrix `A`.

Mit `length(x)` erhält man die größte Dimension des Feldes `x`. Im Fall des obigen Feldes `x` würde man 4 erhalten, im Fall eines Vektors gerade die Länge des Vektors.

1.8 Der Doppelpunktoperator und Untermatrizen

Der Doppelpunktoperator ist sehr nützlich, um Indexfelder und Vektoren mit gleichen Differenzen zwischen aufeinanderfolgenden Elementen zu erhalten. Das `help` zum Doppelpunktoperator erhält man mit Hilfe von `help colon`.

Die Doppelpunktnotation funktioniert nach der Idee, dass ein Vektor erzeugt werden kann, wenn man die Werte `start`, `step` (Schrittweite) und `end` eingibt. Wenn `start`, `step` und `end` ganze Zahlen sind, erhält man mit

```
>> iii = start:step:end
```

einen Vektor von ganzen Zahlen mit gleichen Abständen. Ohne den Parameter `step` hat das Inkrement den Defaultwert 1. Z.B. ist `1:6` der Zeilenvektor `[1 2 3 4 5 6]`. `start`, `step` und `end` müssen keine ganzen Zahlen sein, und `step` muss nicht positiv sein.

Der Doppelpunkt wird auch dafür verwendet, Untermatrizen einer Matrix zu bilden. Wenn `A` eine (mindestens) 5×3-Matrix ist, kann man eine 4×3-Teilmatrix herausnehmen mit `A(2:5,1:3)`. Wenn man eine ganze Zeile haben will, dient der Doppelpunkt als Wildcard; z.B. ist `A(2,:)` die zweite Zeile (ebenso für Spalten). Man kann einen Vektor umdrehen, indem man ihn rückwärts indiziert, z.B. `x(end:-1:1)`, wobei MATLAB die Variable `end` automatisch zur Vektorlänge setzt. `A(:)` produziert einen Spaltenvektor, der durch Aneinanderhängen der Spaltenvektoren der Matrix `A` entsteht. Allgemeineres „Umordnen“ von Matrizen erzielt man mit `reshape(A,m,n)`. Z.B. kann eine 5×4-Matrix `B` umgeordnet werden in eine 2×10-Matrix durch

```
>> Bnew = reshape(B,2,10)
```

Ferner kann der Doppelpunktoperator verwendet werden, um Zeilen oder Spalten aus einer Matrix herauszunehmen, indem man sie zu `[]` setzt.

1.9 Lösung von Matrixgleichungen mit Matrixdivision

Wenn A eine quadratische, reguläre Matrix ist, ist die Lösung der Gleichung $Ax = b$ durch $x = A^{-1}b$ gegeben. In MATLAB könnte man die Lösung mit `x = inv(A)*b` berechnen, aber es ist effizienter, den Backslash-Operator `\` zu verwenden:

```
>> A = [1 2 3; 2 3 4; 4 2 5]; b = [4; 5; 1];
>> x = A\b
x =
   -1.4000
    1.8000
    0.6000
```

`A\b` ist also mathematisch äquivalent zu `inv(A)*b`. MATLAB berechnet jedoch in diesem Fall die Inverse nicht, sondern löst das Gleichungssystem direkt mit Gauß-Elimination. MATLAB berechnet immer ein Ergebnis, auch wenn die Matrix `A` nicht invertierbar ist. In solchen Fällen ist Vorsicht geboten, da die Lösung partikulär, d.h. speziell ist, und damit falsch sein kann. Wenn die Matrix `A` singulär oder schlecht konditioniert ist, wird eine Warnmeldung ausgegeben. Falls die Matrix nicht quadratisch ist, erzeugt MATLAB eine Lösung, die die Summe der Abweichungsquadrate minimiert; siehe `help slash` für Details. Die Größen der Matrizen auf beiden Seiten des Backslash müssen kompatibel sein, und der Divisionsoperator ist für n-dimensionale Felder mit $n > 2$ nicht definiert. Für zwei quadratische reguläre Matrizen `A` und `B` gleicher Größe liefert `A\B` dasselbe Resultat wie `inv(A)*B` und `A/B` dasselbe Resultat wie `A*inv(B)`.

1.10 Komplexe Zahlen

Die imaginäre Einheit i ist in MATLAB in den Konstanten `i` und `j` gespeichert. (Die zweite Bezeichnungsweise wird in der Elektrotechnik verwendet.) Mit Hilfe der imaginären Einheit kann man komplexe Zahlen in der üblichen Art und Weise eingeben, z.B.:

```
>> z = 3 + 4*i
z =
   3.0000 + 4.0000i
```

Mit `j` erhält man das gleiche Resultat, allerdings gibt MATLAB immer `i` zurück. Es ist nicht notwendig, den Multiplikationsoperator `*` vor `i` oder `j` zu schreiben:

```
>> 3 + 4i
ans =
   3.0000 + 4.0000i
```

Beachten Sie, dass `i` und `j` nicht mehr die Bedeutung der imaginären Einheit haben, wenn sie anderweitig als Konstante oder Variable definiert worden sind, z.B. als Laufparameter bei der Programmierung. Das nachfolgende Beispiel zeigt, wie MATLAB diese Schwierigkeit umgeht. Beachten Sie dabei den Unterschied in der Syntax zur Definition der Zahlen `a` und `b`:

```
>> i = 2; a = 3 + 4*i
a =
   11
```

```
>> b = 3 + 4i
b =
   3.0000 + 4.0000i
```

Realteil, Imaginärteil und konjugiert Komplexes einer komplexen Zahl werden mit `real`, `imag` bzw. `conj` berechnet.

1.11 Eingaben aufzeichnen – das „Tagebuch“

Eingaben und MATLAB-Berechnungen können auf einfache Weise mit der Funktion `diary` aufgezeichnet werden. Die Eingaben zwischen den Statements `diary` und `diary off` werden auf einer Datei mit dem Namen `diary` gespeichert und im aktuellen Verzeichnis abgespeichert. Wenn zusammen mit `diary` ein Name angegeben wird, kann man verschiedene Tagebücher anlegen, z.B. erzeugt `diary report.dia`, das ebenfalls mit `diary off` beendet wird, ein „Tagebuch“ mit dem Namen `report.dia`.

1.12 Speichern und Laden von Daten

In MATLAB gibt es mehrere Möglichkeiten, Daten zu speichern und zu laden. Eine davon ist mittels der Befehle `save` und `load`, die Daten im binären Format speichert. Als Beispiel erzeugen wir eine Tabelle mit Sinuswerten für die Winkel zwischen 0 und 2π mit der Schrittweite $\pi/60$:

```
>> x = 0:pi/60:2*pi;
>> y = sin(x);
>> t = [x' y'];
```

Wir speichern nun das Feld `t` auf einer Datei mit dem Namen `io.mat` (die Endung `.mat` wird automatisch hinzugefügt):

```
>> save io t
```

Wenn Sie die Datei in einer anderen MATLAB-Sitzung laden wollen, dann geben Sie einfach ein:

```
>> load io
```

D.h., diese Daten bleiben auch nach Aussteigen aus einer MATLAB-Sitzung erhalten (und haben dann wieder den gleichen Namen), während alle anderen Daten verloren gehen. Mit dem geladenen Feld `t` kann man dann weiterarbeiten. Man kann auf einem `mat`-File auch mehrere Werte speichern, z.B.:

```
>> save test x y
```

Mit `save test` werden alle im Workspace befindlichen Variablen auf `test.mat` gespeichert, mit `save` auf der Default-Datei `matlab.mat`. Dasselbe kann man auch mit *File* → *Save Workspace As* erreichen, wobei man nach dem Namen des `mat`-Files gefragt wird, auf dem man die Daten speichern soll.

Mit dem Befehl `who` erhält man eine Auflistung aller im Workspace befindlichen Variablen, mit `whos` werden noch zusätzlich Größe, Bytezahl und Typ aufgelistet. Wenn man ein offenes Workspace-Window hat, sind diese Befehle aber nicht mehr nötig.

1.13 Betriebssystemkommandos

In MATLAB kann man Betriebssystembefehle eingeben, ohne MATLAB zu verlassen. Diesen Befehlen muss ein Rufzeichen `!` vorangesetzt werden, z.B.:

```
>> !rm test.m
```

2 2D-Graphik

Der Befehl `plot(x,y)` plottet die Werte zweier gleich langer Vektoren `x` und `y` gegeneinander, wobei die Werte des ersten als Abszissenwerte und die Werte des zweiten als Ordinatenwerte genommen werden. Z.B., wenn `x` und `y` Vektoren der Länge `n` sind, so produziert `plot(x,y)` eine durchgezogene Kurve mit den Datenpunkten `(x(1),y(1))`, `(x(2),y(2))`,..., `(x(n), y(n))`. Wenn `x` und `y` Matrizen sind, wird die erste Spalte von `x` gegen die erste Spalte von `y` aufgetragen, die zweite Spalte von `x` gegen die zweite Spalte von `y` usw. Die Matrizen `x` und `y` müssen also entweder die gleiche Dimension haben, oder eines dieser beiden Felder ist ein Vektor der Länge `n` und das zweite eine Matrix mit `n` Zeilen oder Spalten. Dann wird der Vektor gegen alle Spalten bzw. Zeilen der Matrix aufgetragen.

Die folgende Befehlssequenz zeichnet den Graphen einer Sinusfunktion zwischen 0 und 2π:

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

Man kann auch strichlierte, punktierte oder strichpunktierte Kurven zeichnen, die einzelnen Datenpunkte mit verschiedenen Symbolen zeichnen oder die Farbe der Kurven/Symbole auf dem Bildschirm verändern; siehe `help plot`. Z.B. produziert `plot(x,y,'b:')` eine blaue

punktierte Linie, und `plot(x,y,'r*')` macht einen roten Stern an jedem Datenpunkt, ohne eine durchgezogene Linie zu zeichnen. Man kann in einem `plot`-Befehl mehrere Daten angeben, die in denselben Plot gezeichnet werden:

```
>> plot(x1,y1,'Format1',x2,y2,'Format2',...)
```

Wenn die Formate weggelassen werden, wird per Default eine durchgezogene Linie gezeichnet.

Die häufigste Variante, um eine Funktion zu zeichnen, ist, dass man zuerst einen Vektor `x` von Stützstellen erzeugt, an denen die Funktion ausgewertet werden soll. Im einfachsten Fall erzeugt man mit dem Doppelpunkt-Operator äquidistante Stützstellen. Dann wird die Funktion an `x` ausgewertet, und die Funktionswerte werden in einem Vektor `y` abgespeichert. Schließlich erzeugt man den Funktionsgraphen durch `plot(x,y)`.

Den Graphen kann man einen Titel, Achsenbeschriftungen usw. geben, wobei die Beschriftungen in Hochkommas eingegeben werden.

`title`	Titel des Graphen (darüber)
`xlabel`	Beschriftung der x-Achse
`ylabel`	Beschriftung der y-Achse
`text`	Positionierung eines Textes bei gewissen Koordinaten

In unserem Beispiel könnten wir z.B. eingeben:

```
>> xlabel('x')
>> ylabel('sin(x)')
```

Weitere nützliche Befehle sind:

`axis([a b c d])`	verändert das Fenster des aktuellen Graphen zu $a \leq x \leq b$, $c \leq y \leq d$
`hold on`	Einfrieren des aktuellen Plots, um weitere Dinge hineinzuzeichnen
`hold off`	Aufheben der Einfrierung
`subplot`	mehrere Plots in einem Fenster

Die Befehlen `set(gca,'xlim',[xmin xmax])` und `set(gca,'ylim',[ymin ymax])` stellen eine Alternative zu `axis` dar, die x- und y-Grenzen zu setzen. Mit `figure(2)`, `figure(3)`,... kann man ein zweites, drittes ... Plotfenster aufmachen, und mit `figure(1)` kann man wieder auf das erste Fenster zurückgehen, um dort etwas dazuzuzeichnen oder einen neuen Plot hineinzuzeichnen. Mit `clf` („clear figure“) löscht man den Inhalt des aktuellen Plotfensters. Weil MATLAB standardmäßig die Achsen so skaliert, dass die Graphik das Graphikfenster möglichst gut ausfüllt, kommt es oft zu einer beträchtlichen Verzerrung der Proportionen. Diese kann durch den Befehl `axis equal` korrigiert werden, und `axis normal` schaltet wieder in den Standardmodus zurück. Mit `axis square` erhält man einen quadratischen Plot. Der Befehl `subplot(m,n,p)` teilt das Graphikfenster in `m`×`n` Teilplots ein und macht den `p`-ten Teilplot (zeilenweise gezählt) zum aktuellen Plot, in den anschließend etwas hineingezeichnet werden kann.

Mit Hilfe des Befehls `print` können Plotfiles erzeugt werden, die zum Drucker geschickt werden können, z.B. erzeugt

```
>> print -deps plot.ps
```

ein „encapsulated“ Postscript-File und

```
>> print -dps plot.ps
```

ein „normales“ Postscript-File. (Beide sind zum Ausdrucken im PC-Labor mittels `lpr plot.ps` geeignet, aber nur „encapsulated“ Postscript-Files lassen sich in LaTeX einbinden.)

3 Programmieren in MATLAB

3.1 Vergleichende und logische Operatoren

Vergleichende Operatoren ermöglichen den Vergleich von Skalaren (oder Feldern, elementweise). Das Resultat ist ein Skalar (oder ein Feld derselben Größe wie die Argumente), das aus Nullen und Einsern besteht. Wenn das Ergebnis des Vergleiches wahr ist, ist das Resultat (der Eintrag) 1, sonst 0. In MATLAB sind die folgenden Operatoren implementiert:

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

Die Relationen können durch die folgenden logischen Operatoren verbunden oder quantifiziert werden:

`&`	und
`|`	oder
`~`	nicht

3.2 Schleifen und konditionale Verzweigungen

Es gibt vier MATLAB-Befehle für Schleifen und konditionale Verzweigungen: `for`, `while`, `if` – `elseif` – `else` und `switch`. Alle vier Befehle werden mit `end` abgeschlossen.

For. Die mehrfache Ausführung einer Anweisung oder eines Blocks von Anweisungen wird in MATLAB mit einer `for`-Schleife durchgeführt. Z.B. erzeugen für gegebenes `n` die Befehle

```
x = []; for i = 1:n, x = [x,i^2], end
```

oder

```
x = [];
for i = 1:n
  x = [x,i^2]
end
```

einen bestimmten `n`-Vektor, und die Befehle

```
x = []; for i = n:-1:1, x = [x,i^2], end
```

produzieren denselben Vektor in umgekehrter Reihenfolge. Die `for`-Schleife wird vor allem dann verwendet, wenn man im Vorhinein weiß, wie oft der Befehl (höchstens) ausgeführt werden wird.

While. Die allgemeine Form einer `while`-Schleife ist

```
while Bedingung
  Anweisungen
end
```

Die Anweisungen werden ausgeführt, solange die Bedingung wahr ist. Nach jedem Durchlauf der Anweisungen werden dabei die Bedingungen wieder überprüft. Die `while`-Schleife ist vor allem dann sinnvoll, wenn man im Vorhinein nicht weiß, wie oft die Schleife durchgeführt werden muss, aber ein Abbruchkriterium für die Schleife hat.

If. Die allgemeine Form einer einfachen `if`-Anweisung ist

```
if Bedingung
  Anweisungen
end
```

Die Anweisungen werden nur ausgeführt, wenn die Bedingung wahr ist. Es gibt auch mehrfache Verzweigungen, die für Fallunterscheidungen nützlich sind:

```
if Bedingung 1
  Anweisung 1
elseif Bedingung 2
  Anweisung 2
     ⋮
else
  Anweisung n + 1
end
```

In einer zweifachen Verzweigung fehlen die `elseif`-Teile. In verschachtelten Bedingungen werden die Prioritäten durch runde Klammern gekennzeichnet, z.B.

```
if (Bedingung 1 & Bedingung 2) | Bedingung 3
```

Mit Hilfe des Befehls `break` kann man eine `for`- oder `while`-Schleife frühzeitig beenden, z.B., wenn eine gewisse Bedingung erfüllt ist:

```
if Bedingung, break, end
```

Eine Schleife, die abbricht, wenn eine Bedingung erfüllt ist, kann man wie folgt programmieren:

```
while 1
   ...
   if Bedingung, break, end
end
```

Die Bedingung für das Fortsetzen der `while`-Schleife ist immer wahr, weil `1` einer wahren Aussage entspricht, also wird die Schleife erst abgebrochen, wenn die Bedingung in der `if`-Anweisung erfüllt ist.

Switch. Die allgemeine Form einer `switch`-Anweisung lautet

```
switch Switch-Ausdruck
  case Fall 1
    Anweisung 1
  case {Fall 2, Fall 3, ...}
```

```
    Anweisung 2
    ⋮
  otherwise
    Anweisung n
end
```

Die `switch`-Anweisung dient dazu, Anweisungen unter gewissen Bedingungen zu exekutieren. Es wird eine Menge von Anweisungen aus einer beliebigen Anzahl von Alternativen ausgeführt. Jede Alternative wird ein „Fall“ (`case`) genannt und besteht aus der `case`-Anweisung, einem oder mehreren `case`-Ausdrücken (oben: Fall 1, Fall 2, ...) und einer oder mehreren Anweisungen. Es werden die Anweisungen ausgeführt, die dem ersten Fall entsprechen, wo der Switch-Ausdruck den Wert eines `case`-Ausdrucks annimmt. Wenn kein `case`-Ausdruck dem Wert des Switch-Ausdrucks entspricht, dann werden die unter `otherwise` stehenden Anweisungen ausgeführt (wenn ein `otherwise`-Block existiert). Nach der Ausführung der zum entsprechenden `case` oder `otherwise` gehörenden Anweisungen geht die Programmexekution zur Zeile nach dem `end`.

Der Switch-Ausdruck kann ein Skalar oder ein String sein. Anders als das `switch` in C, fällt das MATLAB-`switch` nicht durch. D.h., es wird nur der erste `case` mit Übereinstimmung exekutiert, und darauf folgende Übereinstimmungen werden ignoriert. Daher sind keine `break`-Anweisungen nötig.

3.3 MATLAB-Skripts (m-Files)

Alle Ausdrücke, die nach dem MATLAB-Prompt eingegeben werden können, können auch auf einem Textfile gespeichert und als Skript ausgeführt werden. Dieses Textfile kann mit dem in MATLAB eingebauten Editor oder einem anderen Linux-Editor (z.B. `vim`, `xemacs`) generiert werden. Der Filename muss mit `.m` enden, und das Skript wird in MATLAB ausgeführt, indem man den Filenamen (mit oder ohne `.m`) eingibt.

Mit Hilfe von *File* → *New* → *M-file* öffnet der MATLAB-Editor eine leere Datei mit der Überschrift „untitled“. Um sie nach dem Beschreiben abzuspeichern, muss man in der Menüleiste des Editor-Fensters *File* → *Save As* wählen. MATLAB fragt dann nach dem gewünschten Dateinamen und gibt als Default `untitled.m` an. Die Menüzeile des Editors enthält selbsterklärende Buttons für Abspeichern, Cut, Paste, Undo usw. *Achtung*: Wenn man nur den Editor verlassen will, soll man auf *File* → *Close Editor* oder *File* → *Close filename* gehen (dabei wird auch noch einmal nachgefragt, ob noch nicht gespeicherte Änderungen gespeichert werden sollen), nicht auf *File* → *Exit MATLAB*, denn Letzteres beendet nicht nur den Editor, sondern die ganze MATLAB-Sitzung!

Mit Hilfe von *File* → *Open* kann man eine bereits existierende Datei im aktuellen Verzeichnis öffnen.

3.4 MATLAB-Funktionen

Man kann eigene Funktionen schreiben und zur MATLAB-Umgebung hinzufügen. Diese Funktionen sind eine spezielle Art von m-Files und werden auch – wie oben beschrieben – mit dem eingebauten MATLAB-Editor oder einem Linux-Editor erzeugt. Das erste Wort in dieser Datei muss `function` sein, um der MATLAB-Umgebung mitzuteilen, dass es sich um eine Funktion handelt. Der Filename muss mit `.m` enden und wird der Name der neuen Funktion

für MATLAB. Wir betrachten das Beispiel einer Funktion, die die Hypotenuse eines rechtwinkeligen Dreiecks berechnet, und schreiben den nachfolgenden Programmcode in die Datei `pyt.m`.

```
function h = pyt(a,b)
% pyt berechnet die Hypotenuse eines rechtwinkligen Dreiecks
% nach dem Satz von Pythagoras. Eingaben sind die Seiten des
% Dreiecks (Schenkel).
h = sqrt(a.^2 + b.^2);
```

Die Feldoperationen `a.^2` und `b.^2` ermöglichen es dem Anwender, Felder für die Seiten zu übergeben. Vor allem fürs Plotten (siehe Abschnitt 2) ist es wichtig, die Operationen in Punkt-Notation anzugeben. Nachdem Sie die Datei abgespeichert haben, können Sie in MATLAB eingeben:

```
>> pyt(3,4)
ans =
     5
```

`a` und `b` sind also sogenannte „Dummy-Variablen“, d.h., wenn man die Funktion aufruft, müssen die Argumente nicht unbedingt `a` und `b` heißen (im obigen Beispiele sind es die numerischen Werte `3` und `4`). Man kann in einer `function` oder in einem m-Skript auch ein m-Skript mit dessen Namen (mit oder ohne `.m`) aufrufen. Dann werden alle Befehle aus dem m-Skript an dieser Stelle eingefügt, aber alle Variablen behalten ihren Namen, d.h., m-Skripts enthalten keine Dummy-Variablen.

MATLAB-Funktionen müssen mit `function` beginnen. Die Information, die nach `function` in derselben Zeile folgt, ist eine Deklaration, wie die Funktion heißt und welche Ein- und Ausgabeargumente sie hat. Der Name der Funktion sollte mit dem Namen des m-Files übereinstimmen. Eingabeargumente werden in runden Klammern nach dem Funktionsnamen aufgezählt. Wenn man mehrere Ausgabeargumente hat, umgibt man sie mit eckigen Klammern, z.B.

```
function [x,y] = test(a,b)
```

Im Programm müssen den Feldern `x` und `y` Werte zugewiesen werden.

Eine Zeile, die mit `%` beginnt, ist eine Kommentarzeile. Die erste Gruppe von Kommentaren in einer Funktion werden von der `help`-Utility von MATLAB benützt. D.h., wenn man `help pyt` eintippt, erhält man die obigen drei Kommentarzeilen von `pyt.m` auf dem Bildschirm.

4 Verschiedenes

4.1 Einlesen von Daten und Ausgabe auf eine externe Datei

Die Kommandos, die in Abschnitt 1.12 benutzt wurden, um das Feld `t` zu erzeugen, kann man auch auf eine m-Datei schreiben, z.B. `sinfile.m`. In einer anderen MATLAB-Sitzung kann man diese Datei mit `sinfile` laden und mit den enthaltenen Befehlen das Feld `t` erzeugen. Alternativ dazu können Sie das Feld mit einem Texteditor direkt in die Datei `sinfile.m`

schreiben. Bei dieser Vorgehensweise werden die Daten im ASCII-Format gespeichert, nicht im Binärformat wie in `mat`-Files.

Man kann auch Daten auf externe Dateien schreiben, und zwar mit dem Befehl `fprintf`, der ähnlich wie in Ansi C funktioniert. Z.B.:

```
x = 0:0.1:1; y = [x; exp(x)];
fid = fopen('exp.txt','w');
fprintf(fid,'%6.2f %12.8f\n',y);
fclose(fid);
```

In der zweiten Zeile wird ein File `exp.txt` mit Schreibberechtigung (`w` = write) geöffnet, und der Filename wird der Variable `fid` zugewiesen. In der dritten Zeile wird ausgedruckt, wobei zwischen Hochkommas das Format steht, und zwar die Formatspezifikation von 2 Zahlen, gefolgt von einem Zeilenumbruch `\n`. Das Format `%6.2f` bedeutet, dass die Zahl als Gleitkommazahl mit (höchstens) 6 signifikanten Stellen, davon 2 nach dem Komma, dargestellt wird. Das Format wird auf die 2×11-Matrix `x` spaltenweise angewendet, wobei immer wieder von vorn begonnen wird, wenn man am Ende des Formatbefehls anlangt, d.h., man erhält eine 11×2-Tabelle der Form

```
0.00     1.00000000
0.10     1.10517092
  ⋮
1.00     2.71828183
```

4.2 Der Befehl `input` – direkte Eingabe

Der Befehl `input` ermöglicht es dem Benutzer, während der Exekution eines Programms Werte einzugeben, z.B.:

```
x = input('Eingabe von x: x = ')
```

Der zwischen Hochkommas stehende Text wird am Bildschirm dargestellt, und das Programm wartet so lange, bis der Anwender eine Antwort eingibt, die der Variablen `x` zugewiesen wird. Der Befehl

```
name = input('Geben Sie Ihren Namen ein: ','s')
```

fragt nach einem `character`-String, der dann ohne Hochkommas eingegeben werden muss und der Variable `name` zugewiesen wird.

4.3 Formatierung der Ausgabe

Der Befehl `disp(x)` stellt das Feld `x` dar, ohne den Feldnamen auf dem Bildschirm auszugeben. Ansonsten liefert es das gleiche Resultat wie das Auslassen des Strichpunktes (d.h. es wird das gerade aktuelle Format genommen), außer dass leere Felder nicht dargestellt werden. Wenn `x` ein `character`-String ist, wird der Text dargestellt.

Mit `disp` und den Funktionen `int2str` und `num2str` kann man längere Ausgaben zusammensetzen. `int2str(x)` rundet die Einträge der Matrix `x` auf ganze Zahlen und konvertiert

das Ergebnis in eine Stringmatrix. `num2str(x)` konvertiert die Matrix `x` in einen String in `format short` und, wenn nötig, mit einem Exponenten. Der Befehl `num2str(x,n)` konvertiert die Matrix `x` in eine Stringdarstellung mit höchstens `n` signifikanten Stellen. Z.B. könnte in einem Programm, in dem der Variable `n` ein Wert zugewiesen wurde, der folgende Befehl vorkommen:

```
disp(['Sie haben noch ',int2str(n),' Versuche'])
```

Die Strings werden mit eckigen Klammern zu einem längeren String zusammengesetzt. Ein anderes Beispiel ist:

```
disp(['pi in format short: ',num2str(pi)])
disp(['pi auf 15 signifikante Stellen: ',num2str(pi,15)])
```

Mit Hilfe von `sprintf` hat man mehr Möglichkeiten, Daten als formatierten String zu schreiben. Der Befehl `s = sprintf('Format',A)` formatiert die Daten in der Matrix `A` gemäß dem String 'Format', der die gleiche Syntax wie das Format-Argument in `fprintf` in Abschnitt 4.1 hat, und weist sie der Character-Variable `s` zu, z.B.:

```
disp(['pi auf 6 Nachkommastellen: ',sprintf('%8.6f',pi)])
```

4.4 Der Befehl `error`

Der Befehl

```
error('Fehlermeldung')
```

schreibt die in Hochkommas eingegebene Fehlermeldung auf den Bildschirm und bricht das Programm ab; der Befehl `error` allein bewirkt nur einen Abbruch, ohne jedoch etwas auf den Bildschirm zu schreiben.

4.5 Der Befehl `feval`

Ein nützlicher Befehl ist `feval`, der eine Funktion, die durch einen String spezifiziert ist, auswertet; z.B. berechnet

```
fcn = 'sin'; feval(fcn,0)
```

den Sinus von 0. Das ist nützlich, wenn man eine `function` schreiben will, die eine Funktion als Eingabeparameter haben soll. Die Funktion `function int = simpson(fcn,a,b)` könnte z.B. eine Approximation des Integrals der durch den String `fcn` gegebenen Funktion von `a` bis `b` mit Hilfe der (einfachen) Simpson-Regel berechnen; wenn die Funktion an der Stelle `x` ausgewertet werden soll, schreibt man im Unterprogramm `feval(fcn,x)`.

4.6 Der Befehl `find`

Mit dem Befehl `find` findet man die Indizes eines Vektors, die eine gewisse Bedingung erfüllen, z.B.:

```
>> x = 2:10; find(x > 5)
ans =
     5     6     7     8     9
```

4.7 Der Pfad

Normalerweise befinden sich im MATLAB-Pfad nur die in der MATLAB-Distribution enthaltenen und die im aktuellen Verzeichnis befindlichen Programme sowie das Verzeichnis `~/matlab` (wenn vorhanden). Wenn man auch auf Programme aus anderen Verzeichnissen zugreifen will, kann man z.B.

```
>> path(path,'/eusers/huyer/matlab/integration')
```

das Verzeichnis `/eusers/huyer/matlab/integration` (Eingabe des ganzen Linux-Pfades!) zum Pfad hinzufügen. Wenn man einen Pfad dauerhaft hinzufügen will, muss man diesen Befehl (natürlich ohne das MATLAB-Prompt `>>`) in die Datei `startup.m` im Unterverzeichnis `matlab` (also z.B. in `/eusers/huyer/matlab`) hineinschreiben, die – so vorhanden – bei jedem Start von MATLAB ausgeführt wird.

4.8 Der Befehl `clear`

Mit dem Befehl `clear variable` kann man den Wert einer Variable löschen. Wenn man eine MATLAB-spezifische Variable oder eine MATLAB-Funktion mit einem Wert belegt hat, erhält die Variable durch `clear` wieder ihren vorigen Wert. Mit `clear` werden alle Variablen aus dem Workspace entfernt, was insbesondere sinnvoll ist, wenn man schon Probleme mit dem Speicherplatz hat.

4.9 Fehlermeldungen

MATLAB liefert Fehlermeldungen, wenn die Syntax eines Programms nicht korrekt ist, wenn die Feldgrenzen über- oder unterschritten werden, die Dimensionen bei einer Matrixoperation nicht zusammenpassen oder andere unerlaubte Operationen stattfinden, und bricht das Programm dann ab. Ein komplexes Programm besteht meist aus mehreren, teils verschachtelten Unterprogrammen. MATLAB zeigt nicht nur die Zeile des Fehlers an (im Hauptprogramm und im Unterprogramm bzw. den Unterprogrammen bei größerer Verschachtelungstiefe), sondern die Namen der in der Fehlermeldung genannten Programme sind anklickbar, und beim Anklicken wird der Editor geöffnet. Sei z.B. `runsimpson` ein m-Skript, in dessen 6. Zeile die Funktion `simpson` aufgerufen wird, in deren 11. Zeile sich ein Syntaxfehler befindet. Die Fehlermeldung zeigt die 6. Zeile von `runsimpson.m` und die 11. Zeile von `simpson.m` und die Art des Fehlers an, durch Anklicken von `/eusers/huyer/matlab/simpson.m` wird dieses Programm im Editor geöffnet, und der Fehler kann behoben werden.

4.10 3D-Graphik

Einen einfachen 3D-Plot kann man mit Hilfe von

```
>> mesh(x,y,Z)
```

erzeugen. Dabei ist `x` ein Vektor der Länge `n`, `y` ein Vektor der Länge `m` und `Z` eine `m`×`n`-Matrix. Die Gitterpunkte des gezeichneten Gitters sind (`x(j)`,`y(i)`,`Z(i,j)`), d.h. `x` und `y` entsprechen den Spalten bzw. Zeilen von `Z`.

MATLAB-Literatur

Ottmar Beucher, *MATLAB und Simulink*, Pearson Studium, München, 2006

Frieder und Florian Grupp, *MATLAB 7 für Ingenieure*, Oldenbourg Verlag, München Wien, 2004

Wolfgang Schweizer, *MATLAB kompakt*, Oldenbourg Verlag, München Wien, 2005

Christoph Überhuber, Stefan Katzenbeisser und Dirk Praetorius, *MATLAB 7 – eine Einführung*, Springer, Wien New York, 2004

Jörn Berens und Armin Iske, *MATLAB – Eine freundliche Einführung*, Version 1.1, 1999, `http://www-m3.ma.tum.de/twiki/put/Allgemeines/Skripten/matlab.pdf`

Kermit Sigmon, *MATLAB Primer*, Third Edition, 1993, `http://ise.stanford.edu/Matlab/matlab-primer.pdf` „Klassiker“, ist nun zu einem Buch geworden

`http://www.utexas.edu/its/rc/tutorials/matlab/matlab70.pdf` ausführliches MATLAB 7-Tutorial, 124 Seiten

Verfügbarkeit von MATLAB

Auf

`https://www.academic-center.de/cgi-bin/product/P13511!203513!STUD`

(einem deutschen Internet-Softwareversand) wird eine Studentenversion von MATLAB angeboten, die leider mit 87 EUR ziemlich teuer ist.

Alternativen zu MATLAB sind die folgenden beiden MATLAB-„Clones“, die eine ähnliche Syntax wie MATLAB haben, aber nicht alle Features von MATLAB enthalten, und die man kostenlos vom Internet herunterladen kann:

`http://www.octave.org` GNU Octave (Testversion 2.1.72 vom November 2005)

`http://www.scilab.org` Scilab (Version 4.0)