Ottmar Beucher

MATLAB und Simulink

Grundlegende Einführung für Studenten und Ingenieure in der Praxis

3., überarbeitete Auflage

ein Imprint von Pearson Education
München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Einführung in MATLAB

1

ÜBERBLICK

In diesem Kapitel werden die grundlegenden Eigenschaften und Möglichkeiten des numerischen Berechnungs- und Simulationswerkzeugs *MATLAB* dargestellt.

Ziel des Kapitels ist es, den Anfänger in den Umgang mit MATLAB einzuführen und ihn mit den Basisstrukturen dieser Software vertraut zu machen. Zum Verständnis dieser Einführung werden lediglich einige mathematische Grundkenntnisse aus der linearen Algebra (Vektor- und Matrizenrechnung) sowie der Analysis elementarer Funktionen vorausgesetzt.

Auf die Darstellung weitergehender Konzepte, insbesondere auf die Möglichkeiten, die sich aus den MATLAB-Funktionsbibliotheken (den so genannten *Toolboxes*) ergeben, muss an dieser Stelle verzichtet werden, da dies weitreichende Kenntnisse aus den Bereichen der Mathematik, der Signalverarbeitung, der Regelungstechnik und vieler anderer Gebiete erfordert und damit für Studierende des Grundstudiums, an die sich diese Einführung in erster Linie richtet, ungeeignet ist.

1.1 Was ist MATLAB?

MATLAB ist ein *numerisches Berechnungs- und Simulationswerkzeug*, das aus den ursprünglich in der Programmiersprache FORTRAN geschriebenen numerischen Funktionsbibliotheken LINPACK und EISPACK zu einem kommerziellen Werkzeug mit benutzerfreundlicher Bedienoberfläche entwickelt wurde.

Anders als bei den bekannten *Computeralgebra-Programmen* wie MAPLE oder MATHEMATICA, welche in der Lage sind, *symbolische* Operationen durchzuführen, also mathematische Formeln so berechnen können, wie ein Mensch dies normalerweise mit Papier und Bleistift tut, rechnet MATLAB prinzipiell *rein numerisch*. Auf die Computeralgebra-Funktionalität kann jedoch innerhalb der MATLAB-Umgebung durch die so genannte „Symbolics"-Toolbox zugegriffen werden. Diese ist mittlerweile fester Bestandteil von MATLAB 7 und wird auch in der Studentenversion von MATLAB 7 mitgeliefert. Hierbei handelt es sich um eine Adaption von MAPLE in der MATLAB-Sprache. Wir werden auf diese Funktionalität im Rahmen von Abschnitt 1.8 eingehen.

Computeralgebra-Programme benötigen komplexe Datenstrukturen, welche sich für den normalen Nutzer in einer komplizierten Syntax und für den Programmierer in komplexen Programmen niederschlagen. MATLAB hingegen kannte ursprünglich im Prinzip nur *eine Datenstruktur*, auf der alle seine Operationen basierten. Dies war das *numerische Feld*, also mit anderen Worten *die Matrix*. Dies schlägt sich auch im Namen nieder: MATLAB ist die Abkürzung für MATrix LABoratoy.

Dieses Prinzip wurde im Laufe der Entwicklung von MATLAB hin zu einer universellen Programmiersprache nach und nach aufgebrochen. In MATLAB 7 ist auch die Definition weitaus komplexerer Datenstrukturen möglich, wie etwa der Datenstruktur `structure`, die der aus der Programmiersprache C++ bekannten Datenstruktur `struct` ähnelt, oder auch von so genannten *Cell Arrays* bis hin zur Definition von Klassen für die objektorientierte Programmierung[1]. Mit Ausnahme der Strukturen und der Cell Arrays, auf die wir kurz in eigenen Abschnitten zu sprechen kommen werden

1 Alle Datenstrukturen – mittlerweile sind dies 15 verschiedene – lassen sich aber nach wie vor unter dem Oberbegriff des „Feldes" (Array) subsummieren. Aus MATLAB ist sozusagen ein ARRLAB (ARRay LABoratory) geworden. Das numerische Feld, also die klassische Matrix, ist in diesem Konzept quasi nur noch ein Spezialfall.

(vgl. die Abschnitte 1.3.1 und 1.3.2), wollen wir im Rahmen dieser elementaren Einführung auf diese weiter gehenden Möglichkeiten der MATLAB-Programmierung – etwa auf die objektorientierte Programmierung und die Definition eigener Klassen – nicht eingehen. Dies würde zu weitreichende Vorkenntnisse aus der Programmierung erfordern bzw. den Rahmen dieser Einführung sprengen.

Beschränkt man sich auf die grundlegende Datenstruktur der Matrix, so bleibt die MATLAB-Syntax sehr einfach und MATLAB-Programme können weit problemloser geschrieben werden als Programme in anderen Hochsprachen oder Computeralgebra-Programme. Eine ohne großen Schnickschnack konzipierte Kommandooberfläche für den interaktiven Betrieb sowie die einfache Integration eigener Funktionen, Programme und Bibliotheken unterstützt die Handhabung dieses Softwarewerkzeuges. Dies trägt auch zum schnellen Erlernen von MATLAB bei.

Wie bereits erwähnt, ist MATLAB jedoch nicht nur ein numerisches Werkzeug zur Auswertung von Formeln, sondern eine eigenständige Programmiersprache, die die Behandlung komplexer Aufgaben zulässt und über alle wesentlichen Konstrukte einer höheren Programmiersprache verfügt. Da es sich bei der MATLAB-Kommandooberfläche um einen so genannten *Interpreter* handelt und bei MATLAB um eine *Interpretersprache*, können alle Kommandos unmittelbar ausgeführt werden, was das Testen von eigenen Programmen wesentlich erleichtert.

Darüber hinaus verfügt MATLAB 7 auch über einen sehr gut konzipierten eigenen Editor mit Debugging-Funktionalität (vgl. dazu Abschnitt 1.7), welcher das Schreiben und die Fehleranalyse größerer MATLAB-Programme noch leichter macht.

Als letzter großer Vorteil ist die Interaktion mit der speziellen Toolbox *Simulink* zu erwähnen, welche wir in Kapitel 2 vorstellen werden. Hierbei handelt es sich um ein Werkzeug, mit dem Simulationsprogramme auf der Grundlage einer grafischen Oberfläche nach Art von Blockschaltbildern konstruiert werden können. Die Simulation läuft unter MATLAB, und eine leichte Durchgängigkeit zwischen MATLAB und Simulink ist gewährleistet. Auf diese und weitere Eigenschaften von Simulink werden wir im Kapitel 2 im Einzelnen eingehen.

1.2 Elementare MATLAB-Konstrukte

Ausgehend von der grundlegenden Datenstruktur des numerischen Feldes sollen im Folgenden die (nach Einschätzung des Autors) wichtigsten elementaren Konstrukte und Operationen von MATLAB vorgestellt werden. MATLAB wird dabei zunächst ausschließlich im interaktiven Betrieb verwendet. Es soll gezeigt werden, wie (numerische) Berechnungen interaktiv durchgeführt und die Ergebnisse dieser Berechnungen grafisch dargestellt und gesichert werden können.

Die *elementaren* MATLAB-Operationen lassen sich grob in fünf Klassen einteilen:

- Arithmetische Operationen,
- Logische Operationen,
- Mathematische Funktionen,
- Grafikfunktionen,
- I/O-Operationen (Datenaustausch).

Grundsätzlich handelt es sich bei all diesen Operationen um *Operationen auf Matrizen und Vektoren*. Diese werden dabei in *Variablen* gehalten, die – mit sehr wenigen

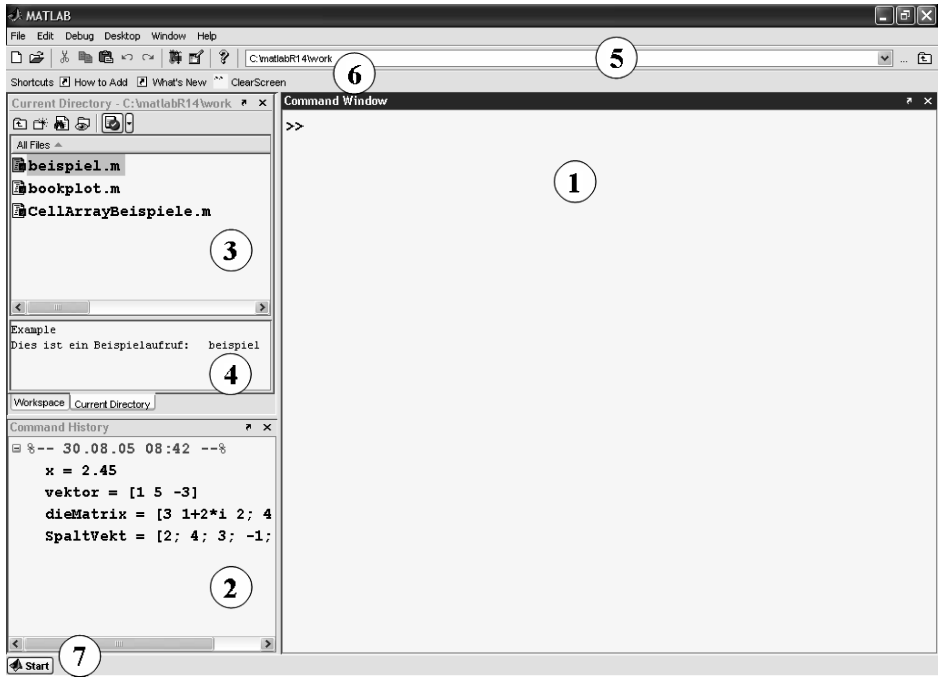

Abbildung 1.1: Die MATLAB-Kommandooberfläche nach dem Start

Einschränkungen – frei auf der MATLAB-Kommandooberfläche definiert werden können.

Die MATLAB-Kommandooberfläche präsentiert sich nach dem Start von MATLAB in der in Abbildung 1.1 dargestellten oder in einer ähnlichen Form[2].

Die wesentlichen Elemente dieser Kommandooberfläche sind:

① das Kommandofenster (*Command-Window*),

② das *Command-History* Fenster,

③ das Fenster des aktuellen Verzeichnisses (*Current Directory*) bzw. (in dieser Darstellung verdeckt) das Variablenfenster (*Workspace*),

④ das Datei-Informations-Fenster,

⑤ die Icon-Leiste mit dem Auswahlmenü für das aktuelle Verzeichnis,

⑥ die *Shortcut*-Leiste,

⑦ der Start-Button.

2 Dies hängt von den Einstellungen des Benutzers ab. Über den Menübefehl `File - Preferences` können diese Einstellungen vorgenommen werden.

Auf die Funktion der einzelnen Elemente soll in den weiteren Abschnitten an geeigneter Stelle eingegangen werden. Für den Augenblick ist nur das Kommandofenster von Bedeutung, welches die wichtigste Benutzerschnittstelle im interaktiven Betrieb darstellt.

MATLAB erwartet im Kommandofenster (①) nach dem *Eingabeprompt* » (in der Studentenversion `EDU»`) Kommandos des Benutzers, welche von MATLAB unmittelbar interpretiert und ausgeführt werden. Der Nutzer kommuniziert somit normalerweise *interaktiv* mit dem MATLAB-System. Auf die Möglichkeit der Programmierung unter MATLAB werden wir in Abschnitt 1.6 zu sprechen kommen.

Bevor wir auf Details der einzelnen Operationsklassen eingehen, soll das Konzept der MATLAB-Variablen erläutert werden, wobei wir in diesem Zusammenhang auch auf einige Besonderheiten der MATLAB-Syntax und des Umgangs mit der MATLAB-Kommandooberfläche hinweisen wollen.

1.2.1 MATLAB-Variablen

Eine MATLAB-Variable ist ein Objekt eines bestimmten Datentyps. Wie eingangs erwähnt, ist der grundlegenste Datentyp derjenige, von dem MATLAB auch seinen Namen ableitet, die *Matrix*. Da wir uns hierauf im Wesentlichen beschränken wollen, ist eine MATLAB-Variable im Folgenden grundsätzlich *eine Matrix*! Diese kann aus reellen oder komplexen Zahlen sowie aus Characters (ASCII-Zeichen) bestehen. Letzteres ist im Zusammenhang mit der Verarbeitung von *Strings* (Text) interessant. Wir wollen diesen Aspekt jedoch für den Moment einmal zurückstellen.

MATLAB-Variablen definieren

Die Matrix wird auf der MATLAB-Kommandooberfläche, im Allgemeinen durch Eingabe über die Tastatur, definiert und einem frei wählbaren Variablennamen entsprechend folgender Syntax zugewiesen:

```
» x = 2.45
```

Durch diese Anweisung nach dem MATLAB-Prompt wird die Zahl 2.45 (eine Zahl ist eine 1×1-Matrix!) der Variablen `x` zugewiesen und kann dann im Folgenden unter diesem Variablennamen angesprochen werden.

MATLAB antwortet auf diese Definition mit

```
x =

    2.4500
```

und bestätigt im interaktiven Modus damit die Eingabe. Bei syntaktischen Fehlern erfolgt eine Fehlermeldung.

Die Zahlen werden in der Voreinstellung immer mit 4 Nachkommastellen (Format `short`) dargestellt. Die Voreinstellung kann über den Menübefehl `File - Preferences ...` in der Karteikarte `Command-Window - Numeric Format` geändert werden. In den meisten Fällen ist die voreingestellte Darstellung jedoch die beste Wahl.

Die nachfolgenden Kommandos definieren einen Zeilenvektor der Länge 3 und eine 2×3-Matrix. Es ist jeweils die Reaktion von MATLAB mit angegeben:

```
» vektor = [1 5 -3]

vektor =

     1     5    -3

» dieMatrix = [3 1+2*i 2; 4 0 -5]

dieMatrix =

   3.0000             1.0000 + 2.0000i   2.0000
   4.0000                  0            -5.0000
```

Man beachte, dass die Matrix `dieMatrix` eine *komplexe Zahl* als Eintrag enthält. Komplexe Zahlen können mit Hilfe der dafür reservierten Symbole `i` und `j` in der der algebraischen Darstellung entsprechenden Form wie oben definiert werden. Nach Möglichkeit sollten deshalb diese Symbole nicht für andere Variablen verwendet werden.

Wie im obigen Beispiel zu sehen, sind die Trennzeichen für die Einträge in den Zeilen der Matrix Leerzeichen (oder alternativ Kommata) und die Spaltentrennzeichen Semikolons. Ein Spaltenvektor lässt sich somit folgendermaßen definieren:

```
» SpaltVekt = [2; 4; 3; -1; 1-4*j]

SpaltVekt =

   2.0000
   4.0000
   3.0000
  -1.0000
   1.0000 - 4.0000i
```

Unterbleibt die Zuordnung zu einem Variablennamen, so ordnet MATLAB das Ergebnis dem Namen `ans` (für *ans*wer, Antwort) zu, wie das folgende Beispiel zeigt:

```
» [2,3,4; 3,-1,0]

ans =

     2     3     4
     3    -1     0
```

Der Workspace

Sämtliche definierten Variablen werden im so genannten *Workspace* von MATLAB gespeichert. Über den Zustand des Workspace kann man sich jederzeit informieren. Das Kommando `who` liefert die Namen der gespeicherten Variablen zurück, das Kommando `whos` daneben noch weitere Informationen, darunter die oftmals sehr wichtige Information über die *Dimension der Matrix* sowie die Speicherbelegung.

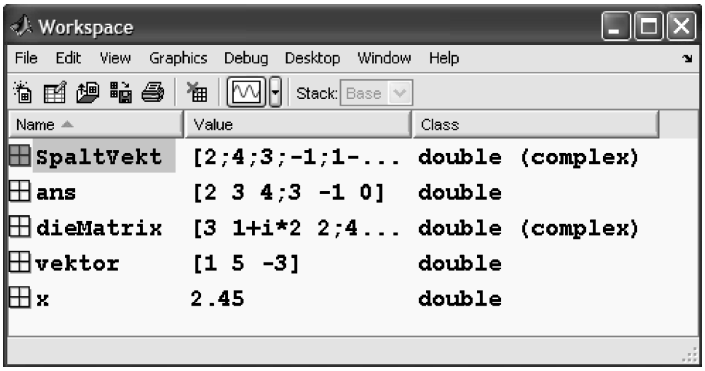

Abbildung 1.2: Der Workspace Browser

Für die in den bisherigen Beispielen definierten Variablen ergibt sich bei Aufruf dieser Kommandos:

```
» who

Your variables are:

SpaltVekt  ans        dieMatrix  vektor     x

» whos
  Name            Size         Bytes  Class

  SpaltVekt       5x1             80  double array (complex)
  ans             2x3             48  double array
  dieMatrix       2x3             96  double array (complex)
  vektor          1x3             24  double array
  x               1x1              8  double array

Grand total is 21 elements using 256 bytes
```

Eine sehr praktische Möglichkeit, sich einen Überblick über den Inhalt des Workspace zu verschaffen, liefert der *Workspace Browser*, der über den Menübefehl `Desktop - Workspace` angewählt werden kann. In Abbildung 1.1 ist der Workspace Browser bereits geöffnet und fest in der Kommandooberfläche verankert („angedockt"). Er ist in dieser Konfiguration durch die Fenster des aktuellen Verzeichnisses (③, ④) verdeckt, kann aber durch einen Klick auf den entsprechenden Reiter in den Vordergrund gebracht werden. Durch einen Klick auf den Pfeil in der Menüleiste kann man das Fenster aus der Verankerung in der Kommandooberfläche lösen[3] („ausdocken"). Dieser *Docking-Mechanismus* kann im Übrigen auf alle Fenster angewandt werden. Inwieweit man davon Gebrauch macht, ist dem Geschmack des Nutzers überlassen.

Abbildung 1.2 zeigt, wie sich der Workspace Browser im ausgedockten Zustand nach den oben abgesetzten Befehlen darstellt.

3 bzw. dort auch wieder verankern („andocken").

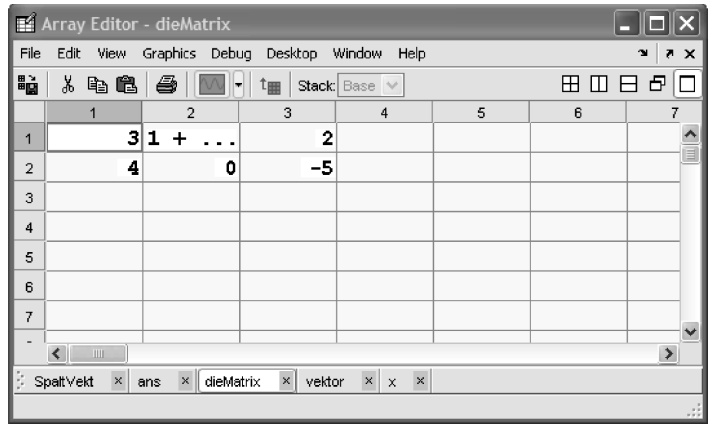

Abbildung 1.3: Darstellung einer Matrix im Array-Editor

Durch Doppelklick auf eine Variable öffnet sich der *Array-Editor* und zeigt den Inhalt der Variablen nach Manier einer Microsoft-Excel-Tabelle an (vgl. Abbildung 1.3). Es können (durch Halten der Steuerungs-Taste und Klicken) auch gleichzeitig mehrere Variablen selektiert und dann mit dem *Open Selection*-Button[4] der Toolbar geöffnet werden.

Die Matrixdimensionen, das Darstellungsformat sowie einzelne Einträge der Matrizen können im Array-Editor geändert werden. Dies ist besonders für große und daher im Kommandofenster unübersichtliche Matrizen sehr praktisch (vgl. Übung 6, Seite 21). Des Weiteren ist das Kopieren, Löschen und Anhängen ganzer Spalten oder Zeilen möglich. Auf diese Weise können mit Hilfe des Windows-üblichen `Copy-Paste`-Mechanismus über die Windows-Zwischenablage auch beispielsweise sehr leicht Daten zwischen Excel und dem Array-Editor (und damit MATLAB) manuell ausgetauscht werden (vgl. Übung 6, Seite 21).

Benötigt man Variablen nicht mehr, so kann man sie am einfachsten auf der Kommandooberfläche mit dem Kommando `clear` löschen. Beispielsweise liefert

```
» clear dieMatrix
» who

Your variables are:

SpaltVekt   ans         vektor      x
```

offenbar die Löschung von `dieMatrix`. Mit `clear` oder `clear all` wird der ganze Workspace gelöscht. Diese Operationen sind auch im Workspace Browser möglich.

Kommandos rekonstruieren

Die früher abgesetzten Kommandos bleiben gespeichert! Auf diese Weise kann man bequem ein *Kommando wiederholen* oder *abändern*. Hierzu müssen lediglich die Pfeil-

4 Die Bedeutung der Buttons kann man am einfachsten dadurch erfahren, dass man den Mauszeiger über den Button führt und einen Moment ruhig hält. Es öffnet sich ein Textfenster mit dem Namen des Button.

tasten ↑ und ↓ gedrückt werden. Die früheren Kommandos erscheinen im MATLAB-Kommandofenster und können (ggf. nach Abänderung) mit der Return-Taste neu ausgeführt werden. Die Definition der gerade gelöschten Matrix `dieMatrix` etwa kann auf diese Weise aufgefunden und die Matrix somit rekonstruiert werden.

Bei längeren MATLAB-Sitzungen ist dieses Durchhangeln durch frühere Kommandos trotzdem recht mühsam. Kennt man aber die Anfangsbuchstaben des Kommandos, so kann man den Suchvorgang abkürzen. Man tippt z. B. zum Auffinden der Matrix `dieMatrix` lediglich

```
» dieM
```

und drückt danach die Pfeiltaste ↑. Es wird dann nur noch nach den Kommandos gesucht, die mit `dieM` beginnen. Ist der Anfang eindeutig, findet man das Kommando sofort wieder.

Weitere bequeme Möglichkeiten, diesen so genannten *History-Mechanismus* zu verwenden, bietet das *Command-History*-Fenster.

In Abbildung 1.1 ist dieses Command-History-Fenster (②) links unten zu sehen. In diesem Fenster sind die in der Vergangenheit abgesetzten Kommandos unter dem jeweiligen Datum der MATLAB-Sitzung aufgelistet. Durch Verschiebung des Scroll-Balkens ist es nun sehr leicht, auch sehr viel weiter zurückliegende Kommandos aufzufinden. Ein Doppelklick auf das Kommando genügt dann, um dieses nochmals auszuführen. Weitere Anwendungsmöglichkeiten werden in Abschnitt 1.4 angesprochen.

Welche der beschriebenen Rekonstruktionsmöglichkeiten für Kommandos verwendet wird, bleibt letztlich dem Geschmack des Nutzers und praktischen Erwägungen überlassen.

Weitere Möglichkeiten der Definition von Variablen

Oft steht man vor dem Problem, eine Matrix oder einen Vektor durch weitere Komponenten zu ergänzen bzw. Spalten und Zeilen herauszulöschen.

Eine Erweiterung kann in der oben beschriebenen Form durch Anhängen an den Variablennamen geschehen. So kann die Matrix `dieMatrix` etwa durch folgendes Kommando um eine weitere Zeile ergänzt werden:

```
» dieMatrix = [dieMatrix; 1 2 3]

dieMatrix =

   3.0000             1.0000 + 1.0000i   2.0000
   4.0000                  0            -5.0000
   1.0000             2.0000              3.0000
```

Eine weitere Spalte könnte durch folgendes Kommando angehängt werden:

```
» dieMatrix = [dieMatrix, [1;2;3]]

dieMatrix =

   3.0000             1.0000 + 1.0000i   2.0000             1.0000
```

```
   4.0000                  0            -5.0000             2.0000
   1.0000             2.0000             3.0000             3.0000
```

oder mit

```
» v = [1;2;3]

v =

     1
     2
     3

» dieMatrix = [dieMatrix, v]

dieMatrix =

   3.0000             1.0000 + 1.0000i   2.0000             1.0000
   4.0000                  0            -5.0000             2.0000
   1.0000             2.0000             3.0000             3.0000
```

Will man die zweite Spalte wieder herauslöschen, so muss man diese mit einem *leeren Vektor* `[ ]` belegen. Die zweite Spalte innerhalb der Variablen `dieMatrix` wird dabei gemäß der üblichen Matrixindizierung angesprochen. Da der Zeilenindex in diesem Fall beliebig ist, wird er mit dem Platzhalter `:` gekennzeichnet. Man erhält:

```
» dieMatrix(:,2) = []

dieMatrix =

     3     2     1
     4    -5     2
     1     3     3
```

Eine Löschung der ersten Zeile wäre demnach mit

```
» dieMatrix(1,:) =[]

dieMatrix =

     4    -5     2
     1     3     3
```

zu erreichen. Ebenso kann ein Zeilen- oder Spaltenvektor herausgegriffen und einer anderen Variablen zugewiesen werden. So wird etwa mit

```
» ersteZeile = dieMatrix(1,:)

ersteZeile =

     4    -5     2
```

die erste Zeile der verbliebenen Restmatrix herausgegriffen.

Statt, wie beschrieben, entsprechende Kommandos im Kommando-Fenster auszuführen, könnten die obigen Operationen natürlich auch innerhalb des Array-Editors durchgeführt werden. Mit ein wenig Übung ist die Bearbeitung auf der Kommando-Ebene jedoch deutlich schneller. Außerdem können diese Operationen auch innerhalb von MATLAB-Programmen, also im *nicht-interaktiven* Betrieb verwendet werden (vgl. Abschnitt 1.6). Sie sind dann die einzige Möglichkeit, die gewünschten Ergebnisse zu erzielen. Die beschriebenen Techniken sind daher von großer Bedeutung, wenn man die Funktionalität von MATLAB als Programmiersprache nutzen will.

Hat man *große Matrizen oder Vektoren* zu bearbeiten, so ist die Ausgabe des Ergebnisses oft sehr störend. Ein Beispiel ist die folgende Definition eines Vektors, welcher aus den Zahlen 1 bis 5000 in Abständen von 2 besteht. Er kann in MATLAB durch folgendes Kommando, bei dem man Anfangswert, Schrittweite und Endwert angibt, einfach definiert werden:

```
» grosserVektor = (0:2:5000)

grosserVektor =

  Columns 1 through 12

     0     2     4     6     8    10    12    14    16    ...

  Columns 13 through 24

    24    26    28    30    und so weiter
```

Die Ausgabe des Vektors ist hier natürlich nicht vollständig wiedergegeben.

Will man die Ausgabe einer MATLAB-Berechnung unterdrücken, so muss das Kommando mit einem Semikolon abgeschlossen werden. Die Anweisung

```
» grosserVektor = (0:2:5000);
```

lässt MATLAB im obigen Fall verstummen.

Möchte man die MATLAB-Antwort aber nicht ganz unterdrücken, sondern das Ergebnis trotzdem auf dem Bildschirm anschauen, so kann mit dem Kommando

```
» more on
```

erreicht werden, dass die Ausgabe bei vollem Command-Window unterbrochen wird, bis der Benutzer durch Tastendruck die Fortführung der Bildschirmausgabe auslöst. Diese Funktion gilt dann für den weiteren Verlauf der interaktiven MATLAB-Sitzung. Sie kann aber durch Eingabe von

```
» more off
```

auch wieder abgeschaltet werden.

Übungen

Es wurden nun alle wesentlichen Konstrukte zur Definition von MATLAB-Variablen zusammengetragen, um die ersten Übungen zu dieser Thematik angehen zu können.

Übung 1 (*Lösung Seite 180*)

Definieren Sie unter MATLAB die folgenden Matrizen bzw. Vektoren und ordnen Sie diese entsprechenden Variablen zu:

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & j & 1 \\ j & j+1 & -3 \end{pmatrix},$$

$$k = 2.75,$$

$$\vec{v} = \begin{pmatrix} 1 \\ 3 \\ -7 \\ -0{,}5 \end{pmatrix},$$

$$\vec{w} = \begin{pmatrix} 1 & -5.5 & -1.7 & -1.5 & 3 & -10.7 \end{pmatrix},$$

$$\vec{y} = \begin{pmatrix} 1 & 1.5 & 2 & 2.5 & \cdots & 100.5 \end{pmatrix}.$$

Übung 2 (*Lösung Seite 180*)

1. Erweitern Sie die Matrix M so zu einer 6×6-Matrix V, dass sie die Form

$$V = \begin{pmatrix} M & M \\ M & M \end{pmatrix}$$

   hat.
2. Löschen Sie aus der Matrix V die 2. Zeile und die 3. Spalte heraus (Streichungsmatrix $V23$).
3. Ordnen Sie die 4. Zeile aus der Matrix V einem neuen Vektor $z4$ zu.
4. Verändern Sie in Matrix V den Eintrag $V(4,2)$ zu $j+5$.

Übung 3 (*Lösung Seite 182*)

Konstruieren Sie aus dem Vektor

$$\vec{r} = \begin{pmatrix} j & j+1 & j-7 & j+1 & -3 \end{pmatrix}$$

eine Matrix N, bestehend aus 6 Spalten, die jeweils $\vec{r}$ enthalten.

Übung 4 (*Lösung Seite 182*)

Prüfen Sie, ob der Zeilenvektor aus Übung 3 an die dort konstruierte Matrix N angefügt werden kann.

Übung 5 (*Lösung Seite 182*)

Löschen Sie alle Variablen des Workspace und rekonstruieren Sie mit Hilfe der gespeicherten Kommandos und der ↑- und ↓-Tasten die Matrix V aus Übung 2.

Führen Sie anschließend den gleichen Vorgang noch einmal mit Hilfe des Command-History-Fensters durch.

Übung 6 (*Lösung Seite 182*)

Belegen Sie die 5. Zeile der Matrix V aus Übung 2 mit Hilfe des Array-Editors mit Nullen.

Übung 7 (*Lösung Seite 184*)

Öffnen Sie die Datei **ExcelDatBsp.xls** der Begleitsoftware in Microsoft Excel. Übertragen Sie die dortigen Daten mit Hilfe der Windows-Zwischenablage in den Array-Editor.

Löschen Sie anschließend im MATLAB-Kommandofester die zweite Spalte der übertragenen Datenmatrix und kopieren Sie diese über Windows-Zwischenablage nach Excel zurück.

1.2.2 Arithmetische Operationen

Bei den arithmetischen Operationen ($+, -, *$ etc.) ist eine wesentliche Eigenschaft von MATLAB zu beachten, an die sich der Anfänger erst gewöhnen muss.

Matrixoperationen

Da die grundlegende Datenstruktur von MATLAB die *Matrix* ist, sind diese Operationen ohne weiteren Zusatz zunächst einmal als *Matrixoperationen* zu verstehen! Dies schließt ein, dass die *Rechenregeln der Matrizenalgebra* unterstellt werden, mit all den damit verbundenen Konsequenzen.

So ist z. B. das Produkt zweier Variablen `A` und `B` unter MATLAB *nicht definiert*, falls die zu Grunde liegende Matrizenmultiplikation $A \cdot B$ nicht definiert ist, das heißt konkret: falls Spaltenzahl von `A` und Zeilenzahl von `B` nicht gleich sind!

Eine Ausnahme von dieser Regel besteht nur, wenn eine der Variablen eine 1×1-Matrix, also ein *Skalar* ist. Dann wird die Multiplikation, ebenfalls den Regeln der linearen Algebra entsprechend, als *Multiplikation mit einem Skalar* interpretiert.

Die folgenden MATLAB-Kommandos[5] verdeutlichen dies an einem Beispiel:

5 Mit % können hinter die Kommandos Kommentare eingefügt werden. Dies wird insbesondere für die spätere Erstellung von MATLAB-Programmen sehr nützlich sein. Die hinter dem % bis zum Zeilenende stehenden Zeichen einer Zeile werden von MATLAB ignoriert.

```
» M = [1 2 3; 4 -1 2]           % definiert 2x3-Matrix M

M =

     1     2     3
     4    -1     2

» N = [1 2 -1 ; 4 -1 1; 2 0 1]  % definiert 3x3-Matrix N

N =

     1     2    -1
     4    -1     1
     2     0     1

» V = M*N                       % Versuch Produkt M*N

V =

    15     0     4
     4     9    -3

» W = N*M                       % Versuch Produkt N*M
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

MATLAB antwortet also auf den Versuch, die 2×3-Matrix M mit der 3×3-Matrix N zu multiplizieren, mit der 2×3-Produktmatrix V. Der Versuch, die Faktoren zu vertauschen, scheitert, da das Produkt einer 3×3-Matrix mit einer 2×3-Matrix *nicht definiert* ist. MATLAB quittiert dies mit der Meldung `Inner matrix dimensions must agree`, einer Fehlermeldung, die auch dem fortgeschrittenen MATLAB-Programmierer immer wieder begegnet.

Feldoperationen

Neben den Matrixoperationen benötigt man in sehr vielen Fällen jedoch auch eine entsprechende arithmetische Operation, welche *komponentenweise* ausgeführt werden soll.

Sind die Operationen *komponentenweise* zu verstehen, was in der MATLAB-Terminologie eine *Feldoperation* oder *Array-Operation* genannt wird, so müssen zumindest diejenigen Operationen neu gekennzeichnet werden, bei denen es zu Verwechslungen mit den Matrixoperationen kommen kann. Dies wird in der MATLAB-Syntax durch das Voranstellen eines Punktes (`.`) vor die Operation gelöst. Ein `*` alleine ist also stets als Matrixmultiplikation, ein `.*` stets als komponentenweise Multiplikation zu verstehen. Dies liefert auch bezüglich der Dimension der Objekte andere Regeln, wie das folgende Beispiel zeigt:

```
» M = [1 2 3; 4 -1 2]           % definiert 2x3-Matrix M
```

```
M =

     1     2     3
     4    -1     2

» N = [1 -1 0; 2 1 -1]          % definiert 2x3-Matrix N

N =

     1    -1     0
     2     1    -1

» M*N                           % Matrixprodukt M*N
??? Error using ==> mtimes
Inner matrix dimensions must agree.

» M.*N                          % komponentenweise Multiplikation

ans =

     1    -2     0
     8    -1    -2
```

Man erkennt, dass das Matrixprodukt $M \cdot N$ diesmal nicht definiert ist! Allerdings ist das Produkt *komponentenweise* definiert. Zu jedem Eintrag von M gibt es einen passenden Eintrag von N, mit dem das Produkt gebildet werden kann.

Ein weiteres Beispiel wäre das häufig vorkommende Quadrieren der Komponenten eines Vektors:

```
» vekt = [ 1, -2, 3, -2, 0, 4]

vekt =

     1    -2     3    -2     0     4

» vekt^2                        % Vektor vekt zum Quadrat
??? Error using ==> mpower
Matrix must be square.

» vekt.^2                       % vekt-Quadrat komponentenweise

ans =

     1     4     9     4     0    16
```

Im ersten Fall möchte MATLAB wieder eine Matrixoperation ausführen. Das Quadrieren einer Matrix ist aber nur möglich, falls die Matrix quadratisch ist, das heißt, gleiche Spalten- und Zeilenzahl hat, was hier nicht der Fall ist. Die *Komponenten* selbst können aber ohne Weiteres *quadriert* werden.