21 MATLAB

Britta Nestler

21.1 Grundlagen

MATLAB ist ein kommerzielles Software-Paket zur interaktiven numerischen Lösung mathematischer Probleme, wissenschaftlicher Berechnungen und zur Visualisierung in fast allen Bereichen technischer und naturwissenschaftlicher Disziplinen.

MATLAB basiert komplett auf Vektoren und Matrizen. Skalare werden als Matrizen der Dimension 1×1 und Vektoren als Matrizen der Dimension $1 \times n$ bzw. $m \times 1$ behandelt.

MATLAB wird durch die Eingabe von `matlab` gestartet, zum Beenden dient der Befehl `quit`. Die Eingabe- oder Kommandozeile für einen Rechenbefehl beginnt bei MATLAB immer mit dem Voranstellen der Eingabeaufforderung '`>>`', die als MATLAB-Prompt bezeichnet wird.

Der Befehl `help` zeigt eine Liste mit Kategorien an, für die MATLAB eine Hilfe besitzt. Mit `help topic` erhält man genauere Informationen zu einem speziellen Themengebiet.

21.2 Einfache Berechnungen

Mehrere hintereinander gestellte Eingabebefehle in einer Zeile werden durch das Setzen von Kommata '`,`' abgetrennt. Durch die Eingabe eines Semikolons '`;`' an der Stelle eines Kommas hinter einer der Zuweisungen, wird deren Anzeige ausgeblendet.

Der Zuweisungsoperator von MATLAB ist das Gleichheitszeichen '`=`'

Die Berechnung arithmetischer Ausdrücke erfolgt entsprechend den Prioritäten:

1. Größen in Klammern '`()`',
2. Potenzen '`^`',
3. Multiplikation '`*`' und Division '`/`' von links nach rechts,
4. Addition '`+`' und Subtraktion '`-`' von links nach rechts.

Das Ergebnis der ersten Rechnung wird automatisch mit `ans` bezeichnet und für die zweite Rechnung weiterverwendet. Der Wert von `ans` wird dabei überschrieben und mit einem neuen Wert belegt.

In einer Zeile können Mehrere Befehle aufgeführt werden. Sie sind jeweils durch Kommata ',' oder Semikolons ';' zu trennen. Bei der Verwendung des Semikolons wird die Ausgabe unterdrückt.

Mit den Cursor-Tasten ↑ und ↓ kann man auf vorherige MATLAB-Eingaben zurückgreifen. Um einen bestimmten Befehl mit bekanntem Anfangsbuchstaben wiederherzustellen, kann der Anfangsbuchstabe gefolgt von ↑ eingegeben werden.

21.2.1 Variablen und Typen

Der Benutzer kann den berechneten Größen eigene Namen zuweisen und später auf diese Variablen zurückgreifen.

> Als Variablenbezeichnung können beliebige Buchstabenkombinationen oder Kombinationen aus Buchstaben und Ziffern jeweils beginnend mit einem Buchstaben gewählt werden.

Variablen, die innerhalb von Funktionen als Eingabe- und Ausgabeparameter verwendet werden, sind üblicherweise lokale Variablen. Operationen auf lokalen Variablen und Zuweisungen an diese innerhalb von Funktionen haben keine Seiteneffekte und damit auch keinen Einfluss auf die Werte von globalen Variablen.

In Skriptdateien (siehe unten) gibt es keine lokalen Variablen. Dem MATLAB-Interpreter sind sofort nach dem Laden einer Skriptdatei alle darin erzeugten Variablen bekannt. In einer Skriptdatei können die Werte von globalen Variablen verändert werden.

Eingebaute Variablen. Beim Starten von MATLAB wird automatisch die eingebaute Variable *i* zur Verfügung gestellt. Sie bezeichnet eine imaginäre Einheit und steht für den Benutzer zur Benennung von Größen nicht zur Verfügung. Spezielle Namen wie z.B. `eps, pi` sind bei MATLAB vorbelegt und sollten daher ebenfalls nicht zur Namensgebung und Zuweisung verwendet werden.

Zahlentypen. MATLAB erkennt verschiedene Arten von Zahlen.

Tabelle 21.1: Zahlentypen

Zahlentyp	Beispiel
Ganze Zahl	2648, -54279
Reelle Zahl	3.861, -18.57
Komplexe Zahl	6.16+2.1i $i = \sqrt{-1}$
Inf `Inf`	unendlich
`NaN`	Not a Number

Es wird die so genannte wissenschaftliche Notation verwendet.

Beispiel:

```
-2.1475e+03 = -2.1475 * 10^3 = -2147.5
```

Die Berechnungen in MATLAB erfolgen in der Genauigkeit `double`. Das Ausgabeformat lässt sich durch den Befehl `format` steuern.

21.2.2 Funktionen

MATLAB kennt die folgenden eingebauten Funktionen

- Elementare Funktionen `abs`, `sqrt`,`exp`, `log`, `log10`
- Trigonometrische Funktionen `sin`, `cos`, `tan`, `sinh`, `cosh`, `tanh` und deren inverse Funktionen `asin`, `acos`, `atan`, `asinh`, `acosh`, `atanh`,
- Rundungsfuktionen `round` (Runden zur nächsten Integer-Zahl), `floor` (Abrunden), `fix` (Runden gegen null) und `ceil` (Aufrunden).

Beispiel für die Anwendung:

```
>> x = 9; sqrt(x)
            ans =
                  3
```

21.2.3 Speicherung von Sitzungen

Die Eingabe von `diary mysession` und bewirkt, dass der nachfolgende Text auf dem Bildschirm in einem File namens `mysession` abgespeichert wird. Die Datei wird in dem Verzeichnis angelegt, von dem aus MATLAB aufgerufen wurde. Es kann ein beliebiger Filename angegeben werden außer `on` und `off`. Die Speicherung einer MATLAB-Sitzung wird mit

```
>> diary off
```

beendet. Die Datei `mysession` kann mit einem Texteditor geöffnet und weiter bearbeitet werden.

Falls beabsichtigt ist, eine MATLAB-Sitzung während einer Berechnung zu verlassen und zu einem späteren Zeitpunkt fortzusetzen, werden durch

```
>> save thissession
```

die laufenden Werte der Variablen in einer Datei `thissession.mat` abgespeichert. Diese Datei kann nicht editiert werden. Beim nächsten Starten von MATLAB kann die Berechnung fortgesetzt werden durch die Eingabe von:

```
>> load thissession
```

21.2.4 Skriptdateien

Ein `Skriptdatei` ist eine ASCII-Textdatei, die eine Folge von MATLAB-Anweisungen enthält. Es ist erforderlich, dass der Dateiname die Endung '`.m`' besitzt.

Die MATLAB-Anweisungen in einer Skriptdatei lassen sich in einer MATLAB-Sitzung durch Eingabe des Dateinamens ausführen. Das Anlegen von Skriptdateien ist bei häufiger Verwendung von Anweisungsabfolgen besonders sinnvoll und bewirkt eine deutliche Einsparung von Arbeits- und Zeitaufwand.

Es wird nur das Ergebnis der Befehle am Bildschirm angezeigt, nicht aber die Befehle selbst. Die in der Datei stehenden Anweisungen werden sichtbar durch vorherige Eingabe von

```
>> echo on
```

und entsprechend durch

```
>> echo off
```

wieder ausgeblendet.

Kommentare. Text einer Zeile, der auf ein '`%`'-Zeichen folgt, wird bei der Ausführung einer MATLAB-Anweisung ignoriert. Auf diese Weise können in das MATLAB-Dokument erklärende Kommentare zur Vorgehensweise, als Hilfen und zur Nutzung für andere Anwender eingebracht werden.

21.2.5 Steuerungsbefehle

Eine Liste der Variablen, die innerhalb der laufenden Sitzung verwendet werden, wird erzeugt durch

```
>> whos
```

Tabelle 21.2: Allgemeine Steuerungsbefehle in MATLAB

Befehl	Bedeutung
`help`	Interne Hilfefunktion, Online-Dokumentation
`doc`	Laden von Textdokumentationen
`what`	Verzeichnisauflistung aller M-, MAT- und MEX-Files
`type`	Auflistung von M-Files
`lookfor`	Stichwortsuche in der Hilfe-Dokumentation über alle Einträge
`which`	Lokalisierung von Funktionen und Files
`who`	Auflistung der aktuell verwendeten Variablen
`whos`	Detaillierte Auflistung der aktuell verwendeten Variablen
`load`	Wiederherstellen von Variablen auf der Festplatte
`save`	Abspeichern des Arbeitsplatzes auf die Festplatte
`clear`	Entfernen der Variablen und Funktionen aus dem Speicher

Befehl	Bedeutung
`cd`	Wechseln des aktuellen Arbeitsverzeichnisses
`dir`	Auflistung aller Verzeichnisse
`delete`	Löschen des Files
`diary`	Speichern des Textes einer MATLAB-Sitzung
`cedit`	Einfügen einer neuen Befehlszeile
`clc`	Löschen des Eingabefensters
`format`	Festsetzen der Ausgabe auf Format
`quit`	Verlassen einer MATLAB-Sitzung

21.3 Grafische Darstellung von Funktionen

21.3.1 Einfache Plots

Um den Grafen einer Funktion, z.B. $y = \sin(3\pi x)$, in einem Intervall $0 < x < 1$ zu zeichnen, wird die Funktion an ausreichend vielen Punkten ausgewertet, die resultierenden Koordinatenpaare (x_i, y_i) werden durch Geradenstücke verbunden und durch den MATLAB-Befehl

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

grafisch dargestellt. Dies geschieht manuell, indem zunächst zwei entsprechende Vektoren `x` und `y` erzeugt werden.

Beispiel: Die `N+1` *Komponenten des Vektors* `x` *werde mit einem Iterator als äquidistante Zahlenfolge bestimmt, diejenigen des Vektors* `y` *durch Anwendung einer Funktion darauf.*

```
>> N = 100; h = 1/N; x = 0:h:1;
>> y = sin(3*pi*x); plot(x,y)
```

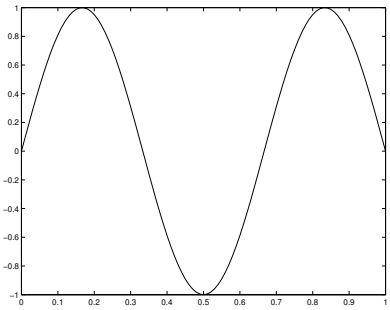

Abb. 21.1: Einfache Funktionsdarstellung in MATLAB

Die Angabe eines Titels und die Beschriftung der Achsen wird erreicht durch die Befehle

```
>> title ('Titel der Abbildung')
>> xlabel ('x-Achse')
>> ylabel ('y-Achse')
>> legend ('Legende zu dem Grafen')
```

Mit dem Befehl `grid` wird ein gepunktetes Gitter in die Grafik eingeblendet und mit `grid off` wieder ausgeschaltet.

Die Zeichenkette innerhalb der Strings `'...'` kann frei gewählt werden. Die Legende wird von MATLAB in eine geeignete Position gelegt, in der sie den Grafen nicht behindert. Sie kann manuell mit der Maus verschoben werden. Die Default-Einstellung zeichnet den Funktionsgrafen mit einer schwarzen Linie ohne Strichelung. Sowohl die Farbe als auch der Linientyp sind einstellbar .Durch Auflistung der einzelnen Funktionsausdrücke werden mehrere Kurven in das Grafikfenster gezeichnet.

21.3.2 Weitere Grafikbefehle

Der Befehl

```
>> axis([x_a  x_e  y_a  y_e])
```

besitzt vier Parameter. Die ersten beiden Größen legen das Minimum `x_a` und Maximum `x_e` entlang der x-Achse fest, die letzten beiden Werte `y_a` und `y_e` den Bereich entlang der y-Achse. Die Skalierung der Achsen lässt sich mit `axis` nachträglich noch beliebig zur Gestaltung des Grafen verändern.

Zusätzlich ist die genaue Positionierung eines erklärenden Textes innerhalb eines Grafen möglich mit der MATLAB-Anweisung

```
>> text ()
```

Mit `zoom` lässt sich ein Teil einer Zeichnung detaillierter auflösen (Mausklick links/rechts bzw. Auswahl). Ein Grafikfenster wird mit `subplot` in eine $m \times n$-Matrix aus kleineren Fenstern unterteilt, in die einzelne Diagramme gezeichnet werden können. Die Teilfenster lassen sich mit 1 bis mn zeilenweise durchnummerieren, beginnend mit dem linken oberen Teilfenster. Mit den Befehlen

```
>> subplot(221), subplot(222), subplot(223), subplot(224)
```

teilt man ein Fenster in eine 2×2-Matrixanordnung mit vier Bildern. Die ersten beiden Ziffern geben die Dimension der Bildermatrix an, die letzte Ziffer legt die Nummer des laufenden Bildes fest. Durch einen neuen Aufruf `plot` wird das Grafikfenster gelöscht und der aktuelle Graf in ein neues Fenster gezeichnet. Um eine Grafik zu verwahren und das Löschen des Grafikfensters zu verhindern, dient der Befehl `hold on`. Entsprechend

gibt `hold off` das aktuelle Bild frei, jedoch ohne das Grafikfenster zu löschen. Das explizite Löschen des Grafikfensters wird durch `clg` bewirkt.

Ein MATLAB-Diagramm kann mit

```
>> print -deps datei
```

als Encapsulated PostScript (EPS) gespeichert werden.

21.3.3 Zeichnen von Oberflächen

In MATLAB ist eine Oberfläche durch eine Funktion $f(x,y)$ definiert, die jedem Koordinatenpunkt (x,y) eine Höhe $z = f(x,y)$ zuordnet. Um eine solche Fläche im Raum zu zeichnen, ist es erforderlich, zunächst die Achsenbereiche festzulegen. Dies liefert ein rechteckiges Gebiet in der (x,y)-Ebene. Weiterhin wird mit

```
>> meshgrid(Iterator,Iterator)
```

auf diesem Gebiet ein Gitter gewählt. Vgl. Abschnitt 21.4.1 zur Definition der Iteratoren. Der Funktionsausdruck wird auf jedem diskreten Gitterpunkt des erzeugten Gitters ausgewertet und mit `mesh` gezeichnet.

Die für 2-D-Grafiken aufgeführten MATLAB-Befehle lassen sich ebenfalls für 3-D-Grafiken verwenden. Bei den Anweisungen für die Achsenbeschriftung ist jedoch zu beachten, dass ein zusätzlicher Befehl für die z-Achse eingegeben wird:

```
>> zlabel ('z-Achse')
```

Beispiel: Gezeichnet wird die Oberfläche einer Funktion, siehe Abb. 21.2.

für den Bereich $2 \leq x \leq 4$ *und* $1 \leq y \leq 3$.

```
>> [X,Y]=meshgrid(2:.2:4,1:.2:3);
>> Z=(X-3).^2-(Y-2).^2;
>> mesh(X,Y,Z)
>> title('Sattel'),xlabel('x'),ylabel('y'),zlabel('z')
```

Durch den Befehl `contour` wird statt der Flächendarstellung ein Konturplot gezeichnet.

21.4 Vektoren und Vektoroperationen

21.4.1 Einfache Operationen

Ein einfacher Vektor wird in rechteckige Klammern durch Eingabe der einzelnen Vektorkomponenten und deren Abtrennung mit Leerstellen gesetzt und so einer Variablen zugeordnet.

Beispiel:

```
>> a = [1 2 3 7 5 9]
```

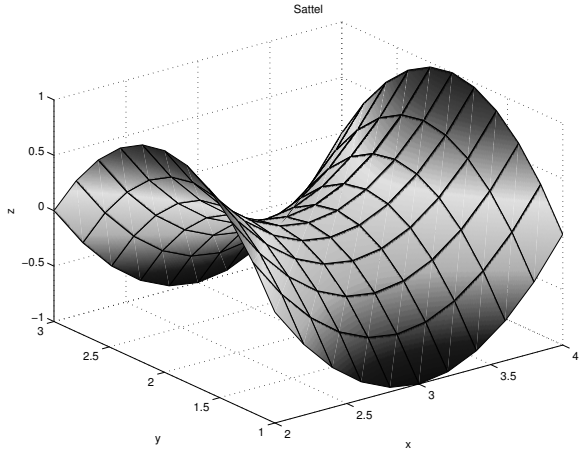

Abb. 21.2: Darstellung einer Oberfläche in MATLAB

Zeilenvektoren werden durch eine Liste von Zahlen angegeben, die entweder durch Kommata oder durch Leerstellen voneinander getrennt sind. Die Komponenten eines Vektors setzt man in rechteckige Klammern.

Mit dem Befehl `length(c)` kann die Anzahl der Einträge abgefragt werden. Der Betrag (die Norm) eines Vektors wird durch die Funktion `norm` bestimmt.

Ein Iterator hat in MATLAB die Form
`Anfang:Schrittweite:Ende`

Dies kann zur Erzeugung eines Vektors mit äquidistanten Komponenten eingesetzt werden.

Beispiel: Ein Vektor mit Elementen in Abständen von Viererschritten wird gesetzt als

```
>> t = 0:4:32
           t =   0   4   8   12   16   20   24   28
32
```

Spaltenvektoren werden generiert, indem die Komponenten durch Semikolons oder neue Zeilen voneinander abgetrennt werden. Durch Transponieren mit dem Operator `'` können Zeilenvektoren in Spaltenvektoren konvertiert werden und umgekehrt.

Die Multiplikation eines Vektors mit einem Skalar oder die Addition und Subtraktion eines Skalars zu einem Vektor wirkt sich auf jede Komponente des Vektors aus:

Beispiel:

```
>> b = t + 5
            b =
                  5   9   13   17   21   25   29   33
37
```

21.4.2 Operationen zwischen Vektoren

Die Addition und Subtraktion von zwei Vektoren ist erlaubt, wenn diese die gleiche Anzahl von Komponenten aufweisen.

Beispiel:

```
>> c = t + b
            c =
                  5   13   21   29   37   45   53
61   69
```

Das Skalarprodukt je eines gleich langen Zeilen- und Spaltenvektors wird durch den Operator '$*$' berechnet.

Beispiel:

```
>> v = [4, 2, 1]; w = [3;-1;0];
>> skalarprodukt = v * w
            skalarprodukt =
                  10
```

Eine Produktbildung zweier Vektoren ist in der Mathematik auch als Hadamard-Produkt oder Punkt-Produkt bekannt. Dabei müssen die Vektoren von derselben Länge und von demselben Typ, also beide Zeilen- oder beide Spaltenvektoren sein. Das Ergebnis ist ein Vektor desselben Typs. In MATLAB wird diese Produktbildung durch den Operator '`.*`' realisiert.

Mathematisch ist die Division eines Vektors durch einen anderen nicht definiert. MATLAB verfügt über einen Operator '`./`', der zwei Vektoren komponentenweise dividiert, diese müssen die gleiche Anzahl von Elementen besitzen und vom gleichen Typ sein.

Beispiel:

```
>> a = 1:5; b = 6:10; division = a ./ b
            division =
                  0.1667   0.2857   0.3750   0.4444
0.5000
```

Die Division durch 0 ergibt die Konstante `Inf` und ein Ausdruck der Form $0/0$ ergibt `NaN` (Not a Number).

Analog zur komponentenweisen Multiplikation und Division gibt es eine komponentenweise Potenzbildung eines Vektors. Der entsprechende Operator ist '`.^`'.

21.5 Matrizen

21.5.1 Einfache Operationen

Eine $m \times n$-Matrix ist ein rechteckiges Array von Zahlen aus m Zeilen und n Spalten. In MATLAB werden die Einträge einer Matrix zeilenweise in derselben Syntax wie für Vektoren eingegeben.

Beispiel: Eingabe einer 2×3*-Matrix.*

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

Mit dem Befehl `size(A)` lässt sich die Dimension der Matrix abfragen. Der Eintrag der `i`-ten Zeile und `j`-ten Spalte einer Matrix wird in durch die Eingabe von `A(i,j)` adressiert.

Große Matrizen lassen sich durch Aneinanderhängen aus mehreren kleineren Matrizen aufbauen. Durch die Verwendung des Operators `'` wie bei Vektoren werden Matrizen transponiert, d.h. die Zeilen und Spalten der Matrix vertauscht.

Eine bequeme Art der Erzeugung einer Diagonalmatrix aus einem Vektor ist mit dem Befehl `diag` möglich.

Beispiel:

```
>> d = [9 -8 7]; D = diag(d)
            D =
                  9     0     0
                  0    -8     0
                  0     0     7
```

Für eine gegebene Matrix A gibt der MATLAB-Befehl `diag(A)` die Diagonalelemente aus.

Ein Doppelpunkt '`:`' wählt in Abhängigkeit von der Stelle, an der seine Eingabe innerhalb des MATLAB-Befehls `A(i,j)` erfolgt, eine entsprechende Zeile oder Spalte zur Einzelangabe aus der Matrix A aus.

21.5.2 Operationen zwischen Matrizen

Ähnlich wie bei Vektoren gibt es auch bei Matrizen ein Punktprodukt. Mit dem Operator '`.*`' werden die Einträge zweier Matrizen derselben Größe komponentenweise multipliziert.

Das Produkt einer Matrix mit einem Vektor ist nur für Spaltenvektoren definiert. Diese müssen dieselbe Anzahl Einträge haben wie die Matrix Spalten besitzt. Allgemein gilt, dass das Produkt einer $m \times n$-Matrix mit einer $n \times p$-Matrix eine eine $m \times p$-Matrix zum Ergebnis hat.

Beispiel:

```
>> A = [5 7 9; 1 -3 -7]; x = [8; -4; 1]; A * x
            ans =
                  21
                  13
```

In Tabelle 21.3 sind verschiedene Matrixoperationen und lineare Algebra-Funktionen zusammengestellt.

Tabelle 21.3: Matrixoperationen in MATLAB

Funktion	Bedeutung
`norm`	Norm einer Matrix bzw. eines Vektors
`rank`	Anzahl linear unabhängiger Zeilen oder Spalten
`det`	Determinante der Matrix
`trace`	Spur einer Matrix, d.h. Summe der Diagonalelemente
`orth`	Orthogonalisierung
`null`	Nullraum
`size`	Größe einer Matrix
`length`	Länge eines Vektors
`eig`	Eigenwerte und Eigenvektoren
`poly`	Charakteristisches Polynom
`inv`	Inverse einer Matrix
`chol`	Cholesky-Faktorisierung
`qr`	Orthogonalzerlegung

21.6 Vergleiche und Kontrollstrukturen

21.6.1 Logische Operatoren

MATLAB stellt `true` und `false` durch die Integer-Werte 1 und 0 dar.

Tabelle 21.4: Logische Operatoren

Operator	Bedeutung	Operator	Bedeutung
`= =`	gleich	∼ =	ungleich
`>`	größer	<	kleiner
`>=`	größer gleich	<=	kleiner gleich

Operator	Bedeutung	Operator	Bedeutung
~	nicht	&	und
|	oder		

Die logischen Operatoren lassen sich auf einzelne Elemente, aber auch auf Vektoren und Matrizen komponentenweise anwenden. Mit '`&`' und '`|`' können einzelne Operationen verbunden werden.

Beispiel:

```
>> x = pi; x ~= 3, x ~= pi
            ans =
                  1
            ans =
                  0
>> x = [-2  3.1  5; -1  0  1];
>> test = x > 3 & x < 4
            test =
                  0     1     0
                  0     0     0
```

21.6.2 Schleifen

for-Schleife. Eine `for`-Schleife hat die generelle Struktur:

```
>> for Zaehler Anweisungen end
```

Alle Befehle zwischen den Anweisungen `for` und `end` werden so oft wiederholt, wie der `Zaehler` angibt. Dafür existieren zwei Möglichkeiten:

- Als Iterator, z.B. `Zaehler = 1 : 10`. Der Zähler nimmt dabei die Werte 1, 2, 3, ..., 10 an.
- Als Aufzählung, z.B. `Zaehler = [17  5  12  6 37]`. Der Zähler nimmt dabei die Werte 17, 5, 12 usw. an.

while-Schleife. Eine `while`-Schleife hat die generelle Struktur:

```
>> while Bedingungen Anweisungen end
```

Die `Anweisungen` werden ausgeführt, solange die `Bedingungen` erfüllt sind. Im Vorhinein ist unbekannt, wie oft die Schleife durchlaufen wird und die logische Bedingung zutrifft.

Abbruch von Schleifen. Der Abbruch einer `for`- oder `while`-Schleife ist mit dem MATLAB-Befehl `break` möglich und sinnvoll für Anwendungen, die mit Suchbefehlen oder mit Zählungen verbunden sind.

21.6.3 if-Anweisungen

Eine `if`-Verzweigung (bedingte Anweisung) hat die generelle Struktur:

```
>> if Bedingung_1 Anweisungen_1
   elseif Bedingung_2 Anweisungen_2
```

```
   .
   else Anweisungen_n
end
```

Die `Anweisungen_i` werden ausgeführt, wenn die jeweilige `Bedingung_i` wahr ist. Jede `if`-Abfrage muss bei MATLAB mit der Eingabe der Anweisung `end` abgeschlossen werden.

21.6.4 Weitere eingebaute Funktionen

Über die bereits vorher genannten elementaren mathematischen Funktionen hinaus verfügt MATLAB noch über weitere nützliche Befehle, die in Tabelle 21.5 niedergelegt sind.

Tabelle 21.5: Weitere MATLAB-Funktionen

Funktion	Anwendung
`sum(x)` `sum(A)`	Summe der Komponenten eines Vektors Summe der Matrixeinträge jeder Spalte einer Matrix A, es wird ein Zeilenvektor ausgegeben
`min(x),max(x)`	Bestimmung der kleinsten bzw. größten Komponente eines Vektors
`min(A),max(A)`	Ausgabe eines Zeilenvektors mit den jeweils kleinsten/größten Einträgen in jeder Spalte
`rand` `rand(m,n)`	Ausgabe einer willkürlichen Zahl zwischen 0 und 1, Ausgabe einer $m \times n$-Matrix mit willkürlichen Einträgen, die zwischen 0 und 1 liegen (wiederholtes Ausführen ergibt unterschiedliche Ergebnisse)
`find(logische Bedingung)`	Ausgabe einer Liste von Stellen (Indizes), an denen die Elemente eines Vektors/einer Matrix eine bestimmte logische Bedingung erfüllen

21.6.5 Eigene MATLAB-Funktionen

Mit Hilfe von Funktionsdateien ist die Definition eigener Funktionen möglich. Diese eigenen Funktionen können entweder direkt während einer MATLAB-Sitzung benutzt oder zur Überlagerung bestehender MATLAB-Implementierungen eingesetzt werden.

Dies wird erreicht, indem für die neue Funktion `Name` eine Datei `Name.m`. angelegt wird. Die erste Zeile des Files muss die Syntax-Definition für die neue Funktion enthalten, durch das entprechende Schlüsselwort `function` erkennt der MATLAB-Interpreter, dass es sich nicht um eine schrittweise abzuarbeitende Skriptdatei handelt:

```
>> function [Liste der Ausgabedaten] =
            funktions_name(Liste der Eingabedaten)
```

Innerhalb der Funktion zugewiesene Variablen sind lokale Variablen.

Dokumentation. Die Dokumentation der Funktion `Name` in der Datei `Name.m` soll in Kommentarzeilen (eingeleitet mit '`%`') eine kurze Beschreibung der Nutzung und des Aufrufes umfassen.

Der Aufruf `help Name` liest alle Kommentarzeilen aus dem File heraus und kann somit für eine Anleitung zur Bedienung der neu eingesetzten MATLAB-Funktion genutzt werden.

21.6.6 Zeitnahme

MATLAB erlaubt eine Zeitnahme für einzelne Programmabschnitte in einem Code. Die entsprechenden Funktionen heißen `tic` und `toc`.

`tic` stellt eine Stoppuhr an und `toc` stoppt diese wieder und gibt die CPU-Zeit (Central Processor Unit) in Sekunden an. Die Zeitnehmung variiert abhängig von dem Computermodell, auf dem die MATLAB-Sitzung stattfindet.

21.7 Software und Literatur

Für MATLAB ist eine Vielzahl von zusätzlichen Funktionsbibliotheken erhältlich, in denen diverse technische und naturwissenschaftliche Probleme gelöst werden, siehe Listings unter [MathWorks].

Allgemeine Literatur

Überhuber, C., Katzenbeisser, S.: MATLAB 6.5: Eine Einführung (Springer, 2002)

Gramlich, G., Werner, W.: Numerische Mathematik mit MATLAB. Eine Einführung für Naturwissenschaftler und Ingenieure. (dpunkt.verlag, 2000)

Beucher, O.: MATLAB und SIMULINK lernen (Pearson Studium, 2002)

Grupp, F., Grupp, F.: MATLAB 6 für Ingenieure (Oldenbourg 2002)

[MathWorks] The MathWorks – MATLAB User Documentation, The Language of Technical Computing `http://www.mathworks.com`

[IndUni] Indiana University, A General Overview of MATLAB, `http://www.indiana.edu/~statmath/math/matlab/overview.html`

22 LISP

Gero Schindlmayr

22.1 Übersicht

22.1.1 Entwicklung von LISP

Die Programmiersprache LISP realisiert das Konzept der **funktionalen Programmierung**. Dieses Konzept versteht ein Computerprogramm als Komposition von Funktionen. Der formale Rahmen wird durch das so genannte Lambda-Kalkül von Church gegeben, das im Rahmen der Berechenbarkeitstheorie in den 30er-Jahren entwickelt wurde. Eine erste Version von LISP entstand Anfang der 60er-Jahre durch McCarthy. Aus dieser ursprünglichen Version sind seitdem viele Varianten der Sprache hervorgegangen. Neben Anpassungen für spezielle Anwendungen hat sich COMMON LISP als Standard durchgesetzt. Seit 1994 existiert für COMMON LISP ein ANSI-Standard. Eine wichtige Erweiterung von LISP ist das COMMON LISP OBJECT SYSTEM (CLOS), das objektorientierte Konzepte in LISP integriert.

22.1.2 Einsatz von LISP

Der Name LISP ist abgeleitet von **List Processing**, wobei Listen Gruppen von Symbolen oder Zahlen darstellen. Durch Schachtelung erhält man komplexere **symbolische Ausdrücke**. Der Schwerpunkt von LISP liegt in der Manipulation solcher symbolischen Ausdrücke, die u.a. geeignet sind, sprachliche oder mathematische Inhalte zu beschreiben. Wesentliche Anwendungsfelder von LISP liegen daher im Bereich der künstlichen Intelligenz (artificial intelligence) und im Bereich mathematischer Formelmanipulationsprogramme. LISP wird allerdings auch für die Programmierung bzw. als Konfigurationssprache von Anwendungsprogrammen eingesetzt (z.B. Emacs, AutoCAD). Aus technischer Sicht liegen die Vorteile von LISP in einer sehr einfachen Syntax, die eine gleiche Form für Programme und Daten festlegt, flexiblen Mechanismen für Symbolmanipulation, Interaktivität und Plattformunabhängigkeit.

22.2 Aufbau der Sprache

LISP bietet gewöhnlich ein interaktives Front-End (toplevel loop), das LISP-Ausdrücke nach einem Prompt > einliest und sofort ausführt (evaluiert). Das Ergebnis der Ausführung wird am Bildschirm dargestellt. In den Beispielen wird das Ergebnis der Ausführung durch => von der Eingabe getrennt.