Einführung in MATLAB

W. HUYER

Die folgende Beschreibung bezieht sich auf MATLAB 8.4.0.150421 (R2014b) unter Windows 7 (PC-Labor Stand September 2015).

Inhaltsverzeichnis

1 Grundlagen

Die Software MATLAB (Abkürzung für MATrix LABoratory) wurde Ende der 70er-Jahre mit dem Ziel entwickelt, wichtige Algorithmen der numerischen linearen Algebra, wie das Lösen von linearen Gleichungssystemen, Eigenwertberechnungen usw. in einer interaktiven Umgebung durch einfache und intuitiv zu benutzende Funktionsaufrufe bereitzustellen. Obwohl MATLAB auch symbolische Berechnungen erlaubt, liegt der Schwerpunkt im Gegensatz zu Computeralgebrasystemen primär auf der numerischen Behandlung mathematischer Probleme, d.h. der Berechnung mit endlicher Genauigkeit.

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 kompiliert werden müssen, sondern während der Ausführung interpretiert werden.

MATLAB wird dadurch gestartet, dass man im Startmenü unter *Alle Programme* die Zeile *MATLAB R2014b* auswählt. Nach dem Erscheinen des MATLAB-Logos und dessen Verschwinden öffnet sich ein Fenster mit der MATLAB-Oberfläche.

Die MATLAB-Oberfläche sieht im Default-Format folgendermaßen aus. Unterhalb der Überschrift *MATLAB R2014b* hat sie eine Menüband und darunter weitere Fenster. Die oberste Menüzeile enthält die Wörter *HOME*, *PLOTS* und *APPS*, ein paar Symbole und ein Feld, in dem man einen Begriff eingeben kann, nach dem in der Online-Dokumentation gesucht wird. Die Einstellung *HOME* (Default), *PLOTS* oder *APPS* bestimmt das Aussehen des mittleren Bereiches der Menüleiste. Abgeschlossen wird der Menübereich mit einer Zeile, in der das aktuelle Verzeichnis angezeigt wird und in der man das Verzeichnis wechseln kann.

Die eigentliche MATLAB-Oberfläche ist in dieser Version vertikal zweigeteilt (*two column layout*). Das rechte Fenster ist das Kommandofenster, der wichtigste Teil von MATLAB, in dem die Ein- und Ausgabe erfolgen. Wenn die Kopfzeile *Command Window* dunkelblau ist, ist das Kommandofenster aktiviert, und man kann etwas eintippen. Der Doppelpfeil `>>` (MATLAB-Systemprompt) zeigt an, dass das Kommandofenster zur Verarbeitung einer Eingabe bereit ist.

Die linke Seite der Oberfläche ist horizontal dreigeteilt und enthält oben das Fenster *Current*

Folder, in der Mitte eine Anzeige mit *Details* und unten das Fenster *Workspace*. Das Fenster *Current Folder* zeigt die im aktuellen Verzeichnis enthaltenen Dateien und Verzeichnisse. Wenn man eine Datei oder ein Verzeichnis anklickt, erscheint im unteren Teil Information darüber oder „No details available“. Im Fenster *Workspace* (Arbeitsbereich) 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. „3×4 double“ für eine 3×4-Matrix) und ihr Minimum und Maximum (nur interessant bei Vektoren und Matrizen). Nach dem Start ist dieses Fenster leer, aber es füllt sich, sobald man im Kommandofenster Operationen ausführt.

Geht man mit der Maus auf Trennlinien zwischen den Teilfenstern, so kann man die Breite und Höhe der Fenster verstellen. Durch das einmalige Klicken auf das „Pfeil nach unten“-Symbol am rechten oberen Rand eines Teilfensters erhält man ein Menü, in dem man z.B. ein Teilfenster schließen (Close) oder in ein eigenes Fenster verfrachten (Undock) kann. Die übrig gebliebenen Fenster werden dann entsprechend größer. Oben im Menü kann man unter *Layout* verschiedene Einstellungen der MATLAB-Oberfläche auswählen, z.B. *Default*. Es ist Geschmackssache, was man in der MATLAB-Oberfläche haben will, aber das Kommandofenster ist auf jeden Fall nötig und sollte lang und breit genug sein, damit man viel ohne Scrollen sehen kann. 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 die Eingabe von `quit` oder `exit` im Kommandofenster, die Eingabe von `Strg-Q` und das Schließen des Fensters rechts oder links oben.

MATLAB enthält eine ausführliche Online-Dokumentation. Wenn man im Help nur schmökern will, ist es am besten, oben im Menü *Help* → *Documentation* zu wählen, und man erhält ein Help-Fenster mit Hyperlinks zum Navigieren. Gezielte Information kann man mit Hilfe des Suchfeldes rechts oben im MATLAB-Fenster suchen (Dann öffnet sich auch ein Help-Fenster). Wenn man den Namen `functionname` einer Funktion kennt, über die man etwas wissen will, kann man auch im Kommandofenster `help functionname` eingeben und erhält eine Kurzbeschreibung und einen Link zum ausführlichen *Help*.

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. Mit der Anweisung `clc` („clear command window“) kann das Kommandofenster geleert werden.

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 =
   3.141592653589793
```

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 dem Anklicken von *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 (*Numeric 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.

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. Wir erhalten 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 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 eine positive ganze Zahl k und nichtnegative $n_1, \ldots, n_k \in \mathbb{Z}$. 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 `<ENTER>`-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 `<ENTER>` 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 naheliegende 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
```

Eine Eingabe von `A(5,5) = 1` veranlasst MATLAB dazu, die Matrix `A` zu einer 5×5-Matrix zu vergrößern und die nicht festgelegten Matrixeinträge mit Nullen zu belegen. Überprüfen Sie das Ergebnis der Anweisungen `B=[A;A]` und `C=[A A]`. Man kann also Matrizen aus Blöcken zusammensetzen, vorausgesetzt, die Dimensionen passen.

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)
`randi`	Matrix aus gleichverteilten ganzen Zufallszahlen in einem vorgegebenen Bereich

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(3)` ist die 3×3-Einheitsmatrix und `eye(4,3)` eine 4×3-Matrix mit Einsern auf der Hauptdiagonale.

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`, `randn` und `randi` funktionieren ähnlich wie `zeros` und `ones`, nur werden die Elemente mit Zufallszahlen und nicht mit Nullen bzw. Einsern belegt. Bei `randi` legt zusätzlich das erste Argument den Bereich fest, aus dem die Zufallszahlen stammen sollen, z.B. generiert `randi(5,2,3)` eine 2×3-Matrix mit ganzzahligen Einträgen zwischen 1 und 5, `randi([-5 5],2,3)` eine 2×3-Matrix mit ganzzahligen Einträgen zwischen -5 und 5. `rand` und `randn` ohne Argument liefern eine gleichverteilte bzw. normalverteilte Zufallszahl.

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`	$\vert\ \vert$, sign, vorzeichenbehafteter Divisionsrest, Divisionsrest
`fix`	`floor`	`ceil`	`round`	Runden nach 0, unten, oben, zur nächsten ganzen Zahl
`sqrt`				Quadratwurzel

Eine Auflistung aller elementaren mathematischen Skalarfunktionen erhält man mit dem Befehl `help elfun`.

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)`. Dabei werden die Elemente spaltenweise von `A` genommen und spaltenweise in die neue Matrix geschrieben; z.B.:

```
>> A = [1 2 3 4; 5 6 7 8]; B = reshape(A,4,2)
B =
     1     3
     5     7
     2     4
     6     8
```

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 Warnung ausgegeben. Falls die Matrix nicht quadratisch ist, erzeugt MATLAB eine Lösung, die die Summe der Abweichungsquadrate minimiert; siehe `help mldivide` („matrix left divide“) 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 $\pi/60$ und speichern anschließend das Feld `t` auf einer Datei mit dem Namen `io.mat` (die Endung `.mat` wird automatisch hinzugefügt):

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

Wenn Sie die Datei in einer anderen MATLAB-Sitzung laden wollen, dann geben Sie einfach `load io` ein. 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 Arbeitsbereich befindlichen Variablen auf `test.mat` gespeichert, mit `save` auf der Default-Datei `matlab.mat`. Dasselbe kann man auch mit Anklicken von *Save Workspace* in der Menüleiste erreichen und wird nach dem Namen des `mat`-Files gefragt wird, auf dem man die Daten speichern soll (Default `matlab.mat` im aktuellen Verzeichnis).

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

2 Graphik

2.1 Linien und Punkte

Zur graphischen Darstellung von Linien und Punkten steht die Funktion `plot` zur Verfügung. Diese Funktion erwartet als Argument zwei Vektoren gleicher Länge, wobei der erste die x-Koordinaten und der zweite die y-Koordinaten der darzustellenden Punkte enthält. Standardmäßig werden die Punkte mit Geradenstücken verbunden. Z.B., wenn `x` und `y` Vektoren der Länge `n` sind, so produziert `plot(x,y)` einen Streckenzug durch die Datenpunkte `(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 die Graphen der Sinus- und der Cosinusfunktion zwischen 0 und 2π:

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

Nützlich ist auch (anstelle der ersten Zeile) der Befehl `x = linspace(0,2*pi,200)`, der einen Zeilenvektor aus 200 äquidistanten Punkten im Intervall $[0, 2\pi]$ erzeugt, wobei 0 und 2π dazugehören.

Eine Variante, um eine Funktion zu zeichnen, ist also, 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 oder mit `linspace` 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)`. Im obigen Beispiel wurden sogar auf diese Weise zwei Funktionsgraphen erzeugt.

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.

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) und cos(x)')
```

Weitere nützliche Befehle sind:

`axis([xmin xmax ymin ymax])`	verändert das Fenster des aktuellen Graphen zu $[\texttt{xmin}, \texttt{xmax}] \times [\texttt{ymin}, \texttt{ymax}]$
`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 -dpdf plot.pdf` ein PDF-File, `print -deps plot.eps` ein „encapsulated“ Postscript-File (geeignet zum Einbinden in LaTeX) und `print -djpeg` ein JPEG-File.

2.2 Plotten von Funktionen

Eine andere Möglichkeit zum Plotten von Funktionen bietet der Befehl `fplot(fun,lims)`. Die Funktion `fun` (eine Character-Variable oder ein Function Handle, siehe Abschnitt 3.7) wird im durch `lims` (Vektor der Länge 2) gegebenen Intervall geplottet. Beispiele sind

```
>> fun = '1/(1+x.^2)'; lims=[-5 5]; fplot(fun,lims)
```

und

```
>> fplot('sin',[0 2*pi])
```

Man beachte im ersten Beispiel die punktweise Operation nach `x`. Das zweite Beispiel zeigt, dass der `fplot`-Befehl auf eingebaute oder selbst programmierte Funktionen einer Variablen angewendet werden und dass man den Namen dieser Funktion in Hochkommas als erstes Argument eingeben kann. `fplot(fun,lims,n)` plottet die Funktion unter Benutzung von mindestens `n` + 1 Punkten; siehe `help fplot`.

2.3 3D-Graphik

Einen einfachen 3D-Plot (Fläche) 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`.

Mit `plot3(x,y,z)` kann man einen dreidimensionalen Plot einer Menge von Datenpunkten generieren, wobei `x`, `y` und `z` gleich lange Vektoren sind (z.B. Darstellung einer Parameterkurve im Raum).

3 Programmieren in MATLAB

3.1 MATLAB-Skripts

Alle Ausdrücke, die nach dem MATLAB-Prompt eingegeben werden können, können auch auf einem Textfile gespeichert und als Skript ausgeführt werden. MATLAB besitzt einen eingebauten Editor. Der Filename eines MATLAB-Skripts oder einer MATLAB-Funktion muss mit `.m` enden, und das Skript wird in MATLAB ausgeführt, indem man den Filenamen ohne `.m` eingibt (im Kommandofenster, in einem anderen Skript oder in einer MATLAB-Funktion). Ein MATLAB-Skript dient also der automatischen Ausführung einer Folge von MATLAB-Befehlen, hat weder Eingabe- und Ausgabeparameter und benutzt denselben Basis-Arbeitsbereich wie das interaktive Befehlsfenster, d.h., die im Skript verwendeten Variablen sind – falls sie nicht explizit mit `clear` (siehe Abschnitt 4.7) gelöscht werden – auch nach Ablauf des Skriptes noch im Basis-Arbeitsbereich vorhanden und es kann auch auf alle im Basis-Arbeitsbereich vorhandenen Variablen im Skript zugegriffen werden. Die Ausführung eines Skripts bedeutet nichts anderes als das Einfügen des Inhaltes des Skripts an dieser Stelle. Das Skript muss entweder im aktuellen Arbeitsverzeichnis oder im MATLAB-Suchpfad (siehe Abschnitt 4.6) liegen.

Durch Klicken auf *New Script* ganz links im Menüband öffnet sich ein Editor-Fenster mit dem Titel *untitled* über dem Kommandofenster, außerdem ändert sich das Menüband. Mit *Save* kann man das Eingegebene speichern (Default ist dann `untitled.m`), mit *New* ein neues Skript (*Script*) oder eine Funktion (*function*) generieren, mit *Open* eine bereits vorhandene Datei öffnen, mit *Run* das gerade geladene Skript ausführen. Rechts oben in der Menüleiste befinden sich Symbole für die üblichen Funktionen wie *Cut*, *Copy*, *Paste* und *Undo*. Den Editor beendet man, indem man auf das Kreuzchen am rechten oberen Rand des Editor-Fensters klickt (dann erscheint die Erklärung *Close Editor*); dabei wird noch einmal nachgefrabt, ob noch nicht gespeicherte Dateien gespeichert werden sollen.

3.2 MATLAB-Funktionen

Man kann eigene Funktionen schreiben und zur MATLAB-Umgebung hinzufügen. Diese Funktionen sind eine spezielle Art von m-Files und werden auch mit dem eingebauten MATLAB-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. Mit dem Menüpunkt *New* → *Function* (siehe oben) öffnet der MATLAB-Editor ebenso eine neue Datei, aber in der ersten Zeile wird der Benutzer schon darauf hingewiesen, dass eine Funktion mit einer Zeile

der Form `function [ output_args ] = filename( input_args )` beginnen und darauf ein Kommentar mit der Beschreibung der Funktion folgen soll. Das `end` in der letzten Zeile einer Funktion ist hingegegen nicht unbedingt nötig.

MATLAB-Funktionen können also Eingabe- und Ausgabeparameter haben. Jede Funktion besitzt einen eigenen lokalen Arbeitsbereich, d.h., die in der Funktion definierten Variablen sind nur lokal in dieser Funktion sichtbar und existieren nach Ende der Funktion nicht mehr. In der Funktion kann außerdem nicht direkt auf die Variablen im Basis-Arbeitsbereich des Befehlsfensters zugegriffen werden, sondern nur über die Eingabeparameter.

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

```
function h = pyt(a,b)
% pyt berechnet die Laenge der Hypotenuse eines rechtwinkligen
% Dreiecks nach dem Satz von Pythagoras. Eingaben sind die
% Seitenlaengen 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. 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 Skript auch ein Skript mit dessen Namen (ohne `.m`) aufrufen. Dann werden alle Befehle aus dem Skript an dieser Stelle eingefügt, aber alle Variablen behalten ihren Namen, d.h., 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] = testfcn(a,b)`. Im Programm müssen den Feldern `x` und `y` Werte zugewiesen werden. Ein leeres Klammerpaar `[]` wird benutzt, wenn die Funktion keine Ausgabeparameter besitzt

Eine Zeile, die mit `%` beginnt, ist eine Kommentarzeile, und ein `%`-Zeichen innerhalb einer Zeile bedeutet, dass der Text danach ein Kommentar ist. 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.

Genau genommen wird mit dem Aufruf `testfcn` immer die erste Funktion in `testfcn.m` ausgeführt, denn im Anschluss an diese Hauptfunktion können noch weitere Funktionen definiert werden, die dann aber nur in diesem m-File bekannt sind und nur hier mit ihrem Funktionsnamen aufgerufen werden können, aber z.B. nicht direkt vom Kommandofenster aus oder von einer anderen Funktion oder einem anderen Skript. Als Beispiel betrachten wir eine Variante der obigen Funktion `pyt.m`:

```
function h = pyt(a,b)
h = sqrt(quadrat(a)+quadrat(b));

function y = quadrat(x)
y=x.*x;
```

Möchte man auf eine Variable des Basis-Arbeitsbereich auch in einer Funktion zugreifen, ohne sie mittels Parameterliste zu übergeben, oder von mehreren Funktionen aus auf dieselbe Variable zugreifen, so muss diese Variable überall dort, wo sie sichtbar sein soll, vor ihrer ersten Benutzung mit dem Schlüsselwort `global` deklariert werden, und besitzt überall denselben Namen. Genaueres findet man mit `help global`.

Die Ausführung einer Funktion wird üblicherweise beendet, wenn das Ende des Codes erreicht ist. Wenn man, z.B. bei der Erfüllung einer gewissen Bedingung, eine vorzeitige Beendigung wünscht, verwendet man den Befehl `return`. Die Variablen `nargin` und `nargout`, in einer `function` verwendet, bezeichnen die Anzahl der Eingabe- bzw. Ausgabe-Parameter der Funktion. Die Größe `nargin` wird benötigt, wenn eine Funktion, abhängig von der Anzahl der Argumente, verschieden operiert, oder um fehlenden Eingabe-Parametern Defaultwerte zuweisen zu können. Die Variable `nargout` kann dazu dienen, sich die Berechnung von Größen, die gar nicht als Ausgabeparameter angegeben sind, zu ersparen.

3.3 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

Zur Illustration geben Sie die folgende Befehlsfolge ein (absichtlich ohne Strichpunkt am Ende jeder Zeile) und analysieren Sie die Ergebnisse:

```
>> A = 3*ones(2)
>> B = eye(2)
>> C = A & B
>> D = ~C
```

Die Eingabe `whos` zeigt, dass es in MATLAB auch einen logischen Datentyp `logical` gibt.

3.4 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.

3.5 Strukturierte Datentypen

Ähnlich wie in den Programmiersprachen C und C++ lassen sich in MATLAB Variablen unterschiedlichen Typs in einer Struktur (`struct`) zusammenfassen. Diese können z.B. dazu benutzt werden, zusammengehörige Parameter als eine Variable an eine Funktion zu übergeben. Ein Kreis ist z.B. bestimmt durch seinen Radius und Mittelpunkt:

```
>> circle.radius=4;
>> circle.midpoint=[1 2 3];
>> circle
circle =
      radius: 4
    midpoint: [1 2 3]
```

Eine weitere Möglichkeit, Variablen unterschiedlichen Typs in einem Feld abzuspeichern, sind die `cell`-Arrays; siehe `help cell`.

3.6 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)`.

3.7 Function Handle

Eine andere Möglichkeit, eine Funktion als Eingabeparameter einer anderen Funktion zu übergeben, ist der Function Handle, ein MATLAB-Datentyp, der einem Funktionszeiger in C entspricht; siehe `help function_handle`. Als Beispiel betrachten wir die MATLAB-Funktion `fzero` (siehe `help fzero`), die eine numerische Approximation der Nullstelle einer übergebenen Funktion berechnet:

```
h = @sin;
xstart = 3;
x0 = fzero(h,xstart)
```

Ein Function Handle wird mit dem Symbol `@` gekennzeichnet. Auch viele andere eingebaute MATLAB-Funktionen erwarten als Eingabeparameter eine Funktion, die mit Hilfe eines Function Handle aufgerufen wird.

Man kann auch einen Function Handle definieren, ohne auf eine bereits existierende Funktion zu verweisen. Die Syntax für diese sogenannte anonyme Funktion ist `handle = @(arglist) expression`, z.B.:

```
>> hpol = @(x,y) x.*x+2*y.*y-1;
>> z = hpol(1,1)
z =
     2
>> fzero(@(x) x.^3-2,0)
ans =
    1.2599
```

Im Beispiel mit der Simpson-Regel aus dem vorigen Abschnitt könnte der Eingabeparameter `fcn` auch ein Function Handle sein. In `simpson.m` steht dann bei der Auswertung der Funktion an der Stelle `x` immer `fcn(x)`, und der Programmaufruf erfolgt z.B. durch `simpson(@sin,0,2*pi)`.

3.8 Effiziente Programmierung

Man sollte, wann immer möglich, vektorisierte Operationen verwenden statt `for`-Schleifen, weil MATLAB die vektorisierte Version schneller und effizienter durchführt und der Code auch übersichtlicher ist. Mit den Befehlen `tic` und `toc` im folgenden Beispiel wird eine Stoppuhr gestartet und abgelesen:

```
>> n = 1.e6;
>> a = rand(n,1); b = rand(n,1); c = zeros(n,1);
>> tic, for i=1:n; c(i) = a(i)*b(i); end, toc  % 1. Version
>> tic, c = a.*b; toc                          % 2. Version
```

Die zweite Version ist ungefähr um einen Faktor 10 schneller. Der Vektor `c` wurde mit Nullen vorbelegt, damit keine Zeit dadurch verloren geht, den Vektor durch Zuweisung von neuen Einträgen wachsen zu lassen; vergleichen Sie (direkt nach der obigen Befehlsfolge eingegeben):

```
>> clear c
>> tic, for i=1:n; c(i) = a(i)*b(i); end, toc
```

Auch selbst programmierte Skalarfunktionen sollten so implementiert werden, dass der Aufruf mit einem Vektor oder einer Matrix als Übergabe möglich ist; vgl. Abschnitt 3.2. In der Funktion sollten daher die elementweisen Operationen benutzt werden.

3.9 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 11. Zeile von `simpson.m` und die 6. Zeile von `runsimpson.m` und die Art des Fehlers an, durch Anklicken von `simpson` wird dieses Programm im Editor geöffnet, und der Fehler kann behoben werden.

4 Verschiedenes

4.1 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.

4.2 Einlesen von Daten und Ausgabe auf eine externe Datei

Mit Hilfe eines Skripts können im ASCII-Format gespeicherte Daten eingelesen werden. 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
```

4.3 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 4.2 hat, und weist sie der Character-Variable `s` zu, z.B.:

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

www

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 `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
```

4.6 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.

```
>> addpath('\\fs.univie.ac.at\homedirs\huyerw7\Documents\MATLAB\num15')
```

das Unterverzeichnis `num15` von `MATLAB` (Eingabe des ganzen Windows-Pfades, damit es nicht nur lokal gilt) zum Pfad hinzufügen. Wenn man einen Pfad dauerhaft hinzufügen will, muss man diesen Befehl (natürlich ohne MATLAB-Prompt `>>`) in die Datei `startup.m` im Unterverzeichnis `MATLAB` hineinschreiben, die – so vorhanden – bei jedem Start von MATLAB ausgeführt wird.

4.7 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 Arbeitsbereich entfernt, was insbesondere sinnvoll ist, wenn man schon Probleme mit dem Speicherplatz hat. Mit `clear functions` werden alle Funktionen, mit `clear global` alle (in Abschnitt 3.2 kurz erwähnten) globalen Variablen und mit `clear all` alle Variablen, globalen Variablen und Funktionen gelöscht. Der letzte Befehl ist sinnvoll, bevor man eine neue, speicherintensive Rechnung beginnt.

MATLAB-Literatur

Stormy Attaway, *MATLAB. A Practical Introduction to Programming and Problem Solving*, Third Edition, Elsevier, Boston, 2014 vom Netz der Uni Wien auf `http://www.sciencedirect.com/science/book/9780124058767` online abrufbar

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

Timothy A. Davis und Kermit Sigmon, *MATLAB Primer*, Eighth Edition, Chapman & Hall/CRC, Boca Raton, 2010

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

Frank Haußer und Yury Luchko, *Mathematische Modellierung mit MATLAB. Eine praxisorientierte Einführung*, Spektrum Akademischer Verlag, Heidelberg, 2011 vom Netz der Uni Wien auf `http://www.springerlink.com/content/h153hn/#section=786206&page=1` online abrufbar

Brian R. Hunt, Ronald L. Lipsman und Jonathan M. Rosenberg, *A Guide to MATLAB for Beginners and Experienced Users*, Second Edition, Cambridge University Press, Cambridge, 2006

Alfio Quarteroni, Fausto Saleri und Paola Gervasio, *Scientific Computing with MATLAB and Octave*, Third Edition, Springer, Berlin Heidelberg, 2010

Wolfgang Schweizer, *MATLAB kompakt*, 3. Auflage, Oldenbourg Verlag, München, 2008

Christoph Überhuber, Stefan Katzenbeisser und Dirk Praetorius, *MATLAB 7 – eine Einführung*, Springer, Wien New York, 2005

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

Kermit Sigmon, *MATLAB Primer*, Third Edition, 1993, mehrfach im WWW zu finden, z.B. auf `http://www.minet.uni-jena.de/fakultaet/iam/personen/primer.pdf` „Klassiker“, Vorgänger des oben zitierten Buches

Verfügbarkeit von MATLAB

MathWorks bietet auf

`http://de.mathworks.com/academia/student_version/`

eine Studentenversion von MATLAB an, die leider nicht billig 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.gnu.org/software/octave/` GNU Octave (Version 4.0.0 vom Mai 2015)

`http://www.scilab.org` Scilab (Version 5.5.2, Testversion 6.0.0)