st
scientific tools

Ottmar Beucher

MATLAB und Simulink

4., aktualisierte Auflage

PEARSON
Studium

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

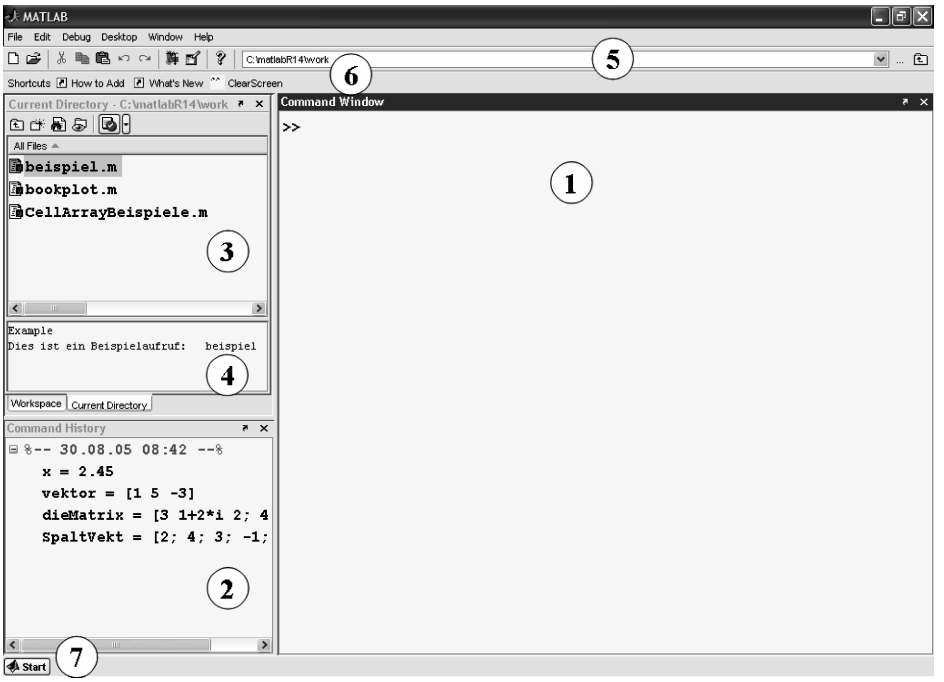

Abbildung 1.1: Die MATLAB-Kommandooberfläche nach dem Start

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.

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.

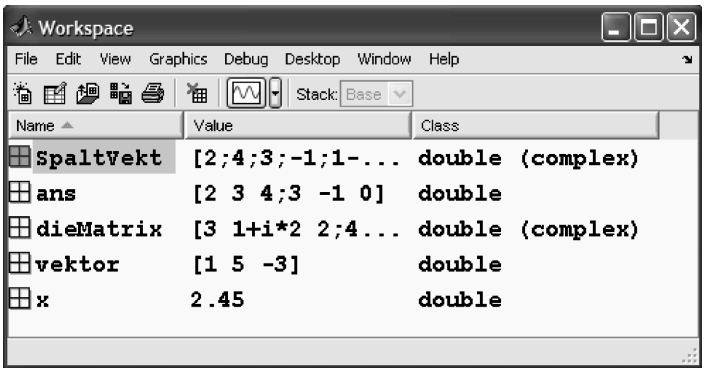

Abbildung 1.2: Der Workspace Browser

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“).

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.

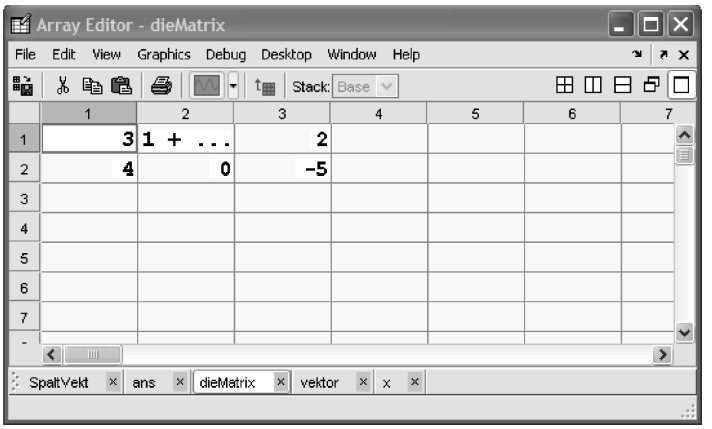

Abbildung 1.3: Darstellung einer Matrix im Array-Editor

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 23). 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 23).

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 der 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.

Die Divisionen

Besondere Bedeutung hat das Divisionszeichen. Man unterscheidet bei MATLAB zwischen *rechter Division* / und *linker Division* \.

Auch die Tatsache, dass es zwei Divisionen gibt, ist wieder eine Folge der Matrizenalgebra-Interpretation. Die *Division zweier Matrizen A* und *B*, also $X = A/B$, ist *normalerweise nicht definiert!* Der Quotient kann im Falle quadratischer Matrizen sinnvoll nur als $X = A \cdot B^{-1}$ interpretiert werden, falls die inverse Matrix B^{-1} existiert. Falls A^{-1} existiert, wäre aber auch die „Division von links“ $X = A\backslash B$ sinnvoll, interpretiert in diesem Falle als $X = A^{-1} \cdot B$.

Diesen beiden Situationen trägt MATLAB mit der Definition der linken und der rechten Division Rechnung. Wir verdeutlichen dies an einem einfachen Beispiel mit zwei *invertierbaren* 2×2-Matrizen.

```
» A = [2 1 ;1 1]           % Matrix A

A =

     2     1
     1     1

» B = [- 1 1;1 1]          % Matrix B

B =

    -1     1
     1     1

» Ainv = [1 -1; -1, 2]     % Inverse zu A

Ainv =

     1    -1
    -1     2

» Binv = [-1/2 1/2; 1/2, 1/2]  % Inverse zu B

Binv =

   -0.5000    0.5000
    0.5000    0.5000

» X1 = A/B                  % rechte Division

X1 =

   -0.5000    1.5000
         0    1.0000
```

```
» X2 = A*Binv                % Kontrollrechnung

X2 =

   -0.5000    1.5000
         0    1.0000

» Y1 = A\B                   % linke Division

Y1 =

   -2.0000    0.0000
    3.0000    1.0000

» Y2 = Ainv*B                % Kontrollrechnung

Y2 =

    -2     0
     3     1
```

Das nächste Beispiel zeigt aber, dass MATLAB bei der Interpretation der Divisionen noch einen Schritt weiter geht:

```
» A = [2 1 ;1 1]             % Matrix A

A =

     2     1
     1     1

» b=[2; 1]                   % Spaltenvektor b

b =

     2
     1

» x = A\b                    % linke Division A "durch" b

x =

    1.0000
    0.0000

» y = A/b                    % rechte Division A "durch" b
??? Error using ==> mrdivide
Matrix dimensions must agree.
```

Offenbar produziert hier die „Division von links“ $\vec{x} = A\backslash\vec{b}$ die (in diesem Fall eindeutige) *Lösung des linearen Gleichungssystems* $A\vec{x} = \vec{b}$, wie die folgende Probe beweist:

```
» A*x

ans =

    2.0000
    1.0000
```

Die rechte Division ist nicht definiert.

Linke und rechte Division lassen sich auch für nichtquadratische Matrizen verwenden, um unter- oder überbestimmte Gleichungssysteme zu lösen. Da diese Anwendungen aber weiter reichende mathematische Kenntnisse voraussetzen, als hier angenommen werden können und sollen, wird auf die Darstellung dieser Möglichkeiten verzichtet [26].

Zusammenfassung und Überblick

In Tabelle A.1.1, Seite 254, aus Anhang A sind noch einmal alle arithmetischen Operationen und ihre Wirkung als *Matrixoperationen* mit einem Beispiel zusammengefasst.

Tabelle A.1.2, Seite 255, gibt die arithmetischen Operationen und ihre Wirkung als *Feldoperationen* wieder.

Das MATLAB-Demoprogramm **aritdemo.m** der Begleitsoftware demonstriert die verschiedenen Möglichkeiten bei arithmetischen Operationen. Insbesondere lassen sich hiermit die Unterschiede zwischen Matrix- und Feldoperationen sowie das Matrixkonzept von MATLAB einüben. Es kann durch die Eingabe von **aritdemo** im Kommandofenster gestartet[6] werden.

Übungen

Bearbeiten Sie die folgenden Aufgaben zur Einübung der arithmetischen Operationen.

Übung 8 (*Lösung Seite 184*)

Starten Sie das MATLAB-Demoprogramm **aritdemo.m** durch Aufruf des Befehls **aritdemo** im MATLAB-Kommandofenster und arbeiten Sie das Programm durch.

Übung 9 (*Lösung Seite 184*)

Berechnen Sie

1. das Standardskalarprodukt der Vektoren

$$\vec{x} = \begin{pmatrix} 1 & 2 & \frac{1}{2} & -3 & -1 \end{pmatrix} \quad \text{und} \quad \vec{y} = \begin{pmatrix} 2 & 0 & -3 & \frac{1}{3} & 2 \end{pmatrix}$$

mit Hilfe einer Matrixoperation und mit Hilfe einer Feldoperation,

6 Kopieren Sie dazu das Programm in das aktuelle Arbeitsverzeichnis.

2. das Produkt der Matrizen

$$A = \begin{pmatrix} -1 & 3.5 & 2 \\ 0 & 1 & -1.3 \\ 1.1 & 2 & 1.9 \end{pmatrix} \quad \text{und} \quad B = \begin{pmatrix} 1 & 0 & -1 \\ -1.5 & 1.5 & -3 \\ 1 & 1 & 1 \end{pmatrix},$$

3. mit Hilfe einer geeigneten Feldoperation aus der Matrix

$$A = \begin{pmatrix} -1 & 3.5 & 2 \\ 0 & 1 & -1.3 \\ 1.1 & 2 & 1.9 \end{pmatrix}$$

die Matrix

$$C = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1.9 \end{pmatrix}.$$

Übung 10 (*Lösung Seite 185*)

Testen Sie die linke Division $A\backslash\vec{b}$ mit der Matrix

$$A = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$$

und dem Vektor

$$\vec{b} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

und interpretieren Sie das Ergebnis.

Übung 11 (*Lösung Seite 186*)

Testen Sie die rechte Division $\vec{b}/A$ mit der Matrix

$$A = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$$

und dem Vektor

$$\vec{b} = \begin{pmatrix} 2 & 1 \end{pmatrix}$$

und interpretieren Sie das Ergebnis.

Übung 12 (*Lösung Seite 186*)

Berechnen Sie mit Hilfe der Division von rechts (oder links) die inverse Matrix von

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ -1 & 0 & 0 \end{pmatrix}.$$

1.2.3 Logische und relationale Operationen

Auf logische und relationale Operatoren wollen wir an dieser Stelle nicht detailliert eingehen, sondern nur die grundlegendsten Dinge ansprechen. Prinzipiell handelt es sich hierbei um Operatoren, die als *Feld*operatoren agieren (das heißt, dass sie prinzipiell *komponentenweise* auf die Einträge eines Vektors bzw. einer Matrix wirken) und als Ergebnis Wahrheitswerte liefern.

Beispielsweise kann verglichen werden, ob die Komponenten zweier Matrizen an den gleichen Stellen einen Eintrag $\neq 0$ (=logisch wahr) haben. Die folgende MATLAB-Sequenz zeigt, wie dies mit Hilfe des *logischen Operators* `&` (logisches Und) bewerkstelligt wird und was das Resultat dieser Operation ist.

```
» A=[1 -3 ;0 0]

A =

     1    -3
     0     0

» B=[0 5 ;0 1]

B =

     0     5
     0     1

» Erg=A&B

Erg =

     0     1
     0     0
```

Die Ergebnismatix *Erg* enthält also überall dort eine 1 (logisch wahr), wo beide passenden Komponenten der beteiligten Matrizen gemeinsam $\neq 0$ (logisch wahr) sind, und überall dort eine 0 (logisch falsch), wo sie es nicht sind!

Ähnlich arbeiten die *relationalen Operatoren*, also die Vergleichsoperatoren. Mit der folgenden Sequenz wird geprüft, welche Komponenten der Matrix *A* größer sind als die entsprechenden Komponenten der Matrix *B*. Es werden dabei die Matrizen des vorangegangenen Beispiels verwendet.

```
» Vgl= A>B

Vgl =

     1     0
     0     0
```

Die Matrix *Vgl* zeigt an, dass nur die erste Komponente von *A* größer ist als die von *B*.

Es ist müßig, darauf hinzuweisen, dass auch hier, wie bei allen Feldoperationen, die Dimensionen der Matrizen identisch sein müssen.

Weitere relationale Operationen sind $\geq, \leq$ (in der MATLAB-Syntax sind dies `<=`, `>=`), $<$ sowie $==$ (gleich) und $\sim=$ (ungleich). Als logische Operationen hat man neben der oben besprochenen Verundung noch das logische Oder (`|`), die logische Negation ($\sim$) und das exklusive Oder `xor`.

Für weitere Informationen hierzu steht wie für die übrigen Operationen und Funktionen auch die MATLAB-Hilfe zur Verfügung (vgl. Abschnitt 1.5). Für entsprechende Hinweise zu diesem Abschnitt kann man beispielsweise über den Menübefehl `Help - MATLAB Help` nach dem Stichwort `operators` suchen. Alternativ dazu kann man im MATLAB-Kommandofenster `help ops` eingeben. In beiden Fällen erhält man dann eine Liste von MATLAB-Operatoren, in der u. a. auch die logischen und relationalen Operatoren aufgelistet sind.

Als Illustration der Möglichkeiten dieser Operatorklasse sei ein Beispiel angegeben, das in manchen Simulationen vorkommt. Die Aufgabe ist dabei, aus einem Ergebnisvektor jene Komponenten *herauszuselektieren* und zu einem Vektor zusammenzufassen, welche eine bestimmte Größenordnung, sagen wir 2, übersteigen. Dies kann mit folgender Sequenz erreicht werden:

```
» vect=[-2, 3, 0, 4, 5, 19, 22, 17, 1]

vect =

    -2     3     0     4     5    19    22    17     1

» vglvect=2*ones(1, 9)

vglvect =

     2     2     2     2     2     2     2     2     2

» vgl=vect>vglvect

vgl =
```

```
     0     1     0     1     1     1     1     1     0

» erg=vect(vgl)

erg =

     3     4     5    19    22    17
```

Zur Festlegung des Vergleichsvektors wurde hier das MATLAB-Kommando `ones` verwendet, welches zunächst eine Matrix von Einsen liefert (entsprechend den angegebenen Komponenten). Der Vergleich mit `vglvect` liefert die Stellen, an denen die Bedingung (> 2) für den Vektor `vect` erfüllt ist. Das Kommando `erg=vect(vgl)` fasst diese Komponenten mit Hilfe des Vektors `vgl` zusammen. Dabei werden nur die Indizes verwendet, für die dieser Vektor $\neq 0$ ist.

Eine solche Technik kommt oft zur Anwendung, wenn aus Messdaten Werte eliminiert werden sollen, die eine bestimmte Schwelle über- oder unterschreiten (so genannter „Ausreißer“).

Grundsätzlich müssen Vektoren und Matrizen, die mit relationalen Operatoren verknüpft werden, die gleiche Dimension haben (Feldoperation!). Daher wurde im obigen Beispiel der Vektor `vglvect` konstruiert. Falls jedoch, wie in diesem Falle, nur mit *einem* Wert verglichen wird, so kann auf diese Konstruktion verzichtet werden. Im Beispiel führt auch

```
» vgl=vect>2

vgl =

     0     1     0     1     1     1     1     1     0
```

zum Ziel.

Zu beachten ist, dass das Ergebnis einer logischen Operation *kein numerisches Feld* mehr ist. Die Eingabe

```
» whos
  Name          Size                    Bytes  Class

  erg           1x6                        48  double array
  vect          1x9                        72  double array
  vgl           1x9                         9  logical array
  vglvect       1x9                        72  double array

Grand total is 33 elements using 201 bytes
```

zeigt, dass `vgl` ein logisches Feld (*logical array*) ist, also ein Feld von Wahrheitswerten. Die Auswahloperation `erg=vect(vgl)` würde mit einem entsprechend besetzten numerischen Feld zu einer Fehlermeldung führen.

Logische Felder entstehen nicht nur als Ergebnis von Vergleichsoperationen. Man kann sie auch direkt auf folgende Weise definieren:

```
» logiFeld1 = [true, true, false, true, false, true]

logiFeld1 =

     1     1     0     1     0     1

» numFeld = [1, 1, 0, 1, 0, 1]

numFeld =

     1     1     0     1     0     1

» logiFeld2 = logical(numFeld)

logiFeld2 =

     1     1     0     1     0     1

» whos
  Name            Size                    Bytes  Class

  logiFeld1       1x6                         6  logical array
  logiFeld2       1x6                         6  logical array
  numFeld         1x6                        48  double array

Grand total is 18 elements using 60 bytes
```

Mit Hilfe der Funktionen `true` und `false` können ganze Felder mit Wahrheitswert „wahr“ oder „falsch“ definiert werden. Der Aufruf ohne Argument, wie oben, erzeugt genau einen Wahrheitswert. Auch die Umwandlung von numerischen in logische Arrays mit Hilfe der Funktion `logical` ist möglich.

Durch die Indizierung mit logischen Arrays können, wie im obigen Beispiel, sehr einfach Komponenten aus numerischen Feldern selektiert werden. Hierzu ein weiteres Beispiel, bei dem jede zweite Komponente aus einem Vektor selektiert wird:

```
» vekt = [1; i; 2; 2+j; -1; -j; 0; i+1]

vekt =

   1.0000
        0 + 1.0000i
   2.0000
   2.0000 + 1.0000i
  -1.0000
        0 - 1.0000i
        0
   1.0000 + 1.0000i
```

```
» selct = logical([0 1 0 1 0 1 0 1])

selct =

     0     1     0     1     0     1     0     1

» auswahl = vekt(selct)

auswahl =

        0 + 1.0000i
   2.0000 + 1.0000i
        0 - 1.0000i
   1.0000 + 1.0000i
```

Übungen

Bearbeiten Sie die folgenden Aufgaben zur Einübung im Umgang mit logischen und relationalen Operationen.

Übung 13 (*Lösung Seite 187*)

Testen Sie an den Matrizen A und B von Seite 30 die Wirkungsweise der logischen Operationen Oder (|), Negation (∼) und exklusives Oder `xor` aus. Konsultieren Sie dazu ggf. die MATLAB-Hilfe.

Interpretieren Sie die Ergebnisse.

Übung 14 (*Lösung Seite 188*)

Testen Sie mit Hilfe der Vektoren

$$\vec{x} = \begin{pmatrix} 1 & -3 & 3 & 14 & -10 & 12 \end{pmatrix} \quad \text{und} \quad \vec{y} = \begin{pmatrix} 12 & 6 & 0 & -1 & -10 & 2 \end{pmatrix}$$

die Wirkungsweise der relationalen Operationen <=, >=, <, == und ∼= aus. Konsultieren Sie dazu ggf. die MATLAB-Hilfe.

Übung 15 (*Lösung Seite 189*)

Betrachten Sie die Matrix

$$C = \begin{pmatrix} 1 & 2 & 3 & 4 & 10 \\ -22 & 1 & 11 & -12 & 4 \\ 8 & 1 & 6 & -11 & 5 \\ 18 & 1 & 11 & 6 & 4 \end{pmatrix}$$

Setzen Sie mit Hilfe der relationalen Operatoren alle Einträge > 10 und < -10 der Matrix auf 0.

Hinweis: Führen Sie zunächst die Vergleiche durch und verwenden Sie dann die Ergebnisse dieser Vergleiche, um die besagten Einträge mit geeigneten Feldoperationen auf 0 zu setzen.

Übung 16 (*Lösung Seite 190*)

Betrachten Sie die Matrix

$$D = \begin{pmatrix} 7 & 2 & 3 & 10 \\ -2 & -3 & 11 & 4 \\ 8 & 1 & 6 & 5 \\ 18 & 1 & 11 & 4 \end{pmatrix}.$$

Selektieren Sie mit Hilfe logischer Felder die Diagonale aus der Matrix und speichern Sie diese in einem Vektor `diag`.

1.2.4 Mathematische Funktionen

MATLAB verfügt, insbesondere dann, wenn man die Funktionalität der *Toolboxes* hinzunimmt, über eine Unzahl verschiedener vorgefertigter mathematischer Funktionen.

Für den Anfänger sind jedoch zunächst einmal die so genannten *elementaren Funktionen* relevant [21, 28, 8]. Hierbei handelt es sich um die aus der Analysis einer reellen Veränderlichen bekannten Funktionen wie Cosinus, Sinus, die Exponentialfunktion, den Logarithmus etc.

Angesichts des Datenstrukturkonzepts von MATLAB, welches ja im Wesentlichen auf *Matrizen* beruht, scheint sich hier auf den ersten Blick ein Problem aufzutun, da diese Funktionen für Matrizen ja gar nicht definiert sind.

Auf den zweiten Blick ist jedoch die Auflösung dieses Problems nach den bisher diskutierten Beispielen unmittelbar ersichtlich. Natürlich ist die Wirkungsweise einer elementaren Funktion auf einen Vektor oder eine Matrix wieder nur *komponentenweise* sinnvoll. Die nachfolgende Sequenz zeigt, was unter dem Sinus eines Vektors zu verstehen ist.

```
» t=(0:1:5)

t =

     0     1     2     3     4     5

» s=sin(t)

s =

         0    0.8415    0.9093    0.1411   -0.7568   -0.9589
```

Der Sinus des angegebenen Vektors $\vec{t}$ ist also wiederum ein Vektor, und zwar der Vektor

$$\vec{s} = (\sin(0), \sin(1), \sin(2), \cdots, \sin(5)) .$$

Die volle Bedeutung dieser Technik wird dem MATLAB-Nutzer erst bei umfangreicheren Simulationen klar. Man muss sich vor Augen halten, dass die obige Sequenz eine *Programmierschleife ersetzt!* In der Programmiersprache C++ etwa wäre die Sequenz folgendermaßen zu implementieren:

```
double s[6];
for(i=0; i<6; i++)
   s[i] = sin(i);
```

Der Gewinn ist natürlich bei diesem kleinen Beispiel noch nicht unmittelbar ersichtlich, jedoch lassen sich für größere Simulationen mit Hilfe dieser Technik sehr elegante Lösungen in MATLAB produzieren.

Wir können an dieser Stelle nicht alle elementaren mathematischen Funktionen diskutieren. Einen Überblick über die zur Verfügung stehenden Funktionen erhält man, wenn man auf der MATLAB-Kommandooberfläche das Kommando `help elfun` absetzt. Es wird dann eine Liste mit den Namen der Funktionen mitsamt einer Kurzbeschreibung ausgegeben. Für nähere Informationen gibt man `help <Funktionsname>` ein. So liefert etwa `help asin`:

```
» help asin

 ASIN   Inverse sine.
    ASIN(X) is the arcsine of the elements of X. Complex
    results are obtained if ABS(x) > 1.0 for some element.

    See also sin, asind.

    Overloaded functions or methods
    (ones with the same name in other directories)
       help sym/asin.m

    Reference page in Help browser
       doc asin
```

die Beschreibung des Arcussinus.

An dieser Stelle sollte erwähnt werden, dass Funktionsnamen und Aufrufsyntax in der Hilfe stets in Großbuchstaben angegeben sind. Dies dient jedoch lediglich der Hervorhebung im Hilfetext. Verwendet man die Funktionen im Kommando-Fenster, so müssen die Kommandos im Allgemeinen in Kleinschreibweise abgesetzt werden. Im Gegensatz zu früheren Versionen unterscheidet MATLAB 7 strikt zwischen Groß- und Kleinschreibung („Case Sensitivity“). Im obigen Beispiel führen also die Aufrufe

```
» x = 0.5;
» ASIN(X)
```

```
??? Undefined function or variable 'X'.

» X = 0.5;
» ASIN(X)
??? Undefined command/function 'ASIN'.
```

jeweils zu einer Fehlermeldung, da im ersten Fall die Variable `X` nicht existiert und im zweiten Fall die Funktion `ASIN` nicht bekannt ist.

Ein korrekter Aufruf lautet daher:

```
» x = 0.5;
» asin(x)

ans =

    0.5236
```

Neben den Funktionen aus der `elfun`-Gruppe sind noch die mit `help specfun` auflistbaren so genannten *speziellen mathematischen Funktionen* von Interesse. Diese erfordern jedoch tiefer gehende mathematische Kenntnisse und sollen hier nicht weiter diskutiert werden. Dem interessierten Leser sei jedoch die Übung 22 anempfohlen.

Ein paar weitere Beispiele sollen den Umgang mit den elementaren Funktionen unter MATLAB demonstrieren. Zunächst soll zu einem Vektor von komplexen Zahlen Betrag und Phasenwinkel in *rad* und in *grad* bestimmt werden [21]. Dies ist ein häufig vorkommendes Problem im Zusammenhang mit der Bestimmung der so genannten Übertragungsfunktion linearer Systeme [4, 9, 23].

```
» cnum=[1+j, j, 2*j, 3+j, 2-2*j, -j]

cnum =

  Columns 1 through 4

   1.0000 + 1.0000i        0 + 1.0000i        0 + 2.0000i
   3.0000 + 1.0000i

  Columns 5 through 6

   2.0000 - 2.0000i        0 - 1.0000i

» betrg=abs(cnum)

betrg =

    1.4142    1.0000    2.0000    3.1623    2.8284    1.0000

» phase=angle(cnum)
```

```
phase =

    0.7854    1.5708    1.5708    0.3218   -0.7854   -1.5708

» grad=angle(cnum)*360/(2*pi)

grad =

   45.0000   90.0000   90.0000   18.4349  -45.0000  -90.0000
```

Im folgenden Beispiel sollen alle Einträge einer Matrix (etwa mehrere Messreihen eines Spannungssignals) in Dezibel (dB) umgerechnet werden. Für einen Spannungswert U bedeutet dies, dass die Größe in

$$20 \cdot \log_{10}(U)$$

umgerechnet werden muss:

```
» mess=[25.5 16.3 18.0; ...
        2.0   6.9  3.0; ...
        0.05  4.9  1.1]

mess =

   25.5000   16.3000   18.0000
    2.0000    6.9000    3.0000
    0.0500    4.9000    1.1000

» dBmess=20*log10(mess)

dBmess =

   28.1308   24.2438   25.1055
    6.0206   16.7770    9.5424
  -26.0206   13.8039    0.8279
```

Nebenbei zeigt das obige Beispiel, wie ein zu langes MATLAB-Kommando umgebrochen werden kann, ohne dass es nach Eingabe der Return-Taste abgesetzt wird. Man gibt einfach drei Punkte[7] (…) vor dem Drücken der Return-Taste ein und kann dann das Kommando in der folgenden Zeile fortsetzen.

Im nächsten Beispiel soll eine Liste von Punkten des ersten Quadranten von $\mathbb{R}^2$ von kartesischen in Polarkoordinaten umgerechnet werden. Die Liste ist in Form einer Matrix von (x, y)-Werten gegeben. Bekanntlich (vgl. dazu auch das Beispiel auf

7 Die zur Fortsetzung der Kommandozeile verwendeten Punkte (…) sind in diesem speziellen Fall allerdings nicht mehr obligatorisch, da MATLAB 7 „merkt“, dass es sich um eine unvollständige Matrix-Definition handelt, solange die schließende Klammer nicht gesetzt ist. In früheren MATLAB-Versionen mussten diese Punkte unbedingt stehen. Nach wie vor müssen die Punkte (…) jedoch für den Zweck des Zeilenumbruchs in Kommandozeilen eingesetzt werden.

Seite 96) errechnen sich die Polarkoordinaten (r, ϕ) in diesem Fall gemäß:

$$r = \sqrt{x^2 + y^2}\,, \tag{39.1}$$

$$\phi = \arctan\left(\frac{y}{x}\right)\,. \tag{39.2}$$

Die Berechnung wird durch folgende MATLAB-Sequenz realisiert, bei der die elementaren mathematischen Funktionen `sqrt` und `atan` verwendet werden:

```
» pkte = [ 1 2; 4 3; 1 1; 4 0; 9 1]

pkte =

     1     2
     4     3
     1     1
     4     0
     9     1

» r = sqrt(pkte(:,1).^2 + pkte(:,2).^2)

r =

    2.2361
    5.0000
    1.4142
    4.0000
    9.0554

» phi = atan(pkte(:,2)./pkte(:,1))

phi =

    1.1071
    0.6435
    0.7854
         0
    0.1107

» polark = [r,phi]

polark =

    2.2361    1.1071
    5.0000    0.6435
    1.4142    0.7854
    4.0000         0
    9.0554    0.1107
```

Man beachte die Verwendung des :-Operators und der Feldoperationen bei der Berechnung von `r` und `phi`, die es gestatten, die Berechnung auf einen Schlag am ganzen Vektor durchzuführen.

Abschließend sei bemerkt, dass die obige Rechnung durch folgende Anweisung verkürzt werden könnte:

```
» pkte = [ 1 2; 4 3; 1 1; 4 0; 9 1]

pkte =

     1     2
     4     3
     1     1
     4     0
     9     1

» polark = [sqrt(pkte(:,1).^2 + pkte(:,2).^2), ...
                              atan(pkte(:,2)./pkte(:,1))]

polark =

    2.2361    1.1071
    5.0000    0.6435
    1.4142    0.7854
    4.0000         0
    9.0554    0.1107
```

Übungen

Bearbeiten Sie die folgenden Aufgaben zur Einübung der elementaren mathematischen Funktionen.

Übung 17 (*Lösung Seite 192*)

Berechnen Sie für einen Zeitvektor von Zeitwerten zwischen 0 und 10 im Abstand von 0.1 die Werte des Signals (der Funktion)

$$s(t) = \sin(2\pi 5t)\cos(2\pi 3t) + \mathrm{e}^{-0.1t}\,.$$

Übung 18 (*Lösung Seite 193*)

Berechnen Sie für den Zeitvektor aus Übung 17 die Werte des Signals (der Funktion)

$$s(t) = \sin(2\pi 5.3t)\sin(2\pi 5.3t)\,.$$

Übung 19 (*Lösung Seite 193*)

Runden Sie für den Zeitvektor aus Übung 17 die Werte des Vektors

$$s(t) = 20\sin(2\pi 5.3t)$$

einmal gegen ∞ und zum anderen gegen 0. Finden Sie hierzu die entsprechenden MATLAB-Funktionen. Nutzen Sie dabei MATLABs Hilfemechanismen. Konsultieren Sie gegebenenfalls hierzu auch vorab den Abschnitt 1.5, Seite 88ff.

Übung 20 (*Lösung Seite 194*)

Runden Sie für den Zeitvektor aus Übung 17 die Werte des Vektors

$$s(t) = 20\sin(2\pi 5t)$$

mit einer geeigneten MATLAB-Funktion gegen die nächste ganze Zahl, geben Sie jeweils die ersten 6 Werte von $s(t)$ und der zugehörigen gerundeten Werte in einer zweizeiligen Matrix aus und interpretieren Sie das etwas merkwürdige Ergebnis.

Übung 21 (*Lösung Seite 195*)

Berechnen Sie mit den passenden elementaren mathematischen MATLAB-Funktionen für den Vektor

$$\vec{b} = (1024 \quad 1000 \quad 100 \quad 2 \quad 1)$$

den entsprechenden Vektor der Zweier- und der Zehnerlogarithmen.

Übung 22 (*Lösung Seite 195*)

Führen Sie die Umrechnung von kartesischen Koordinaten in Polarkoordinaten aus dem Beispiel auf Seite 38 mit Hilfe der speziellen mathematischen Funktion `cart2pol` durch. Informieren Sie sich dazu vorab durch Eingabe von `help cart2pol` im Kommando-Fenster über die Syntax dieses Befehls.

1.2.5 Grafikfunktionen

Eine der herausragenden Stärken von MATLAB ist seine Fähigkeit zur *grafischen Visualisierung* der Berechnungsergebnisse.

Hierzu stehen MATLAB sehr einfache, aber dennoch leistungsstarke Grafikfunktionen zur Verfügung. MATLAB ist dabei in der Lage, gewöhnliche Funktionsgraphen in *zweidimensionalen Plots* (xy-Plot) und Funktionen zweier Veränderlicher in perspektivischen *dreidimensionalen Plots* (xyz-Plot) darzustellen.

Einen vollständigen Überblick über alle Grafikfunktionen erhält man durch Eingabe von `help graph2d`, `help graph3d` und `help graphics` im Kommando-Fenster bzw. über die entsprechenden Einträge in der MATLAB-Hilfe. Wir werden in diesem Abschnitt aus dieser Vielzahl von Funktionen lediglich die gebräuchlichsten und wichtigsten vorstellen.

Zweidimensionale Plots

Die wohl wichtigste, weil am häufigsten verwendete Grafikfunktion ist die Funktion `plot`. Die Eingabe `help plot` liefert folgende Information (Auszug):

```
» help plot

 PLOT   Linear plot.
    PLOT(X,Y) plots vector Y versus vector X. If X or Y is a
    matrix, then the vector is plotted versus the rows or
    columns of the matrix, whichever line up ...
```

Die MATLAB-Hilfe, auf deren vollständige Darstellung hier aus Platzgründen verzichtet werden soll, zeigt die grundlegenden Verwendungsmöglichkeiten von `plot`. Darüber hinaus gibt es jedoch noch zahlreiche andere Möglichkeiten, Optionen an `plot` zu übergeben. Eine Diskussion dieser Themen würde allerdings den Rahmen dieses Buches sprengen. Der interessierte Leser sei hierfür auf [20] verwiesen.

Wie die Hilfefunktion zeigt, werden prinzipiell zwei Vektoren benötigt, um einen Graph zu plotten. Der erste Vektor repräsentiert dabei den Vektor der x-Werte und der zweite Vektor den zugehörigen Vektor der y-Werte. Die Vektoren müssen dafür natürlich *gleiche Länge* haben. Wenn dies nicht der Fall ist, setzt MATLAB eine entsprechende Fehlermeldung ab, es sei denn, dass x ein Skalar ist (s. `help plot`). Dies kommt in der Praxis, selbst bei erfahrenen MATLAB-Nutzern, immer wieder vor, ist aber ein leicht zu korrigierender Fehler.

Es können aber auch mehrere Funktionen gleichzeitig geplottet werden, entweder indem man die x, y-Paare nacheinander in die Parameterliste schreibt oder, wenn etwa stets der gleiche x-Vektor verwendet wird, indem man die y-Vektoren zu einer entsprechenden Matrix zusammenfasst.

Zudem kann durch zusätzliche Parameter die Gestalt der Graphen in Linienart und Farbe verändert werden.

Unser erstes Beispiel greift das Beispiel auf Seite 35 nochmals auf und stellt das dort ermittelte Ergebnis grafisch dar:

```
» t=(0:1:5);
» s=sin(t);
» plot(t,s)
```

Man beachte, dass in diesem Beispiel die numerische Ausgabe der Variablen `t` und `s` unterdrückt wurde!

Der Plotbefehl öffnet ein Fenster mit der in Abbildung 1.4 dargestellten Grafik. Die Achseneinteilung wurde hier und in den nachfolgenden Grafiken gegenüber der voreingestellten Einteilung zu Gunsten einer besseren Übersichtlichkeit vergröbert. Auch die Liniendicke wurde aus diesem Grund gegenüber der Originaldarstellung etwas vergrößert. Siehe hierzu auch den entsprechenden Abschnitt auf Seite 48.

Man erkennt, dass das Ergebnis normalerweise in Form eines Polygonzugs geplottet wird. Dies führt manchmal, insbesondere wenn die Abszissenwerte weit auseinander liegen, zu Fehlinterpretationen des Plots. Also Vorsicht! Außerdem werden die Achsen automatisch nur numerisch beschriftet. Gitternetze, Achsenbezeichnungen und Titel muss man mit zusätzlichen MATLAB-Kommandos selbst anbringen.

Die folgenden Befehle generieren den Plot in Abbildung 1.5. Hierbei wurde zur Darstellung ein anderer Linienstil (Kreise) gewählt und die Farbe auf Magenta (was

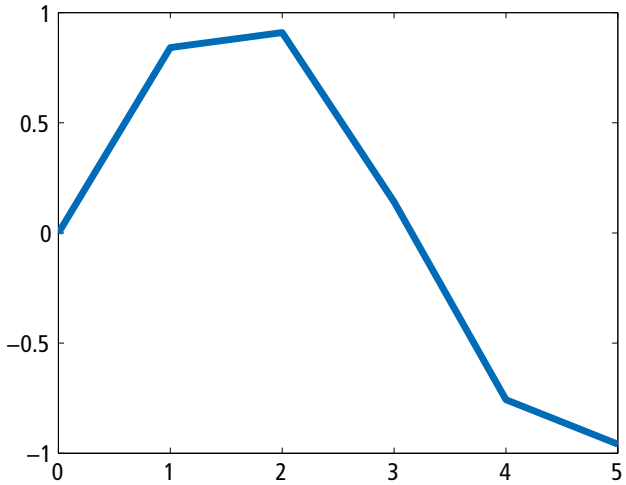

Abbildung 1.4: MATLAB-Beispiel einer einfachen x, y-Grafik

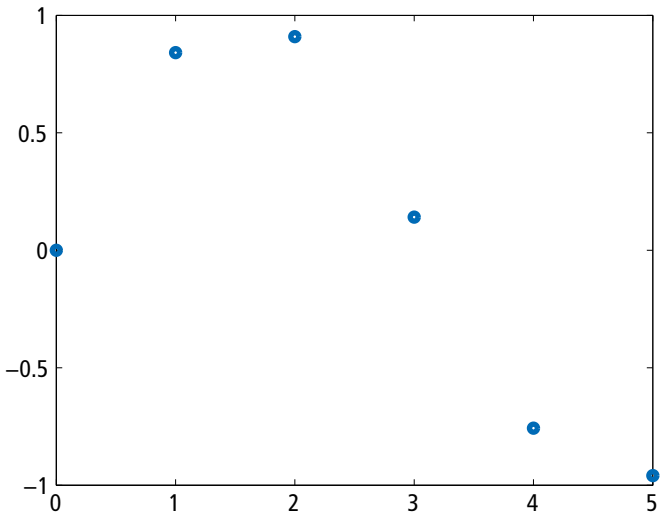

Abbildung 1.5: x, y-Grafik mit anderem „Linienstil“

hier natürlich nicht dargestellt werden kann) umgesetzt (Voreinstellung von MATLAB ist die Farbe Blau).

```
» t=(0:1:5);
» s=sin(t);
» plot(t,s,'mo')
```

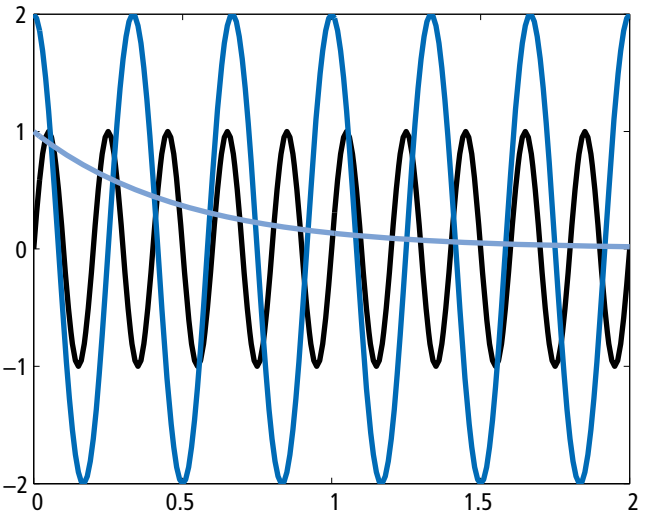

Abbildung 1.6: x, y-Grafik mit mehreren Funktionen

Im nächsten Beispiel werden für den Zeitvektor `t=(0:0.01:2)` ein Sinus der Amplitude 1 und der Frequenz[8] 5 Hz, ein Cosinus der Amplitude 2 und der Frequenz 3 Hz sowie die Exponentialfunktion e^{-2t} berechnet und in einen Graphen geplottet:

```
» t=(0:0.01:2);
» sinfkt=sin(2*pi*5*t);
» cosfkt=2*cos(2*pi*3*t);
» expfkt=exp(-2*t);
» plot(t,[sinfkt; cosfkt; expfkt])
```

Dies liefert die in Abbildung 1.6 dargestellte Grafik.

In Abbildung 1.7 ist diese Grafik nochmals dargestellt, wobei aber von den Möglichkeiten der unterschiedlichen Linienformen Gebrauch gemacht wurde, um die Funktionsgraphen voneinander zu unterscheiden. Die entsprechende Plotanweisung ist

```
» plot(t,sinfkt,'k-', t, cosfkt, 'b--', t, expfkt, 'm.')
```

Hierbei muss also die Variable `t` jedes Mal neu wiederholt werden. Die Sinusfunktion wird dabei mit den obigen Einstellungen schwarz (**k**) und mit einer durchgezogenen Linie (`-`) geplottet, die Cosinusfunktion blau (**b**lue) und gestrichelt (`--`), die Exponentialfunktion magentafarben (**m**agenta) und gepunktet (`.`).

Neben dieser wohl am häufigsten verwendeten Plotfunktion bietet MATLAB noch eine Reihe anderer zweidimensionaler Plotfunktionen an. In der Signalverarbeitung

8 Man beachte: Die allgemeine Form einer harmonischen Schwingung mit Frequenz f, Amplitude A und Nullphase φ ist $f(t) = A\sin(2\pi ft + \varphi)$.

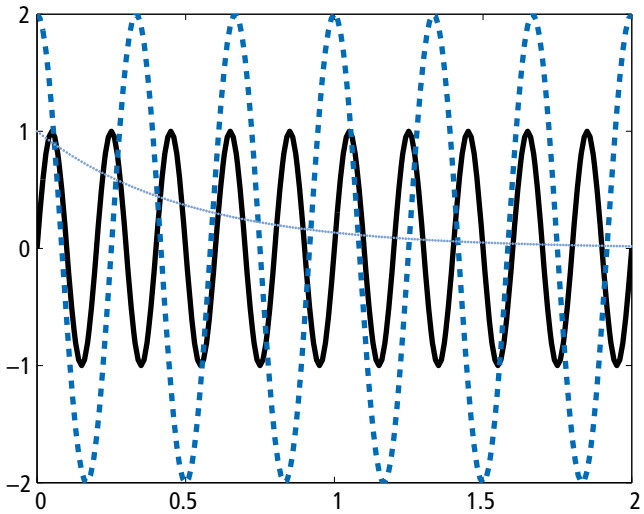

Abbildung 1.7: x, y-Grafik mit mehreren Funktionen und verschiedenen Linienfarben und -stilen

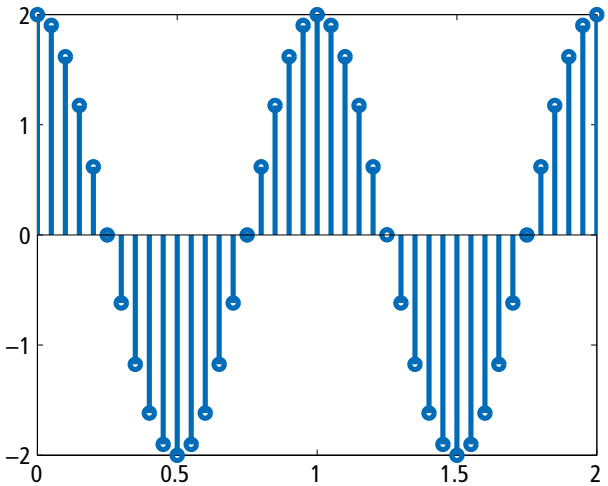

Abbildung 1.8: x, y-Grafik mit der Plotfunktion `stem`

häufiger verwendet wird etwa die Funktion `stem`, welche diskrete Signale in Form einer Lattenzaungrafik darstellt. Diese Funktion eignet sich aber nur für geringe Datenmengen, da sonst die Linien zu nahe aneinander rücken und die die Linien abschließenden kleinen Kreise unschön überlappen. In solchen Fällen ist dann `plot` wieder übersichtlicher. Abbildung 1.8 zeigt das Ergebnis der folgenden MATLAB-Sequenz:

```
» t=(0:0.05:2);
» cosfkt=2*cos(2*pi*t);
```

```
» stem(t,cosfkt)
» box
```

Eine Funktion, die speziell zur Darstellung zeitdiskreter Signale geeignet ist, ist die Funktion `stairs`, welche diese Signale in Form einer Treppenfunktion darstellt. Abbildung 1.9 zeigt für das oben definierte Cosinussignal das Ergebnis des Aufrufs:

```
» stairs(t,cosfkt)
```

Für die Dokumentation solcher Grafiken wesentlich ist auch eine geeignete Beschriftung der Achsen und eine Titelzeile. Außerdem ist es manchmal nützlich, die Grafik mit einem Gitternetz zu versehen, um die Werte besser vergleichen zu können sowie Ausschnitte aus einer Grafik zu vergrößern.

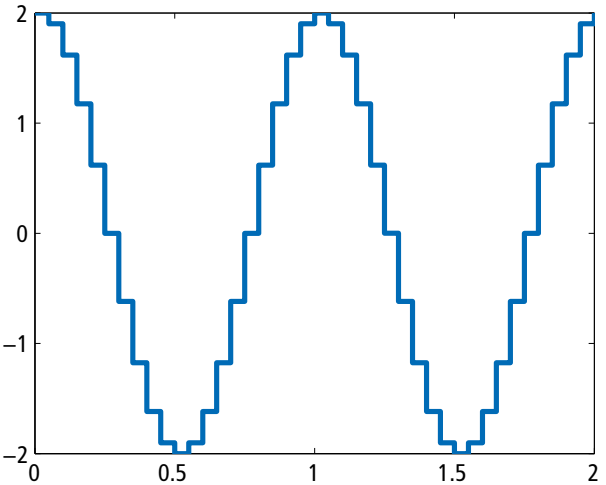

Abbildung 1.9: x, y-Grafik mit der Plotfunktion `stairs`

Auch hierfür bietet MATLAB natürlich Funktionen an. Für die Achsenbeschriftung sind dies `xlabel`, `ylabel` und `title`, für das Gitternetz die Funktion `grid` und für die Vergrößerung die Funktion `zoom`, welche eine Ausschnittsvergrößerung mit Hilfe der Maus gestattet sowie das Kommando `axis`, bei dem die x, y-Bereiche explizit angegeben werden müssen. Beschriftungen innerhalb der Grafik können mit der Funktion `text` vorgenommen werden.

Die Grafiken 1.10 und 1.11 zeigen das Ergebnis folgender MATLAB-Befehlssequenz, in welcher diese Möglichkeiten genutzt werden:

```
» t=(0:0.05:2);
» cosfkt=2*cos(2*pi*t);
» plot(t,cosfkt)
» xlabel('Zeit / s')
» ylabel('Amplitude / V')
» text (0.75,0,'\leftarrow Nulldurchgang','FontSize',18)
» title('Eine Cosinus-Spannung der Frequenz 1 Hz')
```

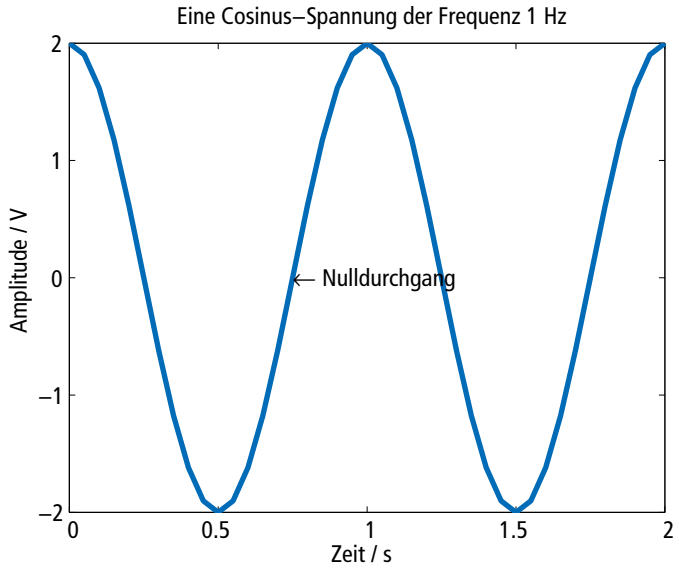

Abbildung 1.10: Beschriftete x, y-Grafik mit der Plotfunktion

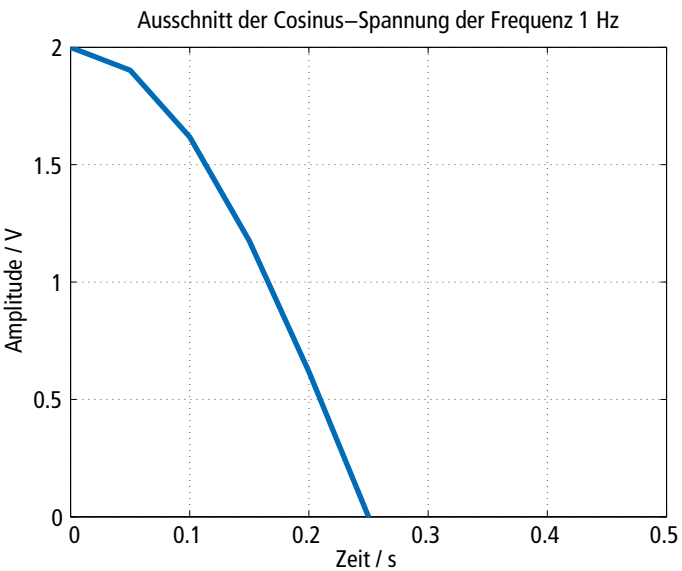

Abbildung 1.11: Ausschnitt aus Abbildung 1.10 mit dem `axis`-Kommando

```
» figure                % Neues Fenster öffnen !!!!
» plot(t,cosfkt)
» xlabel('Zeit / s')
» ylabel('Amplitude / V')
» grid                  % Gitterraster setzen
» axis([0, 0.5, 0, 2]) % Ausschnitt im Intervall [0, 0.5], aber
                        % nur Amplituden im Intervall [0, 2]
» title('Ausschnitt der Cosinus-Spannung der Frequenz 1 Hz')
```

Auf die weiteren zweidimensionalen Plotfunktionen soll an dieser Stelle nicht eingegangen werden. Der Leser möge sich im Helpmenü oder in [20] selbst über die Möglichkeiten informieren und damit experimentieren.

Im Folgenden wollen wir aber noch kurz auf die Möglichkeiten der Nachbearbeitung von Grafiken eingehen, die speziell im interaktiven Betrieb zur Verfügung stehen. Ferner sollen 3D-Grafiken besprochen werden.

Grafiknachbearbeitung

MATLAB 7 zeichnet sich gegenüber seinen Vorgängerversionen durch weitreichende Verbesserungen der Benutzeroberfläche aus (vgl. Abschnitt 1.4). Für die Nachbearbeitung und den Export von Grafiken gibt es daher im interaktiven Betrieb bequeme Möglichkeiten. Da dies für die Dokumentation von MATLAB-Berechnungen von großer Bedeutung ist, soll an dieser Stelle kurz darauf eingegangen werden.

Das Plotfenster einer MATLAB-Grafik stellt sich dem Benutzer in der in Abbildung 1.12 dargestellten Form dar.

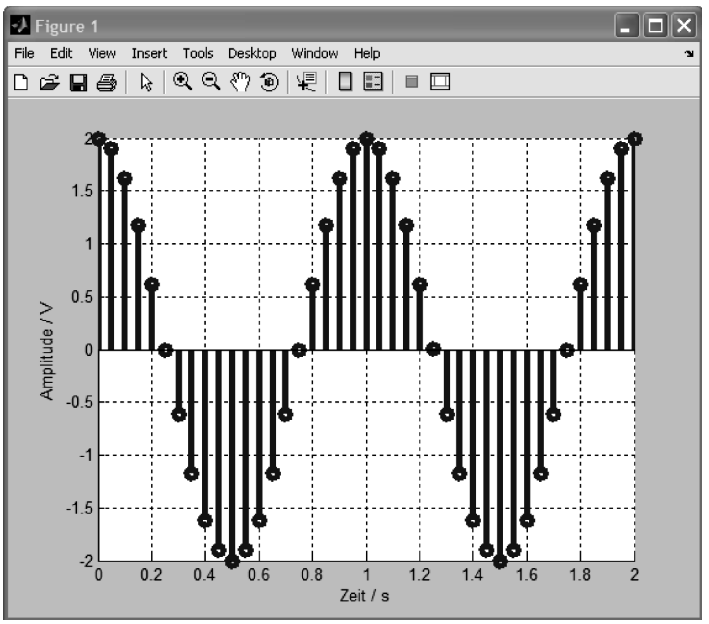

Abbildung 1.12: MATLAB-Plotfenster mit Beispielgraph

Über die unterhalb des Pull-Down-Menüs erkennbare so genannte *Toolbar* können die wesentlichen Funktionen zur Manipulation der Grafik angewählt werden. Die Funktionen sind natürlich auch über das Pull-Down-Menü selbst erreichbar. Neben den Windows-üblichen Icons für Speichern, Drucken etc. im linken Drittel der Toolbar erkennt man im mittleren Drittel Lupensymbole, mit denen die Grafik vergrößert oder verkleinert werden kann, sowie ein Symbol für die Rotation der Grafik. Letzteres ist insbesondere bei der Betrachtung von dreidimensionalen Grafiken (vgl. den nachfolgenden Unterabschnitt) sehr praktisch und von großem Nutzen. Die Icons im rechten Drittel gewähren Zugriff auf weitere nützliche Funktionen, beispielsweise der so genannte *Data Cursor*, mit dem man durch Anklicken der Punkte eines Graphen deren (x, y)-Koordinaten angezeigt bekommt. Oder der Button, mit dem man den Graphen interaktiv mit einer Legende versehen kann. Das Icon `Show Plot Tools` ganz rechts erweitert das Plotfenster um weitere Fenster mit Werkzeugen zur Grafiknachbearbeitung.

Eine detaillierte Darstellung der Möglichkeiten an dieser Stelle würde den Rahmen dieser Einführung sprengen. Wir wollen die Funktionalität stattdessen mit einem kleinen Beispiel illustrieren.

Möchte man z. B. die Titelbeschriftung ändern, so genügt es nach Anwahl des Plotteditierungspfeils in den Titeltext zu klicken und anschließend den Text zu verändern. Entsprechend kann durch einen Doppelklick auf die Linien der Stil der Linien nachträglich verändert werden. Ein Doppelklick auf die Achsen ermöglicht die Veränderung der Achseneinteilung und weiterer Achseneigenschaften. Durch die Mausclicks wird das Plotfenster jeweils um ein Konfigurationsfenster erweitert, in welchem die gewünschten Einstellungen angewählt werden können. Alle Konfigurationsmöglichkeiten auf einen Blick erhält man, wenn man, wie oben bereits erwähnt, das Icon `Show Plot Tools` drückt. Das so erweiterte Plotfenster ist in Abbildung 1.13 wiedergegeben.

Man erhält eine Auswahl von verschiedenen Werkzeugen zur Plotmanipulation. Diese ist im Übrigen kontextabhängig. So öffnet sich das untere Fenster `Property Editor - Axes`, wenn man im Plot das Axenkreuz anklickt. Klickt man den Funktionsgraph an, so wird dieses Fenster durch das entsprechenden `Property Editor`-Fenster ersetzt.

Daneben sind über die Menüeinträge `Insert` und `Tools` noch weitaus mehr Funktionen zu erreichen, die es ermöglichen, das Aussehen der Grafik nachträglich zu beeinflussen. Hier sei an die Experimentierfreudigkeit des Lesers appelliert.

Betrachtet man die zahllosen Möglichkeiten, die das Plotfenster unter MATLAB 7 im interaktiven Betrieb zur Veränderung der Grafiken bietet, so scheint die vorangegangene Diskussion der Plot-Kommandos zu diesem Zweck etwas fragwürdig. In der Tat ist die Bearbeitung der Plots mit den Werkzeugen des Plotfensters einfacher und intuitiver. Zudem müssen keine Kommandos inklusive ihrer Syntax erlernt werden. Es wurde jedoch schon einmal darauf hingewiesen, dass MATLAB eine Programmiersprache ist (vgl. Abschnitt 1.6). Will man aber innerhalb von Programmen Grafiken fertig aufbereiten, so bieten die vorgestellten MATLAB-Kommandos hierfür die einzige Möglichkeit!

Zum Abschluss soll an dieser Stelle noch auf die für die Dokumentation von grafischen Ergebnissen wichtigen Funktionalitäten für den Grafik-Export von MATLAB 7 hingewiesen werden. Im Plotfenster können unter dem Menübefehl `File - Save As ...` verschiedene Formate, wie etwa *Encapsulated Postscript*, *TIFF*, *Portable Network*

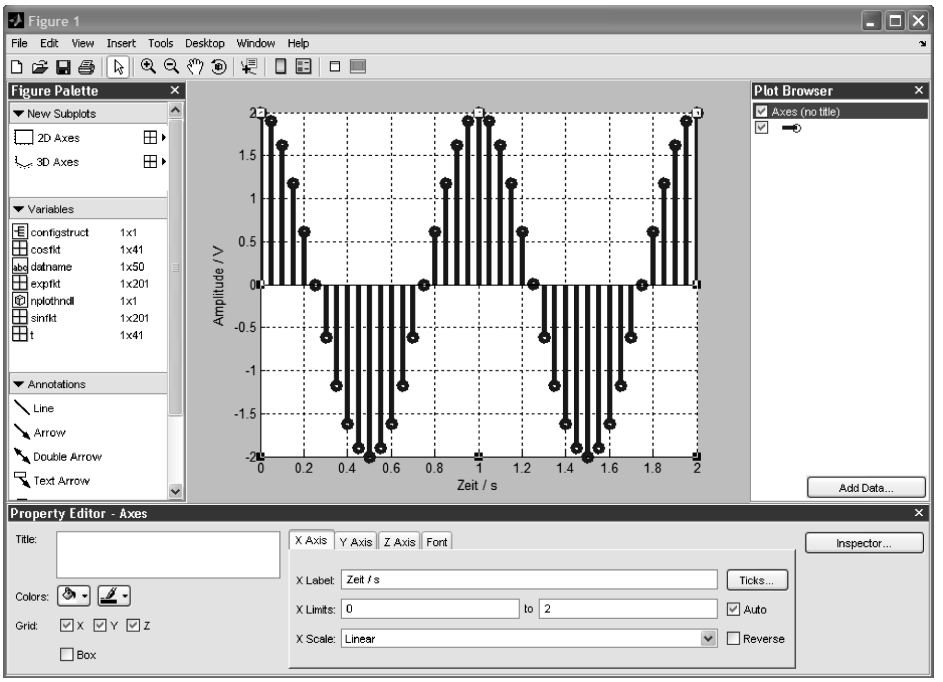

Abbildung 1.13: Plot-Tools-Fenster

Graphics etc., ausgewählt werden, in die eine Grafik konvertiert und gespeichert werden kann. Natürlich sind die Grafiken über `Edit - Copy Figure` weiterhin über die Windows-Zwischenablage in andere Anwendungen exportierbar.

Dreidimensionale Plots

Auch hier wollen und können wir nicht auf alle Möglichkeiten von MATLAB explizit eingehen. Die wichtigsten dreidimensionalen Grafikfunktionen sind wohl `mesh` und `surf` zur Darstellung eines dreidimensionalen Maschen- bzw. Kachelplots und `contour` zur Darstellung eines Höhenlinienplots. Im folgenden Beispiel wird die zweidimensionale Funktion

$$f(x,y) = \sin(x^2+y^2)\mathrm{e}^{-0.2\cdot(x^2+y^2)}$$

mit Hilfe der Funktion `mesh` bzw. `surf` auf dem Quadrat

$$[-3,3]\times[-3,3]\,,$$

welches mit einem Quadrategitter der Kantenlänge 0.1 versehen ist, dargestellt. Berechnet wird dabei der Wert der Funktion an jedem Gitterpunkt. Die Funktionswerte werden mit einer Kachel verbunden und der ganze Graph perspektivisch von einem von MATLAB voreingestellten Blickpunkt (den man allerdings mit den im vorangegangenen Abschnitt beschriebenen Möglichkeiten leicht ändern kann) dargestellt.

Die folgende Befehlsfolge liefert die in Abbildung 1.14 bzw. 1.15 dargestellten Plots:

```
» x=(-3:0.1:3);              % Gitterraster in x-Richtung
» y=(-3:0.1:3)';             % Gitterraster in y-Richtung
» v=ones(length(x),1);       % Hilfsvektor
» X=v*x;                     % Gittermatrix der x-Werte
» Y=y*v';                    % Gittermatrix der y-Werte
»                            % Funktionswerte
» f=sin(X.^2+Y.^2).*exp(-0.2*(X.^2+Y.^2));
» mesh(x,y,f)                % Maschenplot mit mesh
» mxf = max(max(f));         % Maximum der Funktionswerte
» mif = min(min(f));         % Minimum der Funktionswerte
» axis([-3,3,-3,3,mif,mxf])% Achsen anpassen
» xlabel('x-Achse');         % Achsen beschriften
» ylabel('y-Achse');
» figure                     % neuer Plot
» surf(x,y,f)                % Kachelplot mit surf
» axis([-3,3,-3,3,mif,mxf])% Achsen anpassen
» xlabel('x-Achse');         % Achsen beschriften
» ylabel('y-Achse');
```

Interessant ist dabei die Technik zur Erzeugung des x, y-Gitters. Hier wird auf elegante Weise von der Matrixalgebra Gebrauch gemacht, was einem das Schreiben einer Doppelschleife erspart, wie dies etwa in der Programmiersprache C++ geschehen müsste.

Der Leser sollte sich unbedingt die Übung 28, Seite 55, zu Gemüte führen, um zu begreifen, was dabei eigentlich passiert und worin der Trick besteht.

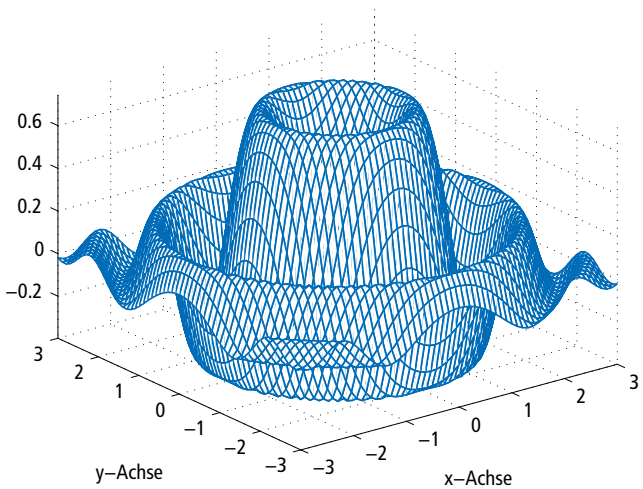

Abbildung 1.14: 3D-Plot mit dem `mesh`-Kommando

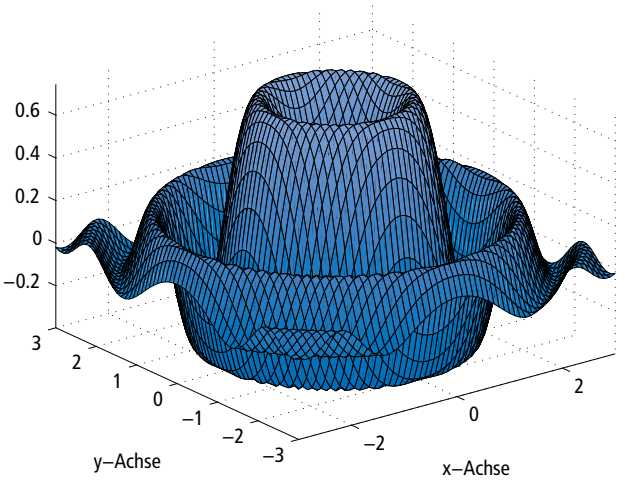

Abbildung 1.15: 3D-Plot mit dem `surf`-Kommando

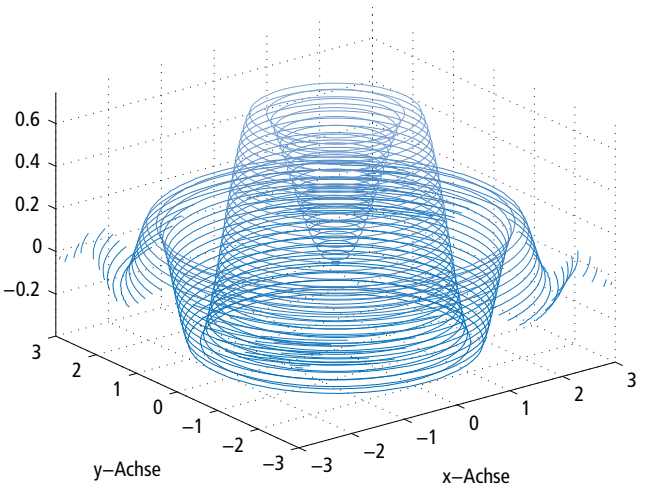

Abbildung 1.16: 3D-Plot mit dem `contour3`-Kommando

Zum Vergleich mit dem Plotergebnis von `mesh` und `surf` sei in der Grafik 1.16 ein 3D-Höhenlinienplot der Funktion dargestellt. Er wurde mit den MATLAB-Kommandos

```
» contour3(x,y,f,30)
» axis([-3,3,-3,3,mif,mxf])% Achsen anpassen
» xlabel('x-Achse');       % Achsen beschriften
» ylabel('y-Achse');
```

erzeugt. Die Zahl 30 gibt dabei die Zahl der darzustellenden Höhenlinien an.

Subplots und Mehrfachplots

MATLAB bietet neben der Darstellung von Plots in verschiedenen Einzelfenstern auch die Möglichkeit, mehrere Plots in *einem* Fenster darzustellen. Dies lässt sich etwa mit dem `subplot`-Kommando bewerkstelligen.

Das folgende Beispiel zeigt, wie die komplexe Funktion $f(t) = t^2 e^{jt} j^t$ nach Betrag und Phase geplottet werden kann, wobei die beiden Plots übereinander dargestellt werden:

```
» t=(0:0.1:5);
» f=(t.^2).*exp(j*t).*(j.^t);  % * und ^ sind Feldoperationen!!
» subplot(211)                 % Obere Grafik anlegen
» plot(t,abs(f))
» subplot(212)                 % Untere Grafik anlegen
» plot(t,angle(f))
```

Abbildung 1.17 zeigt das Ergebnis.

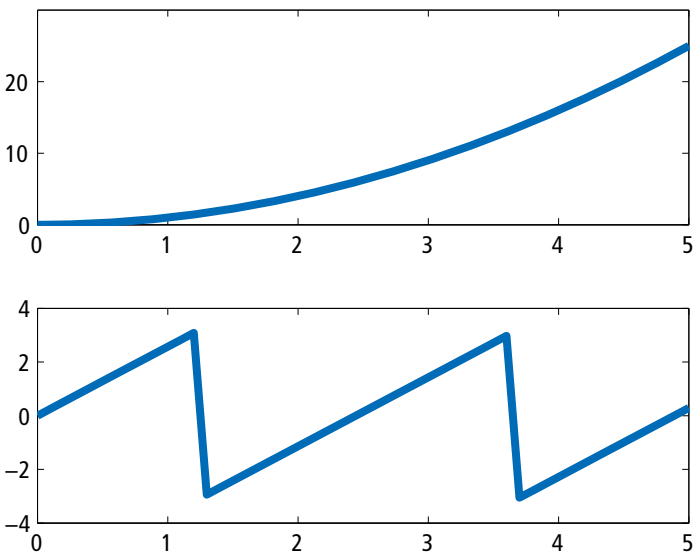

Abbildung 1.17: Plot zweier Grafiken mit dem `subplot`-Kommando

Wie zu sehen ist, legt das Kommando `subplot` lediglich die Achsen der Grafik an der richtigen Stelle an. Den eigentlichen Plot besorgt immer noch der Befehl `plot`.

Der Parameter des `subplot`-Befehls gibt an, wie viele Grafiken in vertikaler Richtung (erste Zahl) und horizontaler Richtung (zweite Zahl) insgesamt geplottet werden sollen. Die dritte Zahl gibt an, welche der Teilgrafiken gemeint ist, gezählt von links oben nach rechts unten. Beispielsweise bedeutet `subplot(325)`, dass die 5. Grafik in einem Array von 3×2 Grafiken (für den nachfolgenden Plotbefehl) gemeint ist.

Eine andere Möglichkeit, mehrere Plots in einem Fenster darzustellen, ist das *Aufeinanderzeichnen* von Grafiken wie in Abbildung 1.6 und 1.7. Für den Plot mehrerer Grafiken *aufeinander* kann neben den auf Seite 44ff beschriebenen Möglichkeiten auch das Kommando `hold` verwendet werden. Es friert die aktuelle Grafik ein, so dass

die nächste Grafik nicht in ein eigenes Fenster, sondern in dasselbe Fenster geplottet wird. Hierzu ein Beispiel:

```
» t=(0:0.5:10);
» sinfkt=sin(2*pi*5*t);
» cosfkt=2*cos(2*pi*3*t);
» plot(t,sinfkt)
» hold
» plot(t,cosfkt)
```

Die aktuelle Grafik ist dabei immer die Grafik, deren Plotfenster „oben“ ist, also dasjenige, mit dem zuletzt gearbeitet wurde. Man sollte daher vor Verwendung des `hold`-Kommandos durch Anklicken des entsprechenden Plotfensters sicherstellen, dass die richtige Grafik überschrieben wird.

Übungen

Bearbeiten Sie die folgenden Aufgaben zur Einübung der Plotfunktionen.

Übung 23 (*Lösung Seite 196*)

Begründen Sie, warum die Sequenz

```
» t=(0:0.01:2);
» sinfkt=sin(2*pi*5*t);
» cosfkt=2*cos(2*pi*3*t);
» expfkt=exp(-2*t);
» plot(t,[sinfkt, cosfkt, expfkt])
```

zu einem MATLAB-Fehler führt.

Übung 24 (*Lösung Seite 197*)

Geben Sie die MATLAB-Befehlsfolge

```
» t=(0:0.5:10);
» sinfkt=sin(2*pi*5*t);
» cosfkt=2*cos(2*pi*3*t);
» expfkt=exp(-2*t);
» plot(t,[sinfkt; cosfkt; expfkt])
```

ein und interpretieren Sie das (etwas seltsame) grafische Ergebnis.

Übung 25 (*Lösung Seite 198*)

Experimentieren Sie mit den Plotfunktionen `plot` und `stem`, wobei Sie unterschiedliche Diskretisierungsschrittweiten und unterschiedliche Liniendarstellungen verwenden sollten.

Verändern Sie nachträglich auch Liniendarstellung und Achseneinteilung mit Hilfe der Editierbefehle des Plotfensters.

Übung 26 (*Lösung Seite 198*)

Experimentieren Sie mit den Plotfunktionen `semilogx`, `semilogy` und `loglog`, um die unterschiedlichen Möglichkeiten der logarithmischen Achsendarstellung zu erlernen.

Verwenden Sie dabei für einen Frequenzvektor

$$\omega = (0.01,\ 0.02, 0.03,\ 0.04, \ldots,\ 5) \quad \text{rad/s}$$

die in der Signalverarbeitung und Regelungstechnik vorkommenden so genannten Übertragungsfunktionen[9] eines Integrators

$$H(j\omega) = \frac{1}{j\omega}$$

und eines Verzögerungsgliedes 1. Ordnung

$$H(j\omega) = \frac{1}{1 + j\omega}\,,$$

wobei Sie jeweils den Absolutbetrag der Funktionen plotten sollten.

Entscheiden Sie, welche Darstellung wohl am geeignetsten ist.

Übung 27 (*Lösung Seite 199*)

Beschriften Sie die Ergebnisplots aus Übung 26 mit Hilfe entsprechender MATLAB-Kommandos und experimentieren Sie mit den Befehlen `axis` und `zoom`.

Führen Sie die gleichen Aufgaben zum Vergleich mit Hilfe der Editierbefehle des Plotfensters durch.

Üben Sie dabei auch die Dokumentation einer MATLAB-Grafik in einem Textverarbeitungsprogramm, indem Sie die Grafik über die Zwischenablage `Edit - Copy Figure`) in das Dokument einfügen (`paste`).

Übung 28 (*Lösung Seite 199*)

Testen Sie die Befehlssequenz zum `surf`-Plot-Beispiel auf Seite 51 mit zwei Vektoren der Form

$$\vec{x} = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix} \quad \text{und} \quad \vec{y} = \begin{pmatrix} -2 & 0 & 2 \end{pmatrix}$$

und machen Sie sich klar, worin der Trick zur Erzeugung des x, y-Gitters besteht.

9 Der Begriff „Übertragungsfunktion“ wird in der einschlägigen Literatur (z. B. [4, 9, 23]) behandelt. Ein Verständnis dieses Begriffs ist aber für die vorliegende Aufgabe nicht wesentlich.

Übung 29 (*Lösung Seite 200*)

Testen Sie einige der übrigen dreidimensionalen Plotfunktionen mit Hilfe des Beispiels auf Seite 51 aus. Einen Überblick über diese Plotfunktionen erhalten Sie durch Eingabe von `help graph3d` in der MATLAB-Kommandooberfläche.

Übung 30 (*Lösung Seite 200*)

Plotten Sie Betrag und Phase der in Übung 26 verwendeten Übertragungsfunktionen jeweils in einer Grafik übereinander (so etwas nennt man dann ein *Bode-Diagramm*). Verwenden Sie für den Betrag auf der y-Achse einen logarithmischen Maßstab.

Übung 31 (*Lösung Seite 201*)

Plotten Sie die Exponentialfunktionen

$$\mathrm{e}^{-t/2} \quad \text{und} \quad \mathrm{e}^{-2t/5}$$

über dem Intervall $[0, 2]$

1. übereinander in einen Plot,
2. nebeneinander in verschiedene Plots,
3. untereinander in verschiedene Plots.

Beschriften Sie einen der Plots in geeigneter Weise und versuchen Sie, diesen Plot dann mit Hilfe der Zwischenablage in einer MS-Word-Datei zu speichern.

Übung 32 (*Lösung Seite 201*)

Berechnen und plotten Sie die Funktion

$$x^2 + y^2$$

auf dem Rechteck $[-2, 2] \times [-1, 1]$.

Verwenden Sie dabei ein Gitternetz, welches in x-Richtung eine äquidistante Unterteilung der Länge 0.2 hat und in y-Richtung eine äquidistante Unterteilung der Länge 0.1.

1.2.6 I/O-Operationen

I/O-Operationen (**I**nput/**O**utput-Operationen) beziehen sich auf den Datenaustausch zwischen MATLAB und externen Dateien. Ein solcher Mechanismus ist sehr nützlich, will man z. B. extern gewonnene Messdaten in MATLAB einlesen oder Ergebnisse von MATLAB-Simulationen in andere Applikationen einbinden.

Die Kommandos `load` und `save`

MATLAB kennt viele Datei-Schnittstellen. Die elementarsten unter ihnen werden durch die Kommandos `load` und `save` repräsentiert.

So kann z. B. mit Kommando

```
» save wssicherung
```

der Inhalt des Workspace (also das, was mit `whos` angezeigt wird) in eine Binärdatei namens `wssicherung.mat` (Endung .mat) gespeichert werden. Zielverzeichnis ist dabei das in dem Menüeintrag `Current Directory` (s. (⑤) in Abbildung 1.1, Seite 14) eingestellte Arbeitsverzeichnis. Diese Einstellung kann selbstverständlich durch die explizite Angabe eines Verzeichnisses vor dem Dateinamen überschrieben werden. Auch ist neben der Sicherung des Workspace die Speicherung einzelner Variablen möglich (s. u.).

In MATLAB 7 ist das Binärformat gegenüber den Vorgängerversionen geändert worden. Dies macht unter Umständen den Austausch solcher Dateien mit anderen MATLAB-Nutzern schwierig. Um diese Abwärtskompatibilität sicherzustellen, kann man beim `save`-Kommando mit Hilfe einer Option (`-V6`) angeben, ob man die Datei im früheren Binärformat abgespeichert haben will. Die folgende Anweisung sichert die Variablen `var1` und `var2` im MATLAB 6 Binärformat in einer Datei, welche in einem (explizit angegebenen) Verzeichnis außerhalb des `Current Directory` angelegt werden soll:

```
» save 'C:\beucher\matlab7\dieVars' var1 var2 -V6
```

Will man Workspace-Variablen grundsätzlich im MATLAB 6 Binärformat abspeichern (was jedoch im Hinblick auf die Weiterentwicklung von MATLAB nicht zu empfehlen ist), so kann man dies im Menü des MATLAB-Desktop unter `File - Preferences ...` in der Karte `General - MAT-Files` einstellen.

Mit `load <pfad/dateiname>` oder `load <dateiname>` können die so gespeicherten Variablen wieder in den Workspace eingelesen werden.

Durch Angabe zusätzlicher Optionen, die in der entsprechenden Hilfe zu den Kommandos beschrieben sind, kann man auch das Dateiformat, in dem die Variablen gespeichert werden, verändern. Ein wichtiger Fall ist hierbei das ASCII-Format (druckbare Zeichen).

So könnte etwa mit

```
» save DieVarX.txt X -ASCII
```

eine Variable `X` des Workspace im ASCII-Format in einer Datei `DieVarX.txt` gesichert werden.

Mit `load` kann dann diese oder auch andere externe ASCII-Dateien, z. B. mit anderen Programmen aufgenommene Messreihen, (wieder) eingelesen werden. Allerdings müssen hierfür die Daten in einem *Matrixformat* aufbereitet sein, damit MATLAB sie einer Matrix (deren Name dann automatisch identisch mit dem Dateinamen ohne Endung ist) im Workspace zuordnen kann. Beispielsweise kann die oben mit dem `save`-Kommando gesicherte Variable `X` mit

```
» load -ASCII DieVarX.txt
```

wieder in den Workspace geladen werden und steht dann dort unter dem Variablennamen `DieVarX` (nicht also `X`) wieder zur Verfügung.

Alternativ zu `save` und `load` können im *interaktiven* Betrieb auch die Menübefehle `File - Save Workspace As ...` und `File - Import Data` verwendet werden. Im

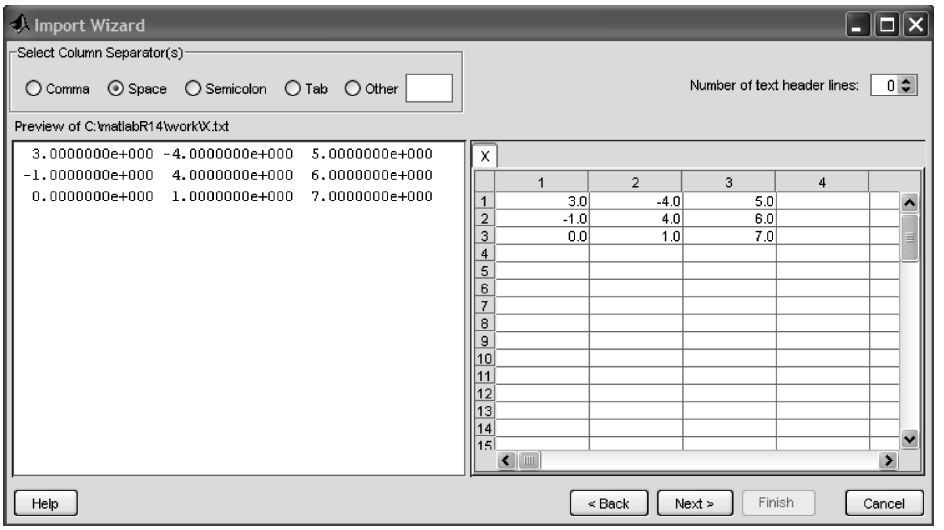

Abbildung 1.18: Der Import Wizard

letzteren Fall öffnet sich der so genannte *Import Wizard*, welcher flexible Möglichkeiten des Datenimports bietet.

Der Import Wizard

Die Kommandos `save` und `load` sind vor allen Dingen dann unverzichtbar, wenn Lade- und Speicheroperationen *in Programmen* (vgl. Abschnitt 1.6) durchgeführt werden sollen. Nutzt man MATLAB 7 im interaktiven Betrieb, also von der Kommandooberfläche aus, so bietet sich zum Laden von Daten der Import Wizard als Alternative an. Durch Anwählen des Menübefehls `File - Import Data` öffnet sich zunächst ein Dateiauswahlfenster, bei dem das Arbeitsverzeichnis und verschiedene Dateiformate zur Auswahl angeboten werden. Klickt man eine Datei an, so öffnet sich das in Abbildung 1.18 dargestellte Dialogfeld:

Hier wird eine Vorschau der zu ladenden Daten angeboten und es können noch einige zusätzliche Einstellungen vorgenommen werden, bevor die Matrix mit dem Auswahlbutton `Finish` im Workspace gespeichert werden kann. In der Kommandoebene wird die Speicherung mit einer entsprechenden Meldung quittiert.

Spezielle I/O-Funktionen

Zu den I/O-Operationen gehören, neben den oben besprochenen, noch zahlreiche Funktionen zum Schreiben und Lesen in und von Dateien, welche zum Teil eine ähnliche Syntax wie entsprechende C/C++-Funktionen haben.

Von speziellem Interesse sind darüber hinaus auch Funktionen zum Einlesen von Bild- und Audioformaten oder zu Formaten anderer Anwendungen, wie etwa Microsoft Excel.

Einen vollständigen Überblick über die Funktionen liefert die Eingabe von `help iofun`, `help audiovideo` und `help imagesci`. Die Kommandos listen eine ganze Reihe von MATLAB-Funktionen auf, mit denen der Import und der Export verschiedener

Dateiformate bewerkstelligt werden kann. Die Diskussion all dieser, zum Teil sehr speziellen, Funktionen würde jedoch viel zu weit führen.

Exemplarisch greifen wir daher zwei Funktionen heraus, um den Umgang mit diesen Dateischnittstellen einzuüben. Wir betrachten im Folgenden die Funktionen `xlsread` und `xlswrite` zur Kommunikation mit MS Excel etwas genauer.

Die Datei **ExcelDatBsp.xls**, die wir bereits in Übung 7 im Zusammenhang mit dem Array-Editor verwendet haben, enthält ein Excel-Sheet („Tabelle 1“) mit einem numerischen Feld (Matrix). Dies ist die einfachste Form einer Excel-Datei, die von `xlsread` verarbeitet werden kann. Befindet sich die Datei in einem Zugriffspfad von MATLAB (im einfachsten Fall im `Current Directory`), so kann diese Matrix mit der Anweisung

```
» ExcelMat = xlsread('ExcelDatBsp')

ExcelMat =

     3     1     2
     4     0    -5
     3     1     2
     4     0    -5
     3     1     2
     ...
```

in den MATLAB-Workspace eingelesen werden und steht dort unter dem angegebenen Variablennamen (`ExcelMat`) zur Verfügung.

Mit den nachfolgenden Anweisungen kann eine Matrix in eine Excel-Datei übertragen werden:

```
» AusSpalte1 = ExcelMat(1:5,1)

AusSpalte1 =

     3
     4
     3
     4
     3

» xlswrite('AuszugExcelDatBsp', AusSpalte1, 'Teil 1')
Warning: Added specified worksheet.
> In xlswrite>activate_sheet at 259
  In xlswrite at 213
```

Der Vektor `AusSpalte1` wird in die Excel-Datei **AuszugExcelDatBsp.xls** geschrieben. Dabei wird, wie gewünscht, ein neues Excel-Sheet mit dem Namen („Teil 1“) angelegt. Eine entsprechende Warnung, dass dieses Sheet neu angelegt wurde, wird zusätzlich ausgegeben.

Auf diese Weise können numerische Werte zwischen MATLAB und Excel ausgetauscht werden. Enthält das Excel-Sheet andere Informationen (etwa Text), so wird der Wert `NaN` (Not a Number) zurückgeliefert, es sei denn, es handelt sich um Überschriftstext. Dieser wird ignoriert.

Die Funktionen verfügen noch über zahlreiche andere Optionen. Z. B. können die zu importierenden Daten im Excel-File interaktiv ausgewählt werden. Die Erforschung dieser Details soll dem Leser überlassen bleiben.

In den Übungen werden weitere Ein-Ausgabe-Schnittstellen von MATLAB diskutiert. Speziell wollen wir auf Übung 37 hinweisen. In dieser Übung wird der Umgang mit Audiodateien durchgespielt.

Der MATLAB-Suchpfad

Im Zusammenhang mit Dateioperationen erscheint es angebracht zu erläutern, wie MATLAB Kommandos, Programme und Dateien sucht und findet.

Bislang haben wir nur vom aktuellen Verzeichnis Gebrauch gemacht. Alle dort gespeicherten Programme und Daten sind MATLAB automatisch bekannt. Darüber hinaus muss MATLAB jedoch auch auf alle Kommando- und Funktionsdefinitionen zugreifen können, die zum Sprachumfang des MATLAB-Kerns und der Toolboxen gehören. Die zugehörigen Verzeichnisse sind im MATLAB-Suchpfad gespeichert, der beim Start eingelesen wird.

Mit dem Kommando `path` können die aktuellen Zugriffspfade von MATLAB angezeigt und auch gesetzt werden.

Durch die Anweisung

```
» path

        MATLABPATH

    C:\matlabR14\toolbox\matlab\ops
    C:\matlabR14\toolbox\matlab\lang
    C:\matlabR14\toolbox\matlab\elmat

    ...

    C:\matlabR14\work
    C:\beucher\Matlab7
    C:\beucher\Buch\Matlab\Begleitsoftware

    ...
```

werden die aktuellen Suchpfade angezeigt.

Im obigen (verkürzt dargestellten) Beispiel sind Verzeichnisse auf die Funktionsgruppen für die schon erwähnten MATLAB-Operatoren (`ops`) oder die elementaren mathematischen Funktionen `elmat` angegeben.

Die Verzeichnisse `C:\beucher\...` sind solche, die der Autor selbst dem Suchpfad hinzugefügt hat.

Will man einen Pfad, etwa den des Verzeichnisses `C:\beucher\Matlab7`, einbinden, so kann dies sehr schnell mit folgendem Befehl erfolgen:

```
» path(path, 'C:\beucher\Matlab7');
```

Natürlich kann ein solches Verzeichnis sehr einfach auch wieder mit Hilfe eines Menübefehls hinzugefügt (und wieder entfernt) werden, im vorliegenden Fall mit `File - Set Path ...`.

MATLAB findet automatisch alle m-Files (Kommandos, eigene MATLAB-Programme, Simulink-Systeme) in den mit `path` angezeigten Verzeichnissen.

Schreibt man eigene Programme (vgl. Abschnitt 1.6), so empfiehlt sich das Anlegen des eigenen Arbeitsverzeichnisses im Suchpfad. Programme sollten aber nicht in irgendeinem Toolbox-Verzeichnis abgelegt werden. Beim nächsten automatischen Update dieser Toolbox sind diese dann nämlich verloren, sofern sie nicht anderweitig gesichert wurden!

Übungen

Bearbeiten Sie die folgenden Aufgaben zum Ein- und Auslesen externer Dateien.

Übung 33 (*Lösung Seite 202*)

Erzeugen Sie mit einem Editor einen Vektor und/oder eine Matrix von reellen Zahlen und speichern Sie diese unter einem Dateinamen ab.

Löschen Sie alle Matrizen des Workspace mit `clear`.

Versuchen Sie anschließend mit dem `load`-Kommando die Dateiinhalte einzulesen und analysieren Sie den Inhalt des Workspace.

Vergleichen Sie hierzu auch unbedingt die nachfolgende Übung 34!

Übung 34 (*Lösung Seite 202*)

Erzeugen Sie mit einem normalen Editor eine ASCII-Datei mit einem Spaltenvektor aus komplexen Zahlen. Lesen Sie diese von der MATLAB-Kommandoebene aus in den MATLAB-Workspace ein. Was stellen Sie fest?

Übung 35 (*Lösung Seite 202*)

Erzeugen Sie mit MATLAB einen Spaltenvektor aus komplexen Zahlen. Speichern Sie diesen Vektor mit `save` im binären Format.

Löschen Sie den Workspace anschließend mit dem Kommando `clear` und lesen Sie dann die Datei mit `load` oder mit Hilfe des Import Wizard wieder in den MATLAB-Workspace ein!

Vergleichen Sie das Ergebnis mit dem aus Aufgabe 34.

Übung 36 (*Lösung Seite 203*)

Definieren Sie im MATLAB-Workspace die komplexe Matrix

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 1+j & 6 & 0 \\ 3+j & 0 & -j \end{pmatrix}.$$

Speichern Sie anschließend mit Hilfe der Funktion `dlmwrite` diese Matrix als *Spaltenvektor* im ASCII-Format in einer Datei **MMatrix.txt** ab.

Konsultieren Sie bezüglich der Verwendung der Funktion die MATLAB-Hilfe (`help dlmwrite`).

Übung 37 (*Lösung Seite 204*)

Lesen Sie mit Hilfe einer geeigneten MATLAB-Funktion eine der unter `C:\WINDOWS\""MEDIA` zu findenden *.wav-Dateien ein und stellen Sie das entsprechende Audiosignal grafisch dar. Achten Sie dabei darauf, dass die Zeitachse mit den richtigen Zeiten beschriftet ist.

Multiplizieren Sie anschließend das Signal mit dem Faktor 10 und speichern Sie es im *.wav-Format unter einem anderen Namen wieder ab.

Wie hören sich die Audiosignale im Vergleich an?

Übung 38 (*Lösung Seite 204*)

Lösen Sie die Aufgabe aus Übung 37 erneut mit dem „Microsoft-Sound“ (Datei `Windows XP-Startvorgang.wav`) und dem Faktor 0.

Übung 39 (*Lösung Seite 204*)

Erstellen Sie einen Ordner `C:\mymatlab` und richten Sie diesen Ordner so ein, dass Sie ihn als Verzeichnis für MATLAB verwenden können.

1.2.7 Elementare Matrixmanipulationen

Zu den am häufigsten verwendeten MATLAB-Befehlen gehören einige so genannte elementare Matrixmanipulationen [21, 28, 8], über die der Befehl `help elmat` einen vollständigen Überblick liefert.

Ein paar dieser Befehle, wie etwa die Kommandos `zeros` und `ones` zum Erzeugen einer Matrix aus Nullen oder Einsen, haben wir bereits in den vorangegangenen Abschnitten verwendet. Diese sind vor allem für die Initialisierung von Vektoren oder Matrizen sehr nützlich.

Nachfolgend wird eine 2×2-Matrix mit Nullen, eine 3×2-Matrix mit Einsen und ein Vektor der Länge eines vorher definierten Vektors $\vec{x}_1$ mit Nullen initialisiert.

```
» M = zeros(2,2)

M =

     0     0
     0     0

» N = ones(3,2)

N =

     1     1
     1     1
     1     1
```

```
» x1 = [1,2,3,4,5,6];
» v=zeros(length(x1),1)

v =

     0
     0
     0
     0
     0
     0
```

Nützlich ist oft auch der Befehl `eye` zur Erzeugung einer Einheitsmatrix. Hier ein Beispiel:

```
» E5 = eye(5)

E5 =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Von großer Bedeutung, vor allem in Programmen, sind auch die automatische Feststellung der Länge eines Vektors und der Größe einer Matrix, die mit den Kommandos `length` und `size` errechnet werden können. Nachfolgend wird die Länge des eben definierten Vektors $\vec{x}_1$ ermittelt:

```
» length(x1)

ans =

     6
```

Die folgende Befehlssequenz eliminiert die 2. Spalte aus der oben erzeugten Einheitsmatrix E5 und bestimmt anschließend die Größe der resultierenden Matrix:

```
» B = E5;              % Umspeichern der Matrix E5
» B(:,2) = [ ]         % Leeren der 2. Spalte

B =

     1     0     0     0
     0     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

```
» [zeilen, spalten] = size(B)   % Größe feststellen

zeilen =

     5

spalten =

     4
```

Zu den nach `help elmat` von MATLAB aufgelisteten Befehlen gehören des Weiteren Kommandos zur Erzeugung spezieller Matrizen und spezieller Variablen und Konstanten. Wir überlassen es dem Leser, diese Kommandos zu erforschen, wobei wir allerdings nicht versäumen wollen, auf das Kommando `why` gesondert hinzuweisen.

Im Zusammenhang mit elementaren Matrixoperationen sollte auch der Operator ′ zur *Transponierung* einer Matrix oder eines Vektors erwähnt werden, obwohl dieser Operator nicht in der entsprechenden Rubrik aufgelistet ist.

Die Transponierung einer Matrix wird erreicht, indem das ′-Symbol der zu transponierenden Matrix nachgestellt wird.

Beispielsweise liefert die folgende Befehlssequenz identische Spaltenvektoren $\vec{x}_1$ und $\vec{x}_2$

```
» x1 = [1; 2; 3; -1; 4; 5]    % Spaltenvektordefinition
» x2 = [1  2  3  -1  4  5]'   % Zeilenvektor transponiert
```

und die Befehlsfolge

```
» M = [1 2; 3 -2; -1 4]       % eine 3x2-Matrix
» N = M'
```

erzeugt aus der 3×2-Matrix

$$M = \begin{pmatrix} 1 & 2 \\ 3 & -2 \\ -1 & 4 \end{pmatrix}$$

die dazu transponierte Matrix

$$N = \begin{pmatrix} 1 & 3 & -1 \\ 2 & -2 & 4 \end{pmatrix}.$$

Bei komplexen Matrixeinträgen werden die Matrizen durch den Operator ′, entsprechend den mathematischen Regeln, geklappt und konjugiert:

```
» M = [i 2; 3 -j]       % eine 2x2-Matrix mit komplexen Einträgen

M =
```

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

» N = M'                  % die transponierte Matrix
N =

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

Aus

$$M = \begin{pmatrix} j & 2 \\ 3 & -j \end{pmatrix}$$

ist also

$$N = \begin{pmatrix} -j & 3 \\ 2 & j \end{pmatrix}$$

entstanden.

Will man die Konjugierung der Einträge vermeiden, so ist die Transponierungsoperation wieder als Feldoperator zu definieren:

```
» K = M.'

K =

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

Aus

$$M = \begin{pmatrix} j & 2 \\ 3 & -j \end{pmatrix}$$

ist in diesem Fall, wie gewünscht,

$$K = \begin{pmatrix} j & 3 \\ 2 & -j \end{pmatrix}$$

entstanden.

Den :-Operator haben wir bereits in Abschnitt 1.2.1 im Zusammenhang mit dem Zugriff auf Spalten und Zeilen einer Matrix kennen gelernt. Sehr nützlich ist auch folgende Anwendung dieses Operators, mit der aus einer Matrix ein Spaltenvektor mit den gleichen Einträgen erzeugt werden kann:

```
» M = [1 2; 3 -2; -1 4]       % eine 3x2-Matrix

M =
```

```
     1     2
     3    -2
    -1     4

» mVec = M(:)

mVec =

     1
     3
    -1
     2
    -2
     4
```

Die Spalten der Matrix werden dabei untereinander organisiert.

Kopien und Vervielfältigungen von Matrizeneinträgen können mit der Funktion `repmat` vorgenommen werden. Mit Hilfe der Anweisung

```
» N = repmat(M,2,2)

N =

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

wird offenbar die obige Matrix `M` viermal in Form eines 2×2-Schemas wiederholt und auf die Variable `N` gespeichert.

Zum Abschluss weisen wir noch auf den `end`-Operator hin, da dieser insbesondere in der MATLAB-Programmierung sehr nützlich ist. Dieser Operator berechnet den letzten Index eines Vektors. Mit der folgenden Anweisung etwa kann der letzte Eintrag eines Vektors gelöscht werden, ohne dass man explizit den Index angeben muss:

```
» zVec = [0, 3, -1, 0, 1, 99]

zVec =

     0     3    -1     0     1    99

» zVec(end) = [ ]


zVec =

     0     3    -1     0     1
```

Diese Technik kann insbesondere innerhalb von Programmen (vgl. Abschnitt 1.6) verwendet werden, wenn sich die Größe eines Vektors im Laufe der Programmausführung ändert und damit der letzte Index bei der Programmierung nicht explizit bekannt ist. Auch Verlängerungen eines Vektors sind auf diese Weise durchführbar, wie folgende Anweisung zeigt:

```
» yVec = [0, 3, -1, 0, 1, 99]

yVec =

     0     3    -1     0     1    99

» yVec(end+1:end+3) = [-1 -2 -3]

yVec =

     0     3    -1     0     1    99    -1    -2    -3

» yVec(end+1) = 100

yVec =

     0     3    -1     0     1    99    -1    -2    -3   100
```

Übungen

Bearbeiten Sie die folgenden Aufgaben zu elementaren Matrixmanipulationen. Konsultieren Sie ggf. mit `help elmat` die MATLAB-Hilfe, um eine geeignete Funktion zu finden.

Übung 40 (*Lösung Seite 205*)

Definieren Sie den Vektor

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

unter MATLAB als Spalten- und Zeilenvektor.

Übung 41 (*Lösung Seite 205*)

Bearbeiten Sie Übung 2, Seite 22, Teil 1, nochmals unter Verwendung der Funktion `repmat`.

Übung 42 (*Lösung Seite 205*)

Erzeugen Sie mit Hilfe einer geeigneten elementaren Matrixmanipulation einen Vektor von zehn Punkten zwischen 0 und 1, welche bezüglich einer *(zehner-)logarithmischen* Skala äquidistant sind. Dies bedeutet, dass die *Zehnerlogarithmen* der erzeugten Zahlen in gleichem Abstand aufeinander folgen sollen.