Computereinsatz in der Mathematik

Eberhard Luik

SS 2019

Inhaltsverzeichnis

1 Einführung in Latex

1.1 Einleitung

1.1.1 Was ist Latex

Latex (auch mit der Schrift LaTeX dargestellt) ist ein äußerst flexibles, umfangreiches Programmpaket zur Erstellung von Dokumenten in Buchdruckqualität und bestens geeignet für mathematisch-naturwissenschaftliche Texte. Es ist für alle wichtigen Betriebssysteme (Windows, Mac, Linux) frei verfügbar. Für mathematische Texte ist Latex inzwischen internationaler Standard.

1.1.2 Die Entstehung von Latex

Mit der Verbreitung der Computer entstand der Wunsch, damit auch Texte mit komplizierten (mathematischen) Formeln zu verarbeiten. Zu diesem Zweck entwickelte Donald E. Knuth Ende der siebziger Jahre das Textsatzsystem **Tex** (Tau Epsilon Chi). Die Anwendung dieses Systems ist aber kompliziert und erfordert Kenntnisse im Buchdruck, um dem Text ein ansprechendes Erscheinungsbild zu verleihen.

Vor diesem Hintergrund entwickelte Laslie Lamport Anfang 1980 das Paket **Latex**, das dem Anwender diese drucktechnischen Details abnimmt. Inzwischen gibt es zu Latex viele Zusatzpakete (sogenannte *usepackages*), z.B. für mathematische Symbole, viele Schriftarten oder um Graphiken bzw. digitale Bilder einzubinden.

1.1.3 Wie funktioniert Latex

Im Gegensatz zu anderen bekannten Textverbeitungssystemen wie z.B. Word arbeitet Latex nicht nach dem **wysiwyg**-Prinzip (**w**hat **y**ou **s**ee **i**s **w**hat **y**ou **g**et). Das bedeutet, dass das gewünschte Erscheinungsbild des Textes bei der Eingabe nicht sofort am Bildschirm angezeigt wird.

Vielmehr müssen Text und Formatierungskommandos mit einem Editor zunächst in eine Datei eingegeben werden. Danach erzeugt das Latex-Formatierungsprogramm eine weitere Datei mit dem gewünschten Layout, welche dann am Bildschirm betrachtet oder auf dem Drucker ausgegeben werden kann. Entsprechend ist bei der Korrektur oder Änderung des Erscheinungsbildes vorzugehen.

Diesbezüglich arbeitet Latex wie ein Compiler bei Programmiersprachen.

1.1.4 Arbeitsschritte in Latex

Aus der oben beschriebenen Funktionsweise ergeben sich die nötigen Arbeitsschritte (unter Linux):

1. **Eingabe in eine Datei**

   Text und Formatierungskommandos sind mit Hilfe eines Editors in eine Datei mit dem Namen <name>.tex einzugeben. Die Datei muss den Zusatz .tex haben; <name> ist frei wählbar.

   Unter Linux kommen vor allem die Editoren **emacs** und **kile** zum Einsatz.

Der Aufruf dieser Editoren lautet:

`kile <name>.tex` (z.B. `kile seite1.tex`)

`emacs <name>.tex` (z.B. `emacs seite1.tex`).

Prinzipiell kann für die Eingabe jedoch jeder Editor verwendet werden, der den Text im ASCII-Format abspeichert.

2. **Formatierung**

   Der Aufruf

   `latex <name>` (z.B. `latex seite1`)

   (ohne den Zusatz `.tex`) erzeugt eine neue Datei mit dem Namen

   `<name>.dvi` (z.B. `seite1.dvi`),

   welche das Erscheinungsbild unseres Textes enthält (in Maschinencode).

3. **Betrachten am Bildschirm**

   Diese neu erzeugte Datei können wir mit dem Kommando

   `kdvi <name>` (z.B. `kdvi seite1`)

   auf dem Bildschirm anschauen. Erst jetzt wird das Erscheinungsbild unseres Textes ersichtlich. Das `kdvi`-Kommando braucht die Datei `<name>.dvi`. Falls diese nicht existiert, so erfolgt eine Fehlermeldung.

   In den neueren Linux-Versionen gibt es auch das Kommando

   `okular <name>.dvi` (z.B. `okular seite1.dvi`)

   Sind noch Änderungen erwünscht, so müssen die Schritte 1 und 2 (in dieser Reihenfolge) wiederholt werden.

4. **Ausgabe auf dem Drucker**

   Mit dem Befehl

   `dvipdf <name>` (z.B. `dvipdf seite1`)

   wird aus der Datei `<name>.dvi` eine pdf-Datei mit dem Namen `<name>.pdf` erzeugt. Dabei steht pdf für *portable document format*. Dies ist das Standard-Format für den Austausch von Texten zwischen verschiedenen Usern bzw. verschiedenen Plattformen (Betriebssystemen).

   pdf-Dateien kann man auf dem Drucker ausgeben, mit dem *Acrobat-Reader* bzw. mit dem Kommando `okular` am Bildschirm betrachten oder per Email verschicken. Damit kann auch ein anderer Benutzer unseren (mit Latex erzeugten) Text lesen, ohne das Programmpaket Latex auf seinem Rechner installieren zu müssen.

Anmerkungen:

1. Neuere Editoren (z.B. Kile oder Emacs) erlauben es, die in 2. - 4. beschriebenen Kommandos aus dem Editor heraus zu starten.
2. Zur Formatierung gibt es auch den Befehl

   `pdflatex <name>` (z.B. `pdflatex seite1`),

   welcher aus der Datei `<name>.tex` sofort die pdf-Datei `<name>.pdf` erzeugt.

1.2 Grundlagen von Latex

1.2.1 Grundstruktur eines Latex-Dokuments

Ein Latex-Dokument besteht aus einem Vorspann (Präambel) und dem eigentlichen Textteil einschließlich Kommandos für das Layout.

In der Präambel werden die Einstellungen für das Dokument vorgenommen, wie zum Beispiel die Dokumentklasse (Buch, Artikel, Brief,...), Textbreite und Texthöhe, Lage der Ränder, Art der Seitennummerierung. Außerdem werden zusätzliche Programmpakete (`usepackages`) geladen, etwa für die deutsche Silbentrennung oder für weitere mathematische Sonderzeichen.

Eine an unsere Bedürfnisse angepasste Präambel sieht wie folgt aus:

```
\documentclass[12pt]{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[ngerman]{babel}
\usepackage{lmodern}
\usepackage{latexsym}
\usepackage{amsfonts}
\usepackage{amssymb}
\usepackage{amsmath}
\pagestyle{empty}
\topmargin -1cm
\textheight 25cm
\textwidth 16.0 cm
\oddsidemargin 0.2cm
\begin{document}



\end{document}
```

Eventuell müssen Sie in der dritten Zeile den Eintrag `utf8` in der eckigen Klammer durch `latin1` ersetzen. Dies hängt davon ab, welche Kodierung Ihr Editor beim Abspeichern verwendet. Werden im Erscheinungsbild des Textes die deutschen Umlaute nicht richtig dargestellt, so ist in der Regel in der Präambel die falsche Kodierung gewählt worden.

Mit dem Kommando `\pagestyle` wird die Art der Seitennummerierung festgelegt. Hier sind folgende Parameter möglich:

`empty`	Keine Seitennummerierung.
`plain`	Die Seitennummerierung erfolgt als zentrierte Fußzeile.
`headings`	Die Seitenzahl wird als Kopfzeile zusammen mit einer automatisch gewählten Überschrift-Information ausgegeben. In der Regel ist dies die augenblickliche Abschnittsüberschrift.
`myheadings`	Die Seitenzahl wird als Kopfzeile zusammen mit einer selbst zu definierenden Überschrift ausgegeben.

Der eigentliche Text (einschließlich Kommandos für das Erscheinungsbild) steht zwischen den Befehlen

```
\begin{document}

\end{document}
```

1.2.2 Steuerzeichen

Einige Zeichen werden vom Latex-Formatierer als Steuerzeichen interpretiert, so z. B.

\	leitet einen Latex-Befehl ein.
%	dient als Kommentarzeichen: der Text rechts von % bis zum Zeilenende wird von Latex ignoriert.
$	wird für die Kennzeichnung von mathematischen Formeln verwendet.
&	ist das Trennungszeichen für die einzelnen Spalteneinträge in Tabellen.
{ bzw. }	dienen zur Markierung von Blöcken.

Sollen diese Zeichen im Text erscheinen, so sind sie mit `$\backslash$, \%, \$, \&, \{` bzw. `\}` einzugeben.

1.2.3 Text- und Mathematik-Modus

Latex unterscheidet zwischen einem Text-Modus (Voreinstellung) und einem Mathematik-Modus (für mathematische Symbole und Formeln). Im professionellen Buchdruck haben die Buchstaben a bzw. A und die mathematischen Größen a bzw. A ein unterschiedliches Erscheinungsbild.

Das Umschalten in den Mathematik-Modus erfolgt

1. im laufenden Text durch $ ... $. Die Zeichen zwischen den Dollarzeichen werden im Layout als mathematische Symbole dargestellt.
2. für eine gesonderte Formelzeile durch \[... \] bzw. $$ ... $$.

Latex-Befehle für mathematische Symbole funktionieren nur im Mathematik-Modus. Das folgende kleine Beispiel (ohne Präambel) demonstriert das Umschalten in den Mathematik-Modus. Die Texteingabe

```
Im professionellen Buchdruck haben die Buchstaben a bzw. A
und die mathematischen Größen $a$ bzw. $A$ ein unterschiedliches
Erscheinungsbild.
\vspace{1ex} \newline
Umfangreiche Formeln, wie zum Beispiel
\[
 K := \{ (a,b) \in \mathbb{R}^2 \: : \: a^2 + b^2 = 1 \}
\]
werden in eine eigene Formelzeile gesetzt.
```

liefert das Layout

> Im professionellen Buchdruck haben die Buchstaben a bzw. A und die mathematischen Größen a bzw. A ein unterschiedliches Erscheinungsbild.
>
> Umfangreiche Formeln, wie zum Beispiel
>
> $$K := \{(a, b) \in \mathbb{R}^2 \ : \ a^2 + b^2 = 1\}$$
>
> werden in eine eigene Formelzeile gesetzt.

1.2.4 Zeilen- und Seitenumbruch

Zeilen- und Seitenumbruch sowie die Seitennummerierung erfolgen in Latex automatisch. Durch das Kommando

```
\usepackage[ngerman]{babel}
```

in der Präambel wird beim Zeilenumbruch die deutsche Silbentrennung verwendet. Dabei treten jedoch vereinzelt Fehler auf. So wird beispielsweise das Wort Strichoperationen falsch getrennt: Stri-choperationen. Für eine korrekte Silbentrennung muss man in diesem Fall dem Latex-Formatierer die Trennung bei der Texteingabe vorschreiben: `Strich\-operationen`.

Das Kommando `\newline` bzw. \\ bewirkt eine neue Zeile im Layout. Eine neue Zeile bei der Texteingabe hat keinen Einfluss auf den Zeilenumbruch im Layout.

Dagegen bewirkt eine Leerzeile im Eingabetext einen neuen Absatz mit einem vergrößerten Zeilenabstand, wobei das erste Wort eingerückt wird. Mit dem Befehl `\noindent` verhindert man das Einrücken.

Das Kommando `\newpage` erzwingt eine neue Seite.

1.2.5 Abstände im Text

Latex wählt die Abstände zwischen den Worten, zwischen den Zeilen und in den Formeln automatisch. Es gibt jedoch einige Befehle, um hierauf Einfluss zu nehmen.

Horizontale Abstände

Am flexibelsten ist das Kommando `\hspace{<abstand>}`. Dabei besteht `<abstand>` aus einer Dezimalzahl und einer Maßeinheit. Als Maßeinheit sind u. A. erlaubt:

absolute Maßeinheiten	relative Maßeinheiten
`cm` Zentimeter	`em` Breite von **m**
`mm` Millimeter	`ex` Höhe von **x**

Relative Maße haben den Vorteil, dass bei Änderung der Schriftgröße die Abstände ebenfalls angepasst werden.

Weitere Möglichkeiten für (horizontale) Abstände sind

`\,`	verkürzte Leerstelle
`\quad`	vergrößerter Abstand
`\qquad`	stark vergrößerter Abstand

Beispiele

Eingabe	Layout	Eingabe	Layout
`abcd efgh`	abcd efgh	`abcd\,efgh`	abcd efgh
`abcd\quad efgh`	abcd efgh	`abcd\hspace{1cm}efgh`	abcd efgh
`abcd\qquad efgh`	abcd efgh	`abcd\hspace{1em}efgh`	abcd efgh

Vertikale Abstände

Für vertikale Abstände zwischen den einzelnen Zeilen gibt es analog den Befehl

`\vspace{<abstand>}`.

Dieses Kommando bewirkt, dass beim nächsten Zeilenumbruch ein entsprechender Abstand eingefügt wird.

Für Abstände zwischen den einzelnen Absätzen gibt es die Befehle `\smallskip` (kleiner Abstand), `\medskip` (mittlerer Abstand) und `\bigskip` (großer Abstand).

Anmerkung: Fällt im Erscheinungsbild das hspace-Kommando auf das Zeilenende bzw. auf den Zeilenanfang, so wird es ignoriert. Analoges gilt für das vspace-Kommando am Seitenende bzw. Seitenanfang. Ist hier trotzdem ein bestimmter Abstand gewünscht, so wird

`\hspace*{<abstand>}` bzw. `\vspace*{<abstand>}`

verwendet.

1.2.6 Schriftarten und Schriftgrößen

Für die Wahl des Schriftbildes gibt es standardmäßig folgende Möglichkeiten:

Name	Kommando	Layout
roman	`\rm`	ergibt dieses Schriftbild
bold face	`\bf`	**ergibt dieses Schriftbild**
slanted	`\sl`	*ergibt dieses Schriftbild*
italic	`\it`	*ergibt dieses Schriftbild*
typewriter	`\tt`	`ergibt dieses Schriftbild`
sans serif	`\sf`	ergibt dieses Schriftbild
small caps	`\sc`	ERGIBT DIESES SCHRIFTBILD

Beispiel: Die Eingabe

```
Voreingestellt ist die Schriftart {\bf roman}. Sie
wird so lange verwendet, bis ein Befehl für eine
andere Schriftart erfolgt. Wird ein neue Schriftart
in einem mit geschweiften Klammern markierten
Block gewählt, so gilt diese Änderung nur in
diesem Block.
\newline
\sc Ab jetzt wird auf die Schriftart {\tt small
caps} umgeschaltet.
```

erzeugt das Schriftbild

Voreingestellt ist die Schriftart **roman**. Sie wird so lange verwendet, bis ein Befehl für eine andere Schriftart erfolgt. Wird ein neue Schriftart in einem mit geschweiften Klammern markierten Block gewählt, so gilt diese Änderung nur in diesem Block.
Ab jetzt wird auf die Schriftart `small caps` umgeschaltet.

In Latex gibt es viele weitere Schriftsätze, deren Aufruf jedoch komplizierter ist. Hier verweisen wir auf die Literatur.

Latex kennt folgende Schriftgrößen:

\Huge	Beispiel
\huge	Beispiel
\LARGE	Beispiel
\Large	Beispiel
\large	Beispiel
\normalsize	Beispiel
\small	Beispiel
\footnotesize	Beispiel
\scriptsize	Beispiel
\tiny	Beispiel

Voreinstellung ist `normalsize`, deren Größe im Layout in der ersten Zeile der Präambel

```
\documentclass[12pt]{article}
```

in der eckigen Klammer festgelegt wird. Hier sind `10pt`, `11pt` oder `12pt` möglich. Dabei ist pt (points) die Maßeinheit: 72.27 pt = 1 inch = 2.54 cm.

1.2.7 Latex-Umgebungen

Eine *Umgebung* hat die Struktur

```
\begin{<umgebung>}
   <text>
\end{<umgebung>}
```

und bewirkt, dass der innerhalb dieser Umgebung stehende Text anders behandelt wird, z.B. als Tabelle oder als mathematische Gleichung. Einige Umgebungen habe optionale und/oder zwingende Parameter:

```
\begin{<umgebung>}[<optpar>]{<zwingpar>}
```

Beispiel: Für Texte, die zentriert gesetzt werden sollen (etwa Gedichte), gibt es die `center` - Umgebung.

```
\begin{center}
Dieser Text wird \\ zentriert gesetzt;
\end{center}
```

Dieser Text wird
zentriert gesetzt;

1.3 Wichtige Elemente im Text-Modus

1.3.1 Listen

Latex stellt einige Umgebungen für Listen bereit, und zwar in Abhängigkeit von der gewünschten Darstellung der einzelnen Listenpunkte.

Markierung durch ein Symbol

Hierzu gibt es die `itemize` - Umgebung:

```
\begin{itemize}
  \item <text>
  \item <text>
        ⋮
\end{itemize}
```

Diese Umgebung darf auch geschachtelt werden. Es werden dann unterschiedliche Markierungssymbole verwendet, wie das folgende Beispiel zeigt:

```
\begin{itemize}
  \item Punkt 1
  \item Punkt 2
     \begin{itemize}
        \item Unterpunkt 1
        \item Unterpunkt 2
           \begin{itemize}
              \item Unterpunkt 2.1
              \item Unterpunkt 2.2
           \end{itemize}
     \end{itemize}
  \item Punkt 3
\end{itemize}
```

- Punkt 1
- Punkt 2
  - Unterpunkt 1
  - Unterpunkt 2
    - Unterpunkt 2.1
    - Unterpunkt 2.2
- Punkt 3

Anmerkung: Zur besseren Lesbarkeit empfehlen wir, den Text innerhalb einer Latex-Umgebung etwas einzurücken (dies ist insbesondere bei der Fehlersuche von Vorteil). Für den Latex-Formatierer ist ein Einrücken nicht erforderlich; er kommt auch damit zurecht:

```
\begin{itemize}\item Punkt 1\item Punkt 2\begin{itemize}\item Unterpunkt 1
\item Unterpunkt 2\begin{itemize}\item Unterpunkt 2.1\item Unterpunkt 2.2
\end{itemize}\end{itemize}\item Punkt 3\end{itemize}
```

Nummerierung der Listenpunkte

Für nummerierte Aufzählungen gibt es die `enumerate` - Umgebung.

```
\begin{enumerate}
  \item <text>
  \item <text>
        ⋮
\end{enumerate}
```

Beispiel

```
\begin{enumerate}
  \item Wintersemester
    \begin{enumerate}
      \item Analysis I
      \item Lineare Algebra I
    \end{enumerate}
  \item Sommersemester
    \begin{enumerate}
      \item Analysis II
      \item Lineare Algebra II
      \item Coma
    \end{enumerate}
\end{enumerate}
```

1. Wintersemester
   (a) Analysis I
   (b) Lineare Algebra I
2. Sommersemester
   (a) Analysis II
   (b) Lineare Algebra II
   (c) Coma

Stichworte für die Listenpunkte

Eine dritte Möglichkeit ist die `description` - Umgebung. Hier werden die einzelnen Punkte durch ein fettgedrucktes Wort markiert.

```
\begin{description}
  \item [<wort1>] <text>
  \item [<wort2>] <text>
        ⋮
\end{description}
```

Beispiel:

```
In dieser Vorlesung werden folgende Themen behandelt:
\begin{description}
  \item[Latex:] In den ersten drei Wochen geben wir eine
     kurze Einführung in das Textverarbeitungssystem Latex.
  \item[Matlab:] Der Schwerpunkt dieser Lehrveranstaltung
```

```
      liegt auf der Vermittlung der Grundlagen von Matlab.
      Diese Kenntnisse werden im nächsten Semester für die
      Vorlesung {\sc Numerik~I} benötigt.
\end{description}
```

In dieser Vorlesung werden folgende Themen behandelt:

Latex: In den ersten drei Wochen geben wir eine kurze Einführung in das Textverarbeitungssystem Latex.

Matlab: Der Schwerpunkt dieser Lehrveranstaltung liegt auf der Vermittlung der Grundlagen von Matlab. Diese Kenntnisse werden im nächsten Semester für die Vorlesung NUMERIK I benötigt.

1.3.2 Tabellen

Für Tabellen gibt es die `tabular` - Umgebung, welche folgenden formalen Aufbau hat:

```
\begin{tabular}[<pos>]{<spalten>}
  <text11> & <text12> & ... & <text1n> \\
  <text21> & <text22> & ... & <text2n> \\
                 ⋮
\end{tabular}
```

Der **optionale** Parameter <pos> dient zur horizontalen Ausrichtung der Tabelle innerhalb der laufenden Textzeile:

`t` (top)	oberste Tabellenzeile ist auf dem Level der Textzeile
`b` (bottom)	unterste Tabellenzeile ist auf dem Level der Textzeile
ohne opt. Param.	Tabellenmitte ist auf dem Level der Textzeile

Der **obligatorische** Parameter <spalten> enthält die Anzahl und die Ausrichtung der Spalten. Für jede Spalte ist genau eines der folgenden drei Formatierungszeichen anzugeben:

`l` (left)	linksbündige Ausrichtung der betreffenden Spalte
`c` (center)	zentrierte Ausrichtung der betreffenden Spalte
`r` (right)	rechtsbündige Ausrichtung der betreffenden Spalte

Bei der Texteingabe dient `&` als Trennungzeichen für die einzelnen Spalten und \\ als Zeilenende.

Beispiel

```
top \hspace{1em}
\begin{tabular}[t]{lcr}
     3 & 145 & -2 \\
    -12 & 1 & 35 \\
    10 & -5  & 165
\end{tabular}
\hspace{1em} bottom \hspace{1em}
\begin{tabular}[b]{lcr}
     3 & 145 & -2 \\
    -12 & 1 & 35 \\
    10 & -5  & 165
\end{tabular}
\hspace{1em} zentriert \hspace{1em}
\begin{tabular}{lcr}
     3 & 145 & -2 \\
    -12 & 1 & 35 \\
    10 & -5  & 165
\end{tabular}
```

top 3 145 -2 / -12 1 35 / 10 -5 165 bottom 3 145 -2 / -12 1 35 / 10 -5 165 zentriert 3 145 -2 / -12 1 35 / 10 -5 165

Selbstverständlich kann eine Tabelle auch mit einem Rahmen versehen werden. Waagerechte Linien erhält man mit dem Befehl \hline, senkrechte durch Angabe von | im obligatorischen Parameter <spalten>.

Beispiel

```
\begin{center}
   \begin{tabular}{|c|c|c|c|} \hline
     & Teilnehmer &  bestanden  &  Prozent \\
     \hline \hline
     gesamt &  143 & 112 & 78.3 \% \\ \hline \hline
     Bachelor & 86 & 69 & 80.2 \% \\ \hline
     Diplom/Lehramt & 57 & 43 & 75.4 \% \\ \hline \hline
     Math. BA & 18 & 15 & 83.3 \% \\ \hline
     Math. Dipl. & 14 & 9 & 64.3 \% \\ \hline
     Lehramt & 41 & 33 & 80.5 \% \\ \hline
     MFÖ & 32 & 29 & 90.6 \% \\ \hline
     Phys. BA & 31 & 24 & 77.4 \% \\ \hline
     Sonstige & 7 & 2 & 28.6 \% \\ \hline
   \end{tabular}
\end{center}
```

	Teilnehmer	bestanden	Prozent
gesamt	143	112	78.3 %
Bachelor	86	69	80.2 %
Diplom/Lehramt	57	43	75.4 %
Math. BA	18	15	83.3 %
Math. Dipl.	14	9	64.3 %
Lehramt	41	33	80.5 %
MFÖ	32	29	90.6 %
Phys. BA	31	24	77.4 %
Sonstige	7	2	28.6 %

1.3.3 Gliederung

Ein größerer Text ist überlicherweise gegliedert, z.B. in Kapitel, Paragraphen, Abschnitte, Unterabschnitte. Die Überschriften werden meist nummeriert und in Extrazeilen mit größerer (Fett)Schrift gesetzt.

In Latex hängt die Anzahl der Gliederungsebenen von der in der Präambel in

`\documentclass[12pt]{article}`

festgelegten Dokument-Klasse ab. In der von uns gewählten `article` - Klasse verwendet man im Normalfall die drei Ebenen

`\section{`<überschrift1>`}`

`\subsection{`<überschrift2>`}`

`\subsubsection{`<überschrift3>`}`

Darunter gibt es noch zwei weitere Gleiderungsklassen, auf die wir hier nicht eingehen.

Für die `book` - bzw. `report` - Klasse gibt es oberhalb der `section` - Ebene noch eine Ebene

`\chapter{`<kapitel>`}`

Beim Aufruf einer dieser Ebenen vergibt Latex eine Gliederungsnummer. Fügt man später einen zusätzlichen Gliederungsbefehl ein, so wird diese Nummerierung automatisch angepasst. Mit dem Befehl

`\setcounter{`<ebene>`}{`<nr>`}`

können wir die Nummerierung beeinflussen.

Beispiel

```
\section{Differentialrechnung}
\subsection{Die Ableitung einer reellen Funktion}
\subsubsection{Die 1. Ableitung}
Hier steht ein Text.
\subsubsection{Die 2. Ableitung}
Hier folgt weiterer Text.
\subsubsection{Ableitungsregeln}
Hierher gehört zum Beispiel die Produktregel.
\subsection{Die Ableitung eines Vektorfeldes}
```

```
\setcounter{section}{5}
\section{Anwendungen der Differentialrechnung}
```

1 Differentialrechnung

1.1 Die Ableitung einer reellen Funktion

1.1.1 Die 1. Ableitung

Hier steht ein Text.

1.1.2 Die 2. Ableitung

Hier folgt weiterer Text.

1.1.3 Ableitungsregeln

Hierher gehört zum Beispiel die Produktregel.

1.2 Die Ableitung eines Vektorfeldes

6 Anwendungen der Differentialrechnung

1.3.4 Inhaltsverzeichnis

Latex speichert die Gliederungsüberschriften zusammen mit ihrer Nummerierung und Seitenzahl in der Datei

<name>`.toc` (`toc` = table of contents).

Mit dem Kommando

`\tableofcontents`

wird daraus ein Inhaltsverzeichnis erzeugt, und zwar an der Stelle im Text, an der dieser Befehl steht (in der Regel nach dem Titelblatt). Dabei wird die Datei <name>`.toc` verwendet, die beim vorigen Latex-Aufruf erzeugt wurde. Deshalb muss das Latex-Kommando zweimal ausgeführt werden, um das korrekte Inhaltsverzeichnis zu erhalten.

1.3.5 Der verbatim-Modus

Die Eingabe des Textes (mehrere Leerzeichen, Zeilenumbruch) hat keinen Einfluss auf das Layout. Gelegentlich möchte man jedoch den Text einschließlich Steuerbefehle so ausgegeben, wie er eingetippt wurde. Dazu dient die `verbatim` - Umgebung. Der zwischen

```
\begin{verbatim}
        ⋮
\end{verbatim}
```

eingeschlossene Text (einschließlich Steuerbefehle) wird in der Schriftart `\tt` so dargestellt, wie er eingegeben wurde. Die darin enthaltenen Latex-Kommandos werden ignoriert.

Beispiel: Die Eingabe von

```
\begin{verbatim}
    \section{Differentialrechnung}
    \subsection{\Large Die Ableitung einer reellen Funktion}
    \subsubsection{Die 1. Ableitung}
    Hier steht ein Text
    \subsubsection{Die 2. Ableitung}
    Hier folgt weiterer Text
    \subsubsection{Ableitungsregeln}
    Hierher gehört zum Beispiel die Produktregel.
\end{verbatim}
```

liefert folgendes Erscheinungsbild:

```
\section{Differentialrechnung}
\subsection{\Large Die Ableitung einer reellen Funktion}
\subsubsection{Die 1. Ableitung}
Hier steht ein Text
\subsubsection{Die 2. Ableitung}
Hier folgt weiterer Text
\subsubsection{Ableitungsregeln}
Hierher gehört zum Beispiel die Produktregel.
```

1.3.6 Textboxen

Eine Box (oder auch ein Kasten) besteht aus einem Text, der von Latex wie ein einzelnes Zeichen behandelt wird. Dafür gibt es eine Reihe von Möglichkeiten; wir behandeln hier nur eine kleine Auswahl.

Zu den vertikalen Boxen gehören die `parbox` und die `minipage` - Umgebung:

`\parbox[`<pos>`]{`<breite>`}{`<text>`}`

bzw.

`\begin{minipage}[`<pos>`]{`<breite>`}`
<text>
`\end{minipage}`

Der optionale Parameter <pos> dient der horizontalen Ausrichtung und hat dieselbe Bedeutung wie bei der `tabular` - Umgebung (vgl. Abschnitt 1.3.2).

Der obligatorische Parameter <breite> gibt die Breite der Box an, die wie eine kleine Seite behandelt wird: am Zeilenende (= Breite der Box) wird automatisch umgebrochen. Die Höhe der Box richtet sich nach der Größe des Textes.

Schließlich wird durch `\fbox{`<text>`}` der ausgewählte Text eingerahmt.

Beispiel

```
In der \fbox{laufenden Zeile} kommt hier eine
  \fbox{\fbox{ \begin{minipage}[t]{5cm}
       eingerahmte Minipage, welche aus zwei Zeilen besteht.
       \end{minipage} }}
Anschließend geht es in der normalen Textzeile weiter.
```

In der laufenden Zeile kommt hier eine eingerahmte Minipage, welche aus zwei Zeilen besteht. Anschließend geht es in der normalen Textzeile weiter.

1.3.7 Querverweise

In Lehrbüchern oder wissenschaftlichen Arbeiten sind Querverweise auf andere Textstellen üblich. In Latex wird die Stelle, auf die Bezug genommen werden soll, markiert durch das Kommando

`\label{`<marke>`}` ,

welches meistens nach einer Überschrift oder in einer mathematischen Formel steht. Damit enthält <marke> die aktuelle Nummerierung, auf die an anderer Textstelle durch

`\ref{`<marke>`}`

verwiesen wird. Der Name <marke> ist frei wählbar, muss aber im gesamten Latex-Dokument eindeutig sein. Durch das Kommando

`\pageref{`<marke>`}`

verweist man auf die entsprechende Seite.

Beispiel

```
\subsubsection{Tabellen}
\label{latex_tabelle}
          ⋮
```

An einer anderen Textstelle wird dann auf diesen Punkt (diese Seite) verwiesen:

```
Wir verweisen auf die in Abschnitt \ref{latex_tabelle}
beschriebene {\tt tabular}-Umgebung. \newline
Die {\tt tabular}-Umgebung wird auf Seite \pageref{latex_tabelle}
beschrieben.
```

Wir verweisen auf die in Abschnitt 1.3.2 beschriebene `tabular`-Umgebung.
Die `tabular`-Umgebung wird auf Seite 13 beschrieben.

1.3.8 Literaturverzeichnis

Zu wissenschaftlichen Arbeiten gehört die Angabe der verwendeten Literatur (in der Regel am Ende des Textes). Ebenso muss an den betroffenen Textstellen auf die entsprechende Literatur verwiesen werden.

Das Literaturverzeichnis wird erzeugt durch

```
\begin{thebibliography}{<muster>}
  \bibitem{<name1>}  <text>
  \bibitem{<name2>}  <text>
  \bibitem{<name3>}  <text>
          ⋮
\end{thebibliography}
```

Der Eintrag für <muster> bestimmt die Breite für das Einrücken von <text>. Handelt es sich um eine zweistellige Anzahl von Literaturangaben, so könnte man

```
\begin{thebibliography}{99}
```

verwenden.

Das Zitieren der entsprechenden Literatur im Text erfolgt durch

```
\cite{<name1>}
```

Beispiel

```
Wir verweisen auf die in \cite{roesch} gemachten Ausführungen.

\begin{thebibliography}{99}
\bibitem{brunner} {\bf Brunner; Brück:} Mathematik für Chemiker,
         2. Auflage. Springer 2008.
\bibitem{zachmann} {\bf Zachmann; Jüngel:} Mathematik für Chemiker,
         6. Auflage. Wiley-VCH 2007.
\bibitem{reinsch} {\bf Reinsch:} Mathematik für Chemiker,
         Teubner 2004.
\bibitem{roesch} {\bf Rösch:} Mathematik für Chemiker, Springer 1993.
\bibitem{scherfner} {\bf Scherfner; Senkbeil:}
       Lineare Algebra für das erste Semester.
        Pearson Studium 2006.
\end{thebibliography}
```

Wir verweisen auf die in [4] gemachten Ausführungen.

Literatur

[1] **Brunner; Brück:** Mathematik für Chemiker, 2. Auflage. Springer 2008.

[2] **Zachmann; Jüngel:** Mathematik für Chemiker, 6. Auflage. Wiley-VCH 2007.

[3] **Reinsch:** Mathematik für Chemiker, Teubner 2004.

[4] **Rösch:** Mathematik für Chemiker, Springer 1993.

[5] **Scherfner; Senkbeil:** Lineare Algebra für das erste Semester. Pearson Studium 2006.

1.4 Wichtige Elemente im Mathematik-Modus

1.4.1 Die Schriftart `mathbb`

Zur Darstellung der Zahlbereiche $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}, \ldots$ gibt es in Latex die Schriftart `mathbb`, welche nur im Mathematikmodus möglich ist. Dieser Zeichensatz ist nur für Großbuchstaben definiert.

Der Aufruf erfolgt durch das Kommando

```
\mathbb{<buchstabe>}
```

und liefert das folgende Alphabet:

$\mathbb{A}$, $\mathbb{B}$, $\mathbb{C}$, $\mathbb{D}$, $\mathbb{E}$, $\mathbb{F}$, $\mathbb{G}$, $\mathbb{H}$, $\mathbb{I}$, $\mathbb{J}$, $\mathbb{K}$, $\mathbb{A}$, $\mathbb{L}$, $\mathbb{M}$,
$\mathbb{N}$, $\mathbb{O}$, $\mathbb{P}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{S}$, $\mathbb{T}$, $\mathbb{U}$, $\mathbb{V}$, $\mathbb{W}$, $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{Z}$,

1.4.2 Griechische Buchstaben

α	\alpha	A	A	ν	\nu	N	N
β	\beta	B	B	ξ	\xi	Ξ	\Xi
γ	\gamma	Γ	\Gamma	o	o	O	O
δ	\delta	Δ	\Delta	π	\pi	Π	\Pi
ϵ	\epsilon	E	E	ρ	\rho	P	P
ε	\varepsilon	E	E	ϱ	\varrho	P	P
ζ	\zeta	Z	Z	σ	\sigma	Σ	\Sigma
η	\eta	H	H	τ	\tau	T	T
θ	\theta	Θ	\Theta	υ	\upsilon	Υ	\Upsilon
ι	\iota	I	I	ϕ	\phi	Φ	\Phi
κ	\kappa	K	K	χ	\chi	X	X
λ	\lambda	Λ	\Lambda	ψ	\psi	Ψ	\Psi
μ	\mu	M	M	ω	\omega	Ω	\Omega

1.4.3 Mathematische Symbole

$\subset$	\subset	$\supset$	\supset
$\subseteq$	\subseteq	$\supseteq$	\supseteq
$\cup$	\cup	$\cap$	\cap
$\in$	\in	$\ni$	\ni
$\notin$	\not \in	$\neq$	\neq
$\leq$	\leq	$\geq$	\geq
$\ll$	\ll	$\gg$	\gg
$\approx$	\approx	$\neq$	\neq
$\sim$	\sim	$\simeq$	\simeq
$\mid$	\mid	$\parallel$	\parallel
$\pm$	\pm	$\mp$	\mp
$*$	\ast	$\circ$	\circ
$\times$	\times	$\perp$	\perp

1.4.4 Pfeile

$\rightarrow$	\rightarrow	$\leftarrow$	\leftarrow
$\uparrow$	\uparrow	$\downarrow$	\downarrow
$\Rightarrow$	\Rightarrow	$\Leftarrow$	\Leftarrow
$\Uparrow$	\Uparrow	$\Downarrow$	\Downarrow
$\longrightarrow$	\longrightarrow	$\longleftarrow$	\longleftarrow
$\Longrightarrow$	\Longrightarrow	$\Longleftarrow$	\Longleftarrow
$\leftrightarrow$	\leftrightarrow	$\updownarrow$	\updownarrow
$\Leftrightarrow$	\Leftrightarrow	$\Updownarrow$	\Updownarrow
$\longleftrightarrow$	\longleftrightarrow	$\Longleftrightarrow$	\Longleftrightarrow
$\Longleftrightarrow$	\iff	$\to$	\to
$\mapsto$	\mapsto		

1.4.5 Mathematische Akzente

Die mathematischen Akzente erhält man wie folgt:

\hat{a}	$\hat{a}$
\bar{a}	$\bar{a}$
\tilde{a}	$\tilde{a}$
\vec{a}	$\vec{a}$
\dot{a}	$\dot{a}$
\ddot{a}	$\ddot{a}$

Für einige dieser Akzente gibt es eine Breitversion:

\widehat{AB}	$\widehat{AB}$
\widetilde{x-y}	$\widetilde{x-y}$

1.4.6 Standardfunktionen

Für die Standardfunktionen werden die in der Mathematik üblichen Bezeichnungen mit einem vorangestellten Backslash verwendet. Beachten Sie den Unterschied im Schriftbild:

richtig: $\exp(ix) = \cos(x) + i\sin(x)$ `$\exp(ix) = \cos(x) + i \sin(x)$`

falsch: $exp(ix) = cos(x) + isin(x)$ `$exp(ix) = cos(x) + i sin(x)$`

Wurzelfunktionen erhält man durch das `\sqrt` - Kommando, zum Beispiel

`\sqrt{2x+1}` $\sqrt{2x+1}$

`\sqrt[5]{2x+1}` $\sqrt[5]{2x+1}$

1.4.7 Hochzahlen und Indizes

Für obere Indizes wird das `^` - Zeichen, für untere Indizes wird das `_` - Zeichen verwendet. Sollen mehrere Zeichen hoch- bzw. tiefgestellt werden, so sind sie in geschweifte Klammern einzuschließen.

Beispiele

`$(a+b)^{13}$` $(a+b)^{13}$

`$A_i^k$` A_i^k

`$2^{2^{i+1}}$` $2^{2^{i+1}}$

Beachten Sie den Unterschied zwischen `$a^{13}$` a^{13} und `$a^13$` a^13 .

1.4.8 Summen, Produkte, Integrale

Für Summen, Produkte und Integrale gibt es zwei Möglichkeiten, abhängig vom gewünschten Erscheinungsbild:

`$\sum_{i=1}^{10} a_i$` $\sum_{i=1}^{10} a_i$

`$\sum\limits_{i=1}^{10} a_i$` $\sum\limits_{i=1}^{10} a_i$

`$\prod_{i=1}^{10} a_i$` $\prod_{i=1}^{10} a_i$

`$\prod\limits_{i=1}^{10} a_i$` $\prod\limits_{i=1}^{10} a_i$

`$\int_a^b f(x) \, dx $` $\int_a^b f(x)\,dx$

`$\int\limits_a^b f(x) \, dx$` $\int\limits_a^b f(x)\,dx$

In Formelzeilen kann auf den `limits` - Befehl verzichtet werden.

Mit dem `shortstack`-Kommando können mehrere Zeilen übereinander gesetzt werden:

```
$\sum\limits_{\scriptsize \shortstack{$i=1$\\$i \neq k$}}^m (t_i-t_k)$
```

erzeugt die Ausgabe $\sum\limits_{\substack{i=1\\ i\neq k}}^{m} (t_i - t_k)$.

1.4.9 Grenzwerte

Auch für die Darstellung von Grenzwerten gibt es diese zwei Varianten. In diesem Zusammenhang wird häufig auch das ∞-Zeichen gebraucht, welches in Latex durch das Kommando `\infty` erzeugt wird.

```
$\lim_{x \to \infty} e^{-x} = 0$
```
$\lim_{x \to \infty} e^{-x} = 0$

```
$\lim\limits_{x \to \infty} e^{-x} = 0$
```
$\lim\limits_{x \to \infty} e^{-x} = 0$

1.4.10 Brüche

Zur Darstellung von Brüchen verwendet man das Kommando

`\frac{`<zähler>`}{`<nenner>`}`

Selbstverständlich sind in Brüchen wieder Brüche möglich.

Beispiele

```
\frac{a+b}{c-d}
```
$\frac{a+b}{c-d}$

```
\frac{2+\frac{1}{1+\exp(2-x)}}{\frac{x^3+2x+1}{x-1}}
```
$$\frac{2+\frac{1}{1+\exp(2-x)}}{\frac{x^3+2x+1}{x-1}}$$

1.4.11 Dynamische Klammern

Latex kann Klammern an die Größe von Formeln anpassen. Dies wird erreicht durch

`\left`<*klammer*> . . . `\right`<*klammer*>

Wichtig: zu jedem `\left` gehört ein `\right`. Dabei darf es sich um unterschiedliche Klammern handeln.

Beispiel

```
\left( \sum \limits _{i=1}^{10} a_i \right)^2
```
$$\left(\sum \limits _{i=1}^{10} a_i \right)^2$$

Ohne dynamische Klammerung ergibt sich

```
( \sum \limits _{i=1}^{10} a_i )^2
```
$$(\sum \limits _{i=1}^{10} a_i)^2$$

1.4.12 Vektoren und Matrizen

Für Vektoren und Matrizen gibt es die `array` - Umgebung, welche wie die `tabular` - Umgebung aufgebaut ist, jedoch nur im Mathematik-Modus funktioniert:

```
\begin{array}[<pos>]{<spalten>}
  <element11> & <element12> & ... & <element1n> \\
  <element21> & <element22> & ... & <element2n> \\
                 ⋮
\end{array}
```

Die Parameter <pos> und <spalten> entsprechen denen der tabular-Umgebung, vgl. Abschnitt 1.3.2.

Im Zusammenhang mit Matizen treten folgende Symbole auf:

\cdots $\cdots$ \vdots $\vdots$ \ddots $\ddots$ \ldots $\ldots$.

Beispiel

```
\[
   B =
    \begin{array}{cccc}
     1 & 2 & 3 & 4 \\
     5 & 6 & 7 & 8
    \end{array}
   ,\hspace{1em}
   A = \left(
    \begin{array}{ccc}
     a_{11} & \cdots & a_{1n} \\
     \vdots &  & \vdots \\
     a_{n1} & \cdots & a_{nn}
    \end{array}
  \right)
\]
```

$$B = \begin{array}{cccc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{array}, \quad A = \left(\begin{array}{ccc} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{array} \right)$$

Einfacher (aber nicht mehr so flexibel) ist die pmatrix-Umgebung, welche alle Spalten zentriert ausrichtet und die runden Klammern automatisch setzt.

Beispiel

```
\[
 \begin{pmatrix}
   a_{11} & a_{12} & a_{13} \\
   a_{21} & a_{22} & a_{23} \\
   a_{31} & a_{32} & a_{33}
 \end{pmatrix}
\]
```

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

1.4.13 Formeln mit automatischer Nummerierung

Dafür gibt es die `equation`-Umgebung. Diese wird im Text-Modus aufgerufen und wählt automatisch den Mathematik-Modus.

```
\begin{equation}
   ....
\label{<name>}
\end{equation}
```

Dabei ist `<name>` frei wählbar.

Beispiel

```
Gegeben sei die Funktion
\begin{equation}
    F(x,y) = \frac{2xy}{\sqrt{1 + x^2 +(y-1)^2}}\;
    \mbox{ für }(x,y) \in \mathbb{R}^2.
    \label{formel1}
\end{equation}
Für die Funktion in (\ref{formel1}) bestimmen
wir die partielle Ableitung
\[
  \frac{\partial F(x,y)}{\partial x}
\]
```

Gegeben sei die Funktion

$$F(x,y) = \frac{2xy}{\sqrt{1 + x^2 + (y-1)^2}} \text{ für } (x,y) \in \mathbb{R}^2. \tag{1}$$

Für die Funktion in (1) bestimmen wir die partielle Ableitung

$$\frac{\partial F(x,y)}{\partial x}\,.$$

In diesem Beispiel tritt das Symbol ∂ `\partial` auf, welches in der Mathematik für partielle Ableitungen verwendet wird. Mit `\mbox{<text>}` wird eine Textbox im Mathematik-Modus erzeugt.

Die gewöhnliche Ableitungen einer reellen Funktion erhalten wir durch

`f^{\prime}(x)` $\rightsquigarrow$ $f'(x)$,

`f^{\prime\prime}(x)` $\rightsquigarrow$ $f''(x)$,

`f^{(n)}(x)` $\rightsquigarrow$ $f^{(n)}(x)$.

1.4.14 Mehrzeilige Formeln

Hier gibt es zwei Möglichkeiten. Ist eine automatische Nummerierung der einzelen Formelzeilen gewünscht, so wird die `eqnarray`-Umgebung verwendet, ansonsten wird die `eqnarray*`-Umbebung. Diese Formeln bestehen aus genau 3 Spalten, wobei die erste

Spalte rechtsbündig, die zweite zentriert und die dritte linksbündig dargestellt wird.

```
\begin{eqnarray}
 <ausdruck1> & <ausdruck2> & <ausdruck3> \label{ <name1> } \\
 <ausdruck4> & <ausdruck5> & <ausdruck6> \label{ <name2> } \\
                 ⋮
\end{eqnarray}
```

Beispiel 1 (mit Nummerierung)

```
Gegeben sei das Gleichungssystem
\begin{eqnarray}
  x + y & = & 10  \label{f1} \\
  x^2 + y^2 + z^2 & = & 30 \label{f2} \\
  2xy + xyz + 4 z & = & 90 \label{f3}
\end{eqnarray}
Wir lösen in (\ref{f1}) nach $x$ auf
und setzen das Ergebnis in (\ref{f2})
und (\ref{f3}) ein.
```

Gegeben sei das Gleichungssystem

$$\begin{aligned} x + y &= 10 && (2)\\ x^2 + y^2 + z^2 &= 30 && (3)\\ 2xy + xyz + 4z &= 90 && (4) \end{aligned}$$

Wir lösen in (2) nach x auf und setzen das Ergebnis in (3) und (4) ein.

Sollen einzelne Zeilen nicht mit einer Nummer versehen werden, so verwendet man in der entsprechenden Formelzeile das Kommando `\nonumber` .

Beispiel 2 (ohne Nummerierung)

```
Gegeben sei das Gleichungssystem
\begin{eqnarray*}
   x + y & = & 10 \\
   x^2 + y^2 + z^2 & = & 30  \\
   2xy + xyz + 4 z & = & 90
\end{eqnarray*}
```

Gegeben sei das Gleichungssystem

$$\begin{aligned} x + y &= 10\\ x^2 + y^2 + z^2 &= 30\\ 2xy + xyz + 4z &= 90 \end{aligned}$$

Anmerkung: Wenn Sie das `amsmath`-Paket in der Präambel eingebunden haben, so können Sie für mehrzeilige Formeln auch die `align`-Umgebung verwenden. Diese liefert in einigen Fällen ein besseres Layout.

Beispiel (ohne Nummerierung)

```
Gegeben sei das Gleichungssystem
\begin{align*}
   x + y & = 10 \\
   x^2 + y^2 + z^2 & = 30  \\
   2xy + xyz + 4 z & =  90
\end{align*}
```

Gegeben sei das Gleichungssystem

$$\begin{aligned} x + y &= 10 \\ x^2 + y^2 + z^2 &= 30 \\ 2xy + xyz + 4z &= 90 \end{aligned}$$

2 Einführung im Matlab

2.1 Grundlagen

2.1.1 Was ist Matlab

Matlab ist ein kommerzielles (daher teures) Softwaresystem für numerische Berechnungen, welches in den MINT-Fächern weit verbreitet ist. Die Grundausstattung (nur darauf bezieht sich diese Einführung) kann durch zahlreiche Zusatzpakete (Toolboxen) erweitert werden, z.B. PDE-Toolbox, Optimization Toolbox, Financial Toolbox.

Der Name Matlab stammt von **Mat**rix **lab**oratory und deutet schon die Philosophie an: Matrizen bilden die Grundelemente. Reelle Zahlen werden als 1×1-Matrizen aufgefasst; ebenso sind Vektoren spezielle Matrizen. Aus diesem Grund sind Kenntnisse in der elementaren Matrizenrechnung erforderlich.

Matlab arbeitet interaktiv, d.h. die eingegebenen Kommandos werden sofort ausgeführt (Interpreter), hat aber alle Elemente einer modernen Programmiersprache.

2.1.2 Matlab starten

Vor dem erstmaligen Aufruf von Matlab empfehlen wir, sich einen eigenen Ordner (z.B. mit dem Namen `matlab`) anzulegen und in diesen vor dem Aufruf zu wechseln.

In einem kommandoorientierten Fenster wird Matlab durch den Befehl

`matlab` bzw. `matlab &`

gestartet. Dabei bewirkt das `&` - Zeichen, dass Matlab in einem neuen, unabhängigen Fenster (d.h. im Hintergrund) läuft. Das Fenster, aus dem Matlab gestartet wurde, steht für andere Aufgaben weiter zur Verfügung. Wird das `&` - Zeichen weggelassen, so ist dieses Fenster so lange blockiert, bis Matlab wieder beendet wird.

Eventuell besitzt die graphische Oberfläche Ihres Rechners ein Matlab-Icon. Dann können Sie Matlab durch Anklicken dieses Symbols starten.

2.1.3 Das Matlab-Fenster

Nach dem Starten erhalten wir ein neues Fenster (siehe unten), welches aus drei Teilfenstern (Voreinstellung) besteht. Das momentan aktive Unterfenster ist mit einem blauen Balken am oberen Rand markiert. Die Größe der Unterfenster lassen sich den individuellen Wünschen anpassen. Dazu klicken Sie mit der linken Maustaste auf den Rand zwischen den Fenstern und verschieben diesen (bei gedrückter linker Maustaste).

Für uns interessant ist zunächst das Unterfenster mit der Überschrift

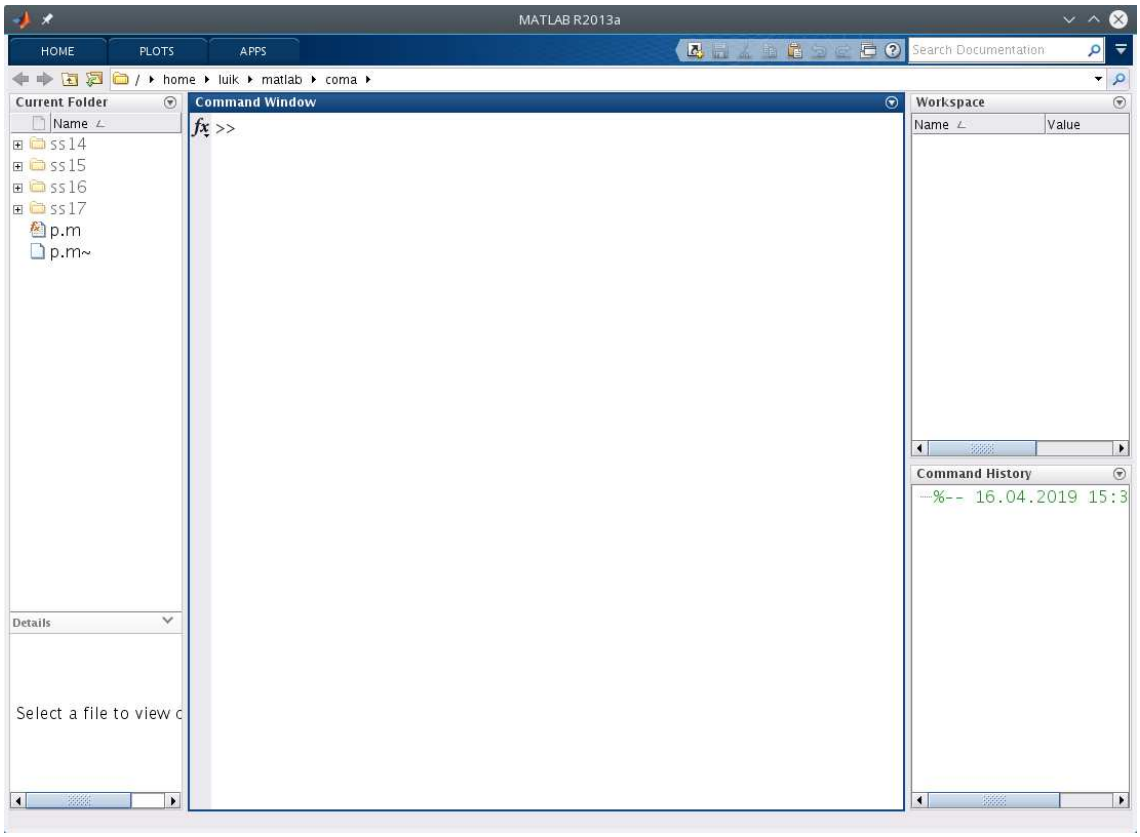
Command Window

Hier geben Sie die Matlab-Befehle ein. Auch die Ergebnisse werden in diesem Fenster angezeigt.

Alle eingegebenen Matlab-Kommandos werden im Fenster

Command History

gespeichert und können von dort wieder in das Command Window kopiert werden. Dies gilt auch für die Matlab-Befehle einer früheren Sitzung.



2.1.4 Matlab beenden

Um Matlab zu beenden, klicken Sie mit der linken Maustaste im MATLAB-Fenster das Symbol links oben an und wählen dann `schließen` aus.

Alternativ können Sie im Command Window den Befehl

```
exit
```

eingeben.

Beim Verlassen wird die aktuelle Matlab-Konfiguration (Größe und Anzahl der Unterfenster) gespeichert und erscheint so beim nächsten Matlab-Aufruf wieder.

2.1.5 Online-Hilfe

Eine komplette Dokumentation von Matlab (in englischer Sprache) befindet sich auf dem Rechner. Diese wird gestartet durch Eingabe von

```
doc
```

im Command Window. Es wird dann ein neues Fenster **Help** eröffnet.

Anmerkung: Ältere Matlab-Versionen verwenden statt `doc` das Kommando `helpdesk`.

Ist man nur an Informationen zu einem bestimmten Matlab-Befehl interessiert, so gibt es zwei Möglichkeiten:

1. Die Eingabe von

   `doc` <*kommando*>

   im Command Window liefert ausführliche Information in einem neuen Help-Window. Dabei bedeutet <*kommando*> ein Matlab-Kommando. Z.B. erhalten Sie durch

   ```
   doc inv
   ```

   Information über das Matlab-Kommando `inv`, welches die inverse Matrix berechnet.

2. Das Kommando

   `help` <*kommando*>

   gibt die Information im Command Window aus. So erzeugt

   ```
   help inv
   ```

   die folgende Ausgabe:

```
>> help inv

 INV    Matrix inverse.
    INV(X) is the inverse of the square matrix X.
    A warning message is printed if X is badly scaled or
    nearly singular.

    See also SLASH, PINV, COND, CONDEST, LSQNONNEG, LSCOV.
```

Mit dem Befehl

`lookfor` <*schlüsselwort*>

wird nach (englischen) Schlüsselwörtern gesucht. So liefert beispeilsweise

```
lookfor sine
```

die folgenden Informationen:

```
COS    Inverse cosine.
ACOSH  Inverse hyperbolic cosine.
ASIN   Inverse sine.
ASINH  Inverse hyperbolic sine.
COS    Cosine.
COSH   Hyperbolic cosine.
SIN    Sine.
SINH   Hyperbolic sine.
TFFUNC time and frequency domain versions of a cosine
       modulated Gaussian pulse.
DST    Discrete sine transform.
IDST   Inverse discrete sine transform.
```

2.1.6 Eingabe von Matlab-Kommandos

Die Eingabe von Matlab-Kommandos erfolgt im Command Window. Matlab arbeitet interaktiv, d.h. jeder Matlab-Befehl wird nach Betätigen der Eingabetaste sofort ausge-

führt (im Gegensatz zu den Programmiersprachen wie Fortran, Pascal oder C, bei denen das komplette Programm zuerst übersetzt werden muss).

Dabei ist zu beachten:

- Es wird zwischen Groß- und Kleinbuchstaben unterschieden.
- Das Zeichen % dient als Kommentarzeichen: Der Text rechts davon bis zum Zeilenende wird als Kommentar betrachtet (und somit nicht als Matlab-Befehl interpretiert).
- Ein Matlab-Befehl kann sich über mehrere Zeilen erstrecken, muss dann jedoch am Zeilenende mit dem Zeichen ... markiert werden.
- In einer Zeile dürfen mehrere Matlab-Befehle stehen. Diese müssen jedoch durch einen Strichpunkt (bzw. ein Komma) getrennt werden.
- Ein Matlab-Kommando kann mit einem Strichpunkt abgeschlossen werden. Dann wird das Ergebnis dieses Befehls nicht im Command Window angezeigt. Ohne Strichpunkt am Ende erscheint das Ergebnis im Command Window.
- In Matlab können auch Linux-Kommandos eingegeben werden. Diese werden mit einem vorangestellten ! gekennzeichnet. So wird durch

  ```
  !ls -al
  ```

  der Inhalt des aktuellen Verzeichnisses im Command Window aufgelistet.

Beispiele

```
x = 1.5
X = 1e-7;
a = 1.5; b = 4.5; c = 3.8; s = (a+b+c)/3; % Mittelwert
u = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
      -1/8 + 1/9 - 1/10 + 1/11 - 1/12;
u
```

`X` und `x` sind verschiedene Variablen. Die letzte Zeile bedeutet, dass der Wert der Variablen `u` im Command Window ausgegeben wird.
Mit dem Kommando

```
clc
```

werden alle Einträge im Command Window gelöscht.

2.1.7 Matlab-Programme

Es besteht die Möglichkeit, mehrere Matlab-Kommandos in eine Datei zu schreiben (mit Hilfe eines Editors). Diese muss den Zusatz `.m` haben und wird als *Matlab-Skript* bezeichnet. Es gibt zwei Möglichkeiten:

1. **Verwendung eines (Linux-)Editors (z.B. emacs, kile)**
   Dies hat den Vorteil, dass ein Matlab-Skript erstellt werden kann, ohne Matlab selbst gestartet zu haben. Sie können Ihr Programm zu Hause eingeben und im Rechner-Pool dann unter Matlab ausführen.

2. **Verwendung des Matlab-Editors**
   Matlab besitzt einen eigenen Editor. Dieser wird aktiviert durch Anklicken (mit der linken Maustaste) von **HOME** und dann **New Script**. Dadurch wird ein neues Fenster eröffnet, und wir können nun das Programm eintippen.

   Soll ein bereits existierendes Matlab-Skript geladen werden, so klicken wir statt **New Script** den Menüpunkt **Open** an und wählen dann den Dateinamen aus.

   Der Matlab-Editor kann auch im Kommand Window durch Eingabe von

   `edit` bzw. `edit` <*datei*>

   gestartet werden. Er funktioniert ähnlich wie Emacs, zusätzlich werden jedoch gewisse Matlab-Befehle farbig dargestellt (*Syntax Highlighting*).

   Ferner ist der Matlab-Editor mit einem *Debugger* kombiniert. Ein Debugger dient dazu, in einem Programm die einzelnen Anweisungen Schritt für Schritt zu verfolgen und Zwischenergebnisse anzuschauen. Damit hat man ein nützliches Mittel bei der Fehlersuche.

Beispiel: Wir geben in die Datei `beispiel1.m` folgende Matlab-Kommandos ein:

```
% Es werden Messergebnisse gezeichnet.
x = [0.333 0.167 0.0833 0.0416 0.0208 0.0104 0.0052];
y = [3.636 3.636 3.236 2.666 2.114 1.466 0.866];
plot(x,y,'+');
xlabel(' Konzentration c','Fontsize',15);
ylabel(' Geschwindigkeit \nu','Fontsize',15);
title('Messergebnisse','Fontsize',15);
```

Dieses Matlab-Programm wird durch Eingabe von

```
beispiel1
```

gestartet (im Command Window). Die einzelnen Matlab-Kommandos werden später erläutert.

2.1.8 Der Matlab-Pfad

Beim Start eines Matlab-Kommandos durchsucht Matlab gewisse Verzeichnisse (Ordner) nach den entsprechenden Matlab-Dateien (im obigen Beispiel nach `beispiel1.m`). Der Matlab-Pfad (matlab search path) legt fest, welche Verzeichnisse (und in welcher Reihenfolge) durchsucht werden. Wird in diesen Ordnern die gesuchte Datei nicht gefunden, so erhält man eine Fehlermeldung.

Der Anwender hat die Möglichkeit, diesen Matlab-Pfad an seine Bedürfnisse anzupassen, z.B. weitere Verzeichnisse aufzunehmen. Dazu wird im Matlab-Fenster

HOME $\longrightarrow$ **Set Path**

angeklickt. Dadurch wird ein weiteres Fenster mit der Überschrift **Set Path** geöffnet, in welchem der bisherige Matlab-Pfad angezeigt wird. Hier können nun die gewünschten Änderungen vorgenommen werden.

2.1.9 Matlab Workspace

Matlab besitzt einen Arbeitsspeicher (*Matlab Workspace*), in dem alle während einer Sitzung erzeugten Variablen mit ihrem zuletzt zugewiesenen Wert abgelegt werden. Den Workspace kann man sich in einem Matlab-Unterfenster anzeigen lassen. Dazu klicken Sie mit der linken Maustaste auf das Feld **Workspace**.

Durch die Matlab-Kommandos `who` bzw. `whos` erhält man die Informationen auch im Command Window.

Beispiel: Nachdem wir das Programm `beispiel1` ausgeführt haben, liefert das Kommando `whos` folgende Informationen:

```
>> whos
   Name      Size          Bytes  Class

   x         1x7              56  double array
   y         1x7              56  double array

   Grand total is 14 elements using 112 bytes
 >>
```

Inhalt des Workspaces speichern
In dem Menüpunkt **HOME** wird der Unterpunkt **Save Workspace As...** angeklickt und danach der Dateiname eingegeben. Die Datei erhält automatisch den Zusatz `.mat`.

Alternativ kann im Command Window der Befehl `save` verwendet werden. So wird zum Beispiel durch das Kommando

`save('april25')`

der Inhalt des Arbeitsspeichers in die Datei namens `april25.mat` kopiert. Beachten Sie, dass im save-Kommando der Name in Hochkommata eingeschlossen werden muss.

Laden in den Workspace
In dem Menüpunkt **HOME** wird der Unterpunkt **Open** angeklickt und danach der Dateiname eingegeben. Dies muss eine Datei mit dem Zusatz `.mat` sein.

Alternativ kann im Command Window der Befehl `load` verwendet werden. So wird zum Beispiel durch das Kommando

`load('april25')`

die Datei `april25.mat` in den Arbeitsspeicher geladen.

Workspace löschen
Der gesamte Workspace wird gelöscht durch Anklicken von **HOME** und danach **Clear Workspace**. Matlab fragt dann nochmals nach: **Are you sure you want to clear your workspace?** Erst wenn Sie dann **yes** anklicken, wird der Arbeitsspeicher gelöscht.

Alternativ können Sie im Command Window mit Befehl dem

`clear all` bzw. `clear` <name>

den gesamten Arbeitsspeicher bzw. einzelne Variablen löschen.

2.2 Umgang mit Matrizen

Matlab arbeitet stets mit dem Datentyp Matrix (engl. *array*). Daher stammt auch der Name Matlab: **Mat**rix **lab**oratory. Eine reelle Zahl ist dann eine 1×1 - Matrix, ein Zeilenvektor eine $1 \times n$ - Matrix und ein Spaltenvektor eine $n \times 1$ - Matrix. Intern werden die Matrixelemente als reelle Zahlen mit doppelter Genauigkeit dargestellt.

2.2.1 Matrix belegen

Eine Matrix wird zeilenweise eingegeben; jede Zeile wird mit einem Strichpunkt abgeschlossen, die einzelnen Elemente der Zeile werden durch Leerzeichen oder Kommata getrennt. So wird zum Beispiel durch

```
A = [1 2 3 4; 5 6 7 8; 9 10 12 12; 13 14 15 16]
```

die Matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \tag{5}$$

definiert. Beachten Sie die eckigen Klammern. Durch

```
x = [1 2 3 4]
y = [10; 20; 30]
```

werden die Vektoren

$$x = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix} \quad \text{und} \quad y = \begin{pmatrix} 10 \\ 20 \\ 30 \end{pmatrix}$$

erzeugt. Auf ein Matrix-Element wird durch Angabe seiner Position (Zeilenindex, Spaltenindex) zugegriffen. Zum Beispiel erhalten wir durch Eingabe von

```
A(3,2) = 0
```

die Matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 0 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \quad .$$

Eine Matrix kann mit Nullen bzw. Einsen vorbelegt werden. Durch

```
A = zeros(4,3)
B = ones(2,5)
```

erhalten wir die Matrizen

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{und} \quad B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad .$$

Die $n \times n$ - Einheitsmatrix wird in Matlab durch das Kommando `eye(n,n)` erzeugt. Dabei muss `n` schon mit einem Wert belegt sein. So liefert

```
n = 3
I = eye(n,n)
```

die Matrix

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad .$$

Das Matlab-Kommando

```
x = 0:0.1:2
```

erzeugt den Zeilenvektor $x = (0, 0.1, 0.2, \ldots, 1.9, 2.0)$. Ferner liefert

```
linspace(a,b,n)
```

n äquidistante Punkte aus dem Intervall $[a, b]$.

2.2.2 Teilbereiche einer Matrix

Von einer bereits definierten Matrix kann man Teilbereiche (Zeilen, Spalten, Untermatrizen) ansprechen. Für die Matrix A aus (5) liefern die Kommandos

```
u = A(2,:)
v = A(:,3)
B = A(1:3,1:2)
C = A(3:4,3:4)
```

die folgenden Vektoren bzw. Matrizen:

$$u = \begin{pmatrix} 5 & 6 & 7 & 8 \end{pmatrix}, \quad v = \begin{pmatrix} 3 \\ 7 \\ 11 \\ 15 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{pmatrix}, \quad C = \begin{pmatrix} 11 & 12 \\ 15 & 16 \end{pmatrix} \quad .$$

Eine Matrix kann auch mit Hilfe von Teilmatrizen definiert werden. So erzeugt zum Beispiel die Sequenz

```
I = eye(2,2);
Z = zeros(2,2);
E = [2 -1; -1 2];
D = [E,Z,I; Z,E,Z; I,Z,E]
```

die Matrix

$$D = \begin{pmatrix} 2 & -1 & 0 & 0 & 1 & 0 \\ -1 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 & -1 \\ 0 & 1 & 0 & 0 & -1 & 2 \end{pmatrix} \quad .$$

Aus einer Matrix können Teilbereiche gestrichen werden. Dadurch ändert sich die Größe der Matrix. Betrachten wir die Matrix A aus (5) und geben dann

```
A(2,:) = []
```

ein, so wird aus A die zweite Zeile gestrichen:

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \quad .$$

Ebenso können Zeilen bzw. Spalten in eine Matrix eingefügt werden. Beispielsweise ergibt

```
A = [A(1:2,:); 21 22 23 24; A(3,:)]
```

die Matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 9 & 10 & 11 & 12 \\ 21 & 22 & 23 & 24 \\ 13 & 14 & 15 & 16 \end{pmatrix} \quad .$$

2.2.3 Der Array-Editor

Die während einer Matlab-Sitzung erzeugten Variablen (Matrizen) können mit dem **Array Editor** betrachtet und verändert werden.

Für den Aufruf gibt es zwei Möglichkeiten:

1. Im Unterfenster **Workspace** wird die entsprechende Variable angewählt. Durch Doppelklick mit der linken Maustaste wird ein neues Fenster geöffnet, in dem der Inhalt der Variablen angezeigt wird.

2. Im **Command Window** wird der Befehl

   ```
   openvar('<variable>')
   ```

   eingegeben, zum Beispiel

   ```
   openvar('A')
   ```

Nun können die einzelnen Matrixelemente bearbeitet oder neue Zeilen bzw. Spalten hinzugefügt werden.

2.2.4 Operationen mit Matrizen

Zwei Matrizen A und B derselben Größe werden addiert bzw. subtrahiert durch

```
C = A + B
D = A - B
```

Diese Operationen werden (wie aus der Mathematik bekannt) elementweise durchgeführt. Vorsicht ist dagegen bei der Multiplikation geboten, denn hier kennt die Mathematik zwei Möglichkeiten: elementweise Multiplikation und das Matrixprodukt. Dementsprechend gibt es auch in Matlab die beiden Möglichkeiten

`C = A * B`	Matrixprodukt
`D = A .* B`	elementweise Multiplikation

Das Matrixprodukt ist nur definiert, falls A eine $m \times n$-Matrix und B eine $n \times r$-Matrix ist und ergibt dann eine $m \times r$-Matrix.

Entsprechendes gilt bei der Potenz:

`C = A^3`	entspricht `C = A*A*A`
`D = A.^3`	elementweise Potenzierung

Die Matrixpotenz A^3 ist nur für eine quadratische Matrix (d.h. eine $n \times n$-Matrix) erklärt.

Die elementweise Division zweier Matrizen erfolgt mittels

```
C = A./B
```

,

was $c_{ij} = a_{ij}/b_{ij}$ bedeutet. Für Matrizen kennt Matlab die Operationen

`D = A/B` (hier ist D die Matrix, welche $A = D \cdot B$ erfüllt),

`x = A\b` (dann ist x die Lösung des lin. Gleichungssystems $Ax = b$).

Möglich sind auch die Anweisungen

```
E1 = A .* 2;
E2 = 2 ./ A;
E3 = A + 2;
E4 = A - 2;
```

Gedanklich wird bei `E3` bzw. `E4` die reelle Zahl 2 zu einer Matrix, welche dieselbe Größe wie A hat, und deren Elemente alle den Wert 2 haben.

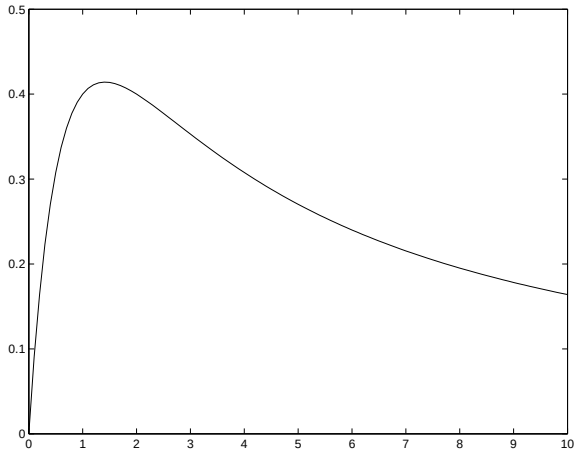
Beispiel: Für die Funktion

$$f(x) := \frac{x}{1 + ax + rx^2}$$

erstellen wir mit $a = 1,\ r = 0.5$ eine Wertetabelle. Anschließend wird diese Funktion gezeichnet. Die entsprechenden Matlab-Befehle lauten

```
a = 1;
r = 0.5;
x = 0:0.1:10;
y = x./(1 + a.*x +r.*x.^2);
plot(x,y);
```

Als Ergebnis erhalten wir die folgende Grafik:



Das Matlab-Kommando `plot` zeichnet die Wertetabelle. Auf diesen Befehl gehen wir später noch ausführlich ein.

2.2.5 Funktionen mit Matrizen

Ist A eine bereits definierte Matrix, so liefert

`size(A)` bzw. `[m,n] = size(A)`

die Größe der Matrix; `m` enthält die Zeilenzahl und `n` die Spaltenzahl. Durch

`B = zeros(size(A))`

wird eine mit Nullen vorbelegte Matrix B erzeugt, die dieselbe Größe wie A hat.

Die Länge eines Vektor x erhält man durch den Befehl

```
length(x)
```
.

Für eine Matrix A liefert das Kommando

```
diag(A)
```

die Diagonale (als Spaltenvektor). Umgekehrt ergibt für einen Spaltenvektor v (mit n Komponenten) der Matlab-Befehl

```
B = diag(v)
```

eine Diagonalmatrix mit den Elementen von v in der Diagonalen. Weiter liefert

```
C = diag(v,1)
```

eine $(n+1)\times(n+1)$-Matix C mit dem Vektor v in der ersten oberen Nebendiagonalen. Zum Beispiel erhalten wir durch

```
e = ones(4,1);
S = diag(11:11:55) + diag(e,1) + diag(e,-1);
```

die Matrix

$$\begin{pmatrix} 11 & 1 & 0 & 0 & 0 \\ 1 & 22 & 1 & 0 & 0 \\ 0 & 1 & 33 & 1 & 0 \\ 0 & 0 & 1 & 44 & 1 \\ 0 & 0 & 0 & 1 & 55 \end{pmatrix} .$$

Durch

```
sum(A)
```

werden die Elemente in den Spalten aufsummiert (ergibt als Ergebnis einen Zeilenvektor), während

```
sum(A,2)
```

die Elemente zeilenweise addiert. Schließlich erhält man durch

```
B = A'
```

die zu A transponierte Matrix. Möglich sind auch

`u = sum(diag(A));`	(summiert die Diagonalelemente, ist die Spur von A)
`z = sum(A(:,3);`	(summiert die Elemente der 3. Spalte)
`v = sum(sum(A));`	(summiert über alle Elemente der Matrix A)
`D = diag(diag(A));`	(Diagonalmatrix)

Für einen Vektor $x = (x_1, \ldots, x_n)$ kennt Matlab die Funktionen

`norm(x) = norm(x,2)`	$\left(\|x\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}\right)$
`norm(x,1)`	$(\|x\|_1 = \|x_1\| + \cdots + \|x_n\|)$
`norm(x,inf)`	$(\|x\|_\infty = \max\{\|x_i\| \,:\, i = 1, \ldots, n\})$
`[s,j] = max(x)`	$(= \max\{x_i : i = 1, \ldots, n\}\ )$
`[s,j] = min(x)`	$(= \min\{x_i : i = 1, \ldots, n\}\ )$

Dann liefert `s` den maximalen (minimalen) Wert und `j` die zugehörige Komponente.

Es gibt auch einige Konstanten, so z.B.

`pi` für die Kreiszahl π,

`eps` für die relative Rechengenauigkeit ε (vgl. Abschnitt 3.2)

Matlab kennt alle wichtigen (aus der Mathematik bekannten) Funktionen, welche durch das Kommando

```
help elfun
```

aufgelistet werden. Die wichtigsten davon sind:

`exp(a)`	Exponentialfunktion
`log(a)`	natürlicher Logarithmus
`log10(a)`	Logarithmus zur Basis 10
`log2(a)`	Logarithmus zur Basis 2
`sqrt(a)`	Quadratwurzel
`nthroot(a,n)`	n-te Wurzel von a
`sin(a)`	Sinus
`cos(a)`	Kosinus
`tan(a)`	Tangens
`ceil(a)`	nächste ganze Zahl $\geq a$
`floor(a)`	nächste ganze Zahl $\leq a$
`round(a)`	rundet zur nächsten ganzen Zahl
`fix(a)`	$\begin{cases} \texttt{ceil(a)} & \text{falls } a \geq 0 \\ \texttt{floor(a)} & \text{falls } a \leq 0 \end{cases}$

Die Variable darf natürlich auch eine Matrix sein. Dann liefern diese Funktionen als Ergebnis eine Matrix derselben Größe. Die Funktionsauswertung erfolgt elementweise.

Beispiel: Wir erstellen von der Funktion

$$f(x) := \frac{a}{1+\exp(b-cx)} \qquad \text{mit } a,b,c \in \mathbb{R}$$

eine Wertetabelle für $x = 0, 0.1, 0.2, \ldots, 9.9, 10$ und zeichnen diese anschließend.

```
a=100;
b=5;
c=1
x=0:0.1:10;
f = a./(1+exp(b - c.*x));
plot(x,f);
```

Hier ist `f` ein Zeilenvektor, der dieselbe Größe wie `x` hat. Seine Komponenten sind die Funktionswerte $f(x_i)$.

2.3 Ein- und Ausgabe

2.3.1 Einlesen aus einer Datei

Zum Einlesen von Größen aus einer Datei bietet Matlab mehrere Möglichkeiten. Diese sind von der Form der Eingabedatei abhängig: nur Zahlen (numerische Größen) oder Text und Zahlen gemischt (alphanumerische Größen).

Numerische Größen

Enthält die einzulesende Datei nur numerische Größen, und zwar in jeder Zeile die gleiche Anzahl von Zahlen (durch Leerzeichen getrennt), so steht uns der Matlab-Befehl

`A = load(' <name> ')`

zur Verfügung. Dabei ist dann `A` eine Matrix. Befinden sich zum Beispiel in der Datei `daten1.ein` die Zahlen

```
1.1 1.2 1.3 1.4 1.5
2.1 2.2 2.3 2.4 2.5
3.1 3.2 3.3 3.4 3.5
4.1 4.2 4.3 4.4 4.5
```

so liefert das Kommanodo

```
A = load('daten1.ein')
```

eine 4×5 - Matrix. Der `load` - Befehl liefert eine Fehlermeldung, falls in den Zeilen unterschiedlich viele Zahlen stehen.

Text und numerische Größen

Im Normalfall hat man eine Eingabedatei, die sowohl Text als auch Zahlen enthält. Häufig sind auch die einzelnen Zeilen unterschiedlich lang. Dann verwendet man zum Einlesen die Matlab-Kommandos `fgetl` (für reine Textzeilen) bzw. `fscanf`.

Dazu muss aber zuerst die Datei mittels

`fid = fopen('<dateiname>')`

zum Lesen angemeldet werden. Durch diesen Befehl wird die Variable `fid` (die natürlich frei wählbar ist) der entsprechenden Datei zugeordnet.

Das Kommando

```
zeile1 = fgetl(fid)
```

liest nun eine Text-Zeile aus der angemeldeten Datei ein und speichert sie in der Variablen `zeile1`. Numerische Größen werden mit dem Matlab-Befehl `fscanf` eingelesen. Er hat die Form (*Syntax*)

`B = fscanf(fid,`<format>, <größe>`)` .

So bewirkt zum Beispiel das Kommando

```
B = fscanf(fid,'%g',[3,2])   ,
```

dass aus der Datei die nächsten 6 Zahlen eingelesen und in einer 3×2 - Matrix gespeichert werden. Dabei ist zu beachten, dass die Datei **zeilenweise** gelesen, die Matrix aber **spaltenweise** belegt wird.

Der Befehl

```
D = fscanf(fid,'%c',[1,10])
```

liest die nächsten 10 Zeichen (*characters*) ein.

Nach dem Einlesen wird die Datei mit dem Befehl

```
fclose(fid)
```

wieder abgemeldet.

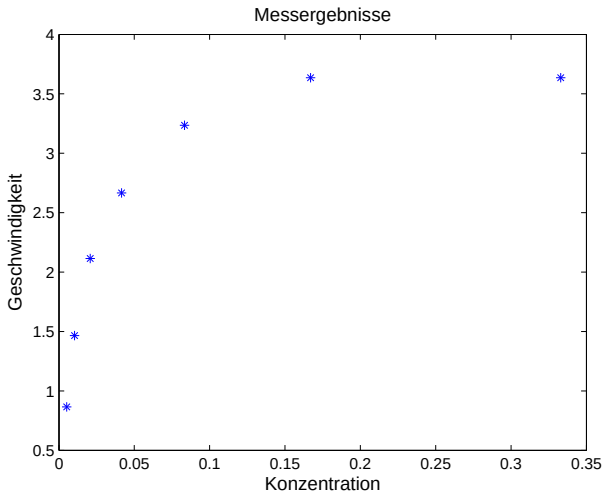
Beispiel: Die Datei `ergebnisse.dat` habe folgende Gestalt:

```
Messergebnisse
Konzentration
Geschwindigkeit
0.333 3.636
0.167 3.636
0.0833 3.235
0.0416 2.666
0.0208 2.114
0.0104 1.466
0.0052 0.866
```

Das folgende Matlab-Programm liest diese Datei ein und erstellt dann eine Zeichnung. Dabei wird der Text für die Überschrift bzw. die Achsenbeschriftung verwendet.

```
fid = fopen('ergebnisse.dat');
zeile1 = fgetl(fid);
zeile2 = fgetl(fid);
zeile3 = fgetl(fid);
Z = fscanf(fid,'%g',[2,7]);
plot(Z(1,:),Z(2,:),'*');
title(zeile1);
xlabel(zeile2);
ylabel(zeile3);
fclose(fid);
```

Als Ergebnis erhalten wir die folgende Grafik:



Manchmal besteht der Wunsch, alle Werte aus einer Datei einzulesen, wobei jedoch die Anzahl der Daten nicht bekannt ist. Nehmen wir mal an, dass die Datei `test.dat` folgende Gestalt hat:

```
1 2 3 4 5 6 7
8 9 10 11 12
13 14 15 16
17 18
19
20 21
```

Die Matlab-Befehle

```
fid = fopen('test.dat');
[x,count] = fscanf(fid,'%g',[1,inf])
fclose(fid);
```

liefern dann als Ergebnisse

```
x =
  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21

count =
      21
```

Die Angabe von `inf` beim Aufruf von `fscanf` bewirkt, dass alle Zahlen gelesen werden. Die Anzahl der Werte wird in der Variablen `count` gespeichert.

2.3.2 Ausgabe in eine Datei

Auch zur Ausgabe von Matlab-Größen in eine Datei gibt es mehrere Möglichkeiten.

Numerische Größen

Zur Ausgabe von rein numerischen Größen in eine Datei steht uns das `save` - Kommando zur Verfügung. Haben wir zum Beispiel eine Matrix

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

definiert, so wird diese mittels

```
save('matrix.dat','-ASCII','A');
```

in eine Datei namens `matrix.dat` gespeichert. Der Zusatz `'-ASCII'` bewirkt die Abspeicherung im Standartformat (Ascii-Format); die Datei kann dann mit jedem gängigen Texteditor gelesen werden.

Im obigen Fall erhalten wir durch `emacs matrix.dat` die Ausgabe

```
1.0000000e+00   2.0000000e+00   3.0000000e+00   4.0000000e+00
5.0000000e+00   6.0000000e+00   7.0000000e+00   8.0000000e+00
```

Text und numerische Größen

Hierfür steht uns das Matlab-Kommando `fprintf` zur Verfügung, welches folgenden Aufbau besitzt:

`fprintf(fid,` <*format*> `,` <*variable*> `);` .

Dabei muss zuerst mit dem `fopen` - Befehl der Variablen `fid` eine Datei zugeordnet werden. Für <*format*> müssen wir eine Formatangabe einsetzen; hier wird angegeben, wie die Variablen in der Datei dargestellt werden sollen. Möglich ist unter anderem:

%d oder %i	signed integer (ganze Zahl)
%u	unsigned integer (natürliche Zahl)
%e	Gleitpunktdarstellung, z.B 1.105171e+01
%f	Festkommadarstellung, z.B. 11.05171
%g	wählt automatisch die günstigste Darstellung
%12.6f	Festkommadarstellung mit insgesamt 12 Stellen, davon 6 Nachkommastellen
%14.8e	Gleitkommadarstellung mit insgesamt 14 Stellen, davon 8 Nachkommastellen
%s	String-Variable (Text)
%c	ein einzelnes Zeichen (character)
\n	neue Zeile

Außerdem kann die Formatangabe noch Text enthalten, der dann ebenfalls in die Datei geschrieben wird. Die Formatangabe wird in Hochkommata eingeschlossen.

Beispiel: Es wird eine Wertetabelle von der Exponentialfunktion erstellt und in der Datei `wertetab.txt` gespeichert.

```
clear all;
x = 0:0.1:1;
y = [x; exp(x)];
fid = fopen('wertetab.txt','w');
fprintf(fid,'Exponentialfunktion\n\n');
fprintf(fid,'%5.2f  %12.8f\n',y);
fclose(fid);
```

Danach hat die Datei `wertetab.txt` den folgenden Inhalt:

```
Exponentialfunktion

0.00    1.00000000
0.10    1.10517092
0.20    1.22140276
0.30    1.34985881
0.40    1.49182470
0.50    1.64872127
0.60    1.82211880
0.70    2.01375271
0.80    2.22554093
0.90    2.45960311
1.00    2.71828183
```

Falls beim Aufruf von `fid = fopen('wertetab.txt','w')` die Datei `wertetab.txt` schon existiert, so wird deren Inhalt zunächst gelöscht. Möchte man die Ergebnisse an die bereits bestehende Datei `wertetab.txt` anhängen, so wird

```
fid = fopen('wertetab.txt','a');
```

verwendet.

2.3.3 Ausgabe auf dem Bildschirm

Falls man bei einem Matlab-Befehl das Semicolon weglässt, so erscheint das Ergebnis sofort auf dem Bildschirm. Mit Hilfe des `format` - Befehls lässt sich die Zahlendarstellug

ändern. Es gibt folgende Möglichkeiten:

`format short`	Festkommadarstellung mit 5 Stellen
`format long`	Festkommadarstellung mit 15 Stellen
`format short e`	Gleitkommadarstellung mit 5 Stellen
`format long e`	Gleitkommadarstellung mit 15
`format compact`	unterdrückt Leerzeilen

Zwei weitere Möglichkeiten bilden die Matlab-Kommandos `disp` und `sprintf`. Dabei entspricht `sprintf` dem in Abschnitt 2.3.2 behandelten Matlab-Befehl `fprintf`, das Ergebnis wird jedoch in eine String-Variable (Text-Variable) gespeichert. Mit dem `disp` - Befehl kann sowohl eine Variable als auch Text ausgegeben werden.

Beispiel:

```
A = [1 2 3 4; 4 5 6 7];
disp('Eingegeben wurde die Matrix A =');
disp(A);
text = sprintf('   A(2,3) = %6.2f',A(2,3));
disp(text);
```

Als Ergebnis erhalten wir auf dem Bildschirm die Ausgabe

```
Eingegeben wurde die Matrix A =
     1     2     3     4
     4     5     6     7

    A(2,3) =   6.00
```

2.3.4 Eingabe über den Bildschirm

Mit dem Befehl `input` können innerhalb eines Matlab-Programmes Daten über den Bildschirm eingelesen werden. Er hat die Form

```
x = input('Text');     ,
c = input('Text','s');     .
```

Matlab gibt dann auf dem Bildschirm den Text aus und wartet auf die Eingabe. Im ersten Fall wird eine numerische Eingabe erwartet, welche in der Variablen `x` gespeichert wird; im zweiten Fall wird die Eingabe als String (Text) behandelt und in `c` abgelegt.

Beispiel: Es wird eine Wertetabelle von der logistischen Kurve

$$L(t) := \frac{a}{1+\exp(b-ct)} \quad ,$$

erstellt, wobei die Parameter a, b, c über den Bildschirm eingelesen werden.

```
clear all
disp(' Es wird eine Wertetabelle der log. Kurve erstellt');
disp(' Geben Sie die folgenden Parameter ein:');
a = input(' a = ');
b = input(' b = ');
c = input(' c = ');
```

```
x=0:0.5:50;
y=[x; a./(1.0 + exp(b -c.*x))];
fid = fopen('wertetab.txt','w');
fprintf(fid,' logistische Kurve mit den Parametern\n');
fprintf(fid,'    a = %g\n',a);
fprintf(fid,'    b = %g\n',b);
fprintf(fid,'    c = %g\n\n',c);
fprintf(fid,'   %6.2f   %9.4f \n',y);
fclose(fid);
```

Anmerkung: Im obigen Programm erfolgt keine Ausgabe auf dem Bildschirm. Deshalb sollte man am Ende noch den Befehl

```
disp('Ergebnisse stehen in der Datei wertetab.txt');
```

aufnehmen. Daran erkennt dann der Anwender, dass das Programm beendet ist und wo die Ergebnisse nachzulesen sind.

2.4 Graphik

Matlab besitzt ein umfangreiches Graphik-Paket. Durch die Eingabe des ersten Graphik-Befehls wird ein neues Matlab-Fenster mit der Überschrift `Figure` geöffnet. Eine erstellte Graphik kann in verschiedenen Formaten (z.B. Postskript, JPEG, PDF) abgespeichert und so in andere Texte (z.B. Latex-Dokumente) eingebunden werden.

2.4.1 2D-Graphik

Zum Zeichnen von reellen Funktionen (oder Messergebnissen) in der Ebenen verwendet man das Matlab-Kommando `plot`. Dabei ist zunächst eine Wertetabelle von der Funktion zu erstellen, welche als Parameter an das `plot` - Kommando übergeben wird. Das folgende Beispiel zeichnet die Funktion $\sin(x)$ im Intervall $[0, 10]$:

```
x = 0:0.1:10;
y = sin(x);
plot(x,y);
```

Jeder Aufruf von `plot` löscht standardmäßig eine bereits vorhandene Kurve. Um dies zu vermeiden, gibt man vor dem `plot` - Aufruf das Kommando

```
hold on
```

ein. Dadurch wird die Kurve in die vorhandene Graphik eingefügt. Mit einem `plot` - Kommando können auch mehrere Kurven gleichzeitig gezeichnet werden. Es gibt zwei Möglichkeiten, um zwei (bzw. mehrere) Kurven in ein Schaubild zu zeichnen:

1. Möglichkeit

```
x = 0:0.1:10;
y = sin(x);
z = sin(x+0.5);
plot(x,y);
hold on;
plot(x,z);
```

2. Möglichkeit

```
x = 0:0.1:10;
y = sin(x);
z = sin(x+0.5);
plot(x,y,x,z);
```

Durch einen weiteren Parameter werden in der Zeichnung Linientyp, Farbe und Markierungstyp bestimmt.

Linientyp
Für den Linientyp gibt es die folgenden Möglichkeiten:

`-`	durchgezogene Linie
`:`	gepunktete Linie
`-.`	Punkt-Strich-Linie
`--`	gestrichelte Linie

Markierungstyp
Weiter besteht die Möglichkeit, die Punkte der Wertetabelle zu markieren bzw. nur die Punkte zu zeichnen (ohne die Verbindungsstrecke). Matlab hat die folgenden Markierungstypen:

`.`	Punkt
`o`	Kreis
`x`	Kreuz
`+`	Plus
`*`	Stern
`s`	Quadrat
`d`	Raute
`v`	Dreieck, Spitze nach unten
`^`	Dreieck, Spitze nach oben
`<`	Dreieck, Spitze nach links
`>`	Dreieck, Spitze nach rechts
`p`	Pentagramm
`h`	Hexagramm

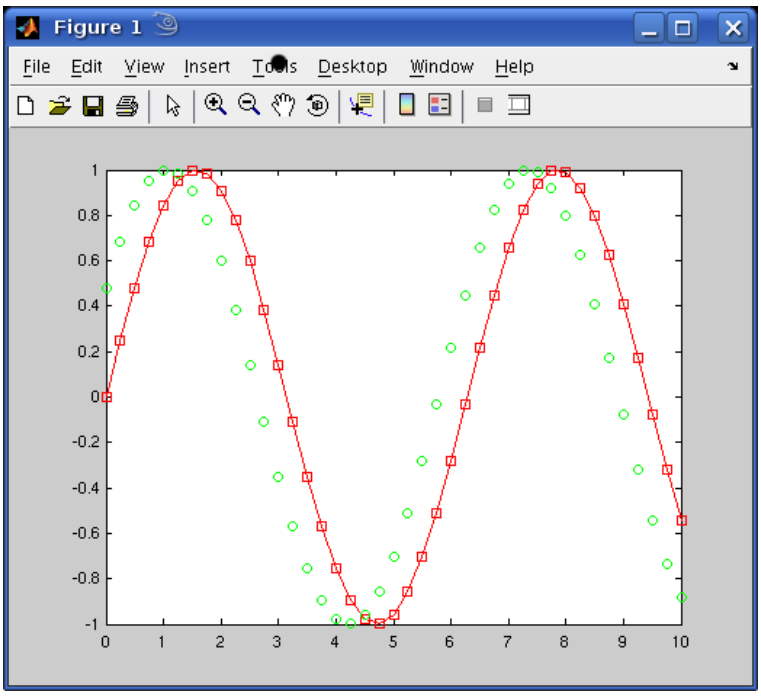
Farben
Die Auswahl der Farbe erfolgt durch:

y	gelb
m	magenta
c	cyan
r	rot
g	grün
b	blau
w	weiß
k	schwarz

Beispiel:

```
x = 0:0.25:10;
y = sin(x);
z = sin(x+0.5);
plot(x,y,'rs-');
hold on;
plot(x,z,'go');
```

Bei der ersten Kurve werden die berechneten Punkte mit einem Quadrat markiert und aufeinander folgende Punkte durch eine Gerade verbunden. Bei der zweiten Kurve werden die Punkte nur durch einen grünen Kreis markiert.



Die Graphik wird in einem neuen Fenster dargestellt, in dem sie weiter bearbeitet werden kann, z.B. Hinzufügen von Text oder Pfeilen.

Desweiteren besteht die Möglichkeit, die Größe und die Farbe der Markierungen zu ändern sowie das Innere der Markierungszeichen durch eine Farbe auszufüllen. Ferner kann man die Dicke der Linien wählen. Dazu werden weitere Parameter in das `plot`-Kommando aufgenommen. Testen Sie mal das folgende Matlab-Programm:

```
x = 0:0.5:10;
y = sin(x);
plot(x,y,'rs--','MarkerEdgeColor','k','MarkerFaceColor','g',...
                'MarkerSize',15,'LineWidth',2)
```

Neben den oben definierten Standard-Farben kann jede beliebige Farbe definiert werden durch Angabe eine Vektors der Länge 3 mit Zahlen zwischen 0 und 1, welcher die Rot-, Grün- und Blau-Anteile (in dieser Reihenfolge) enthält; zum Beispiel

```
plot(x,y,'color',[0.8,0.2,1.0]);
```

2.4.2 3D-Graphik

Matlab besitzt eine ausgezeichnete 3D-Graphik. Wir stellen hier nur einige Elemente vor.

Zeichnen einer Kurve im Raum

In der Mathematik ist eine Raumkurve in der Regel durch eine Parametrisierung gegeben:

$$(x(t), y(t), z(t)) \quad \text{mit } t \in [a, b].$$

Dabei sind $x(t),\ y(t),\ z(t)$ reelle differenzierbare Funktionen.

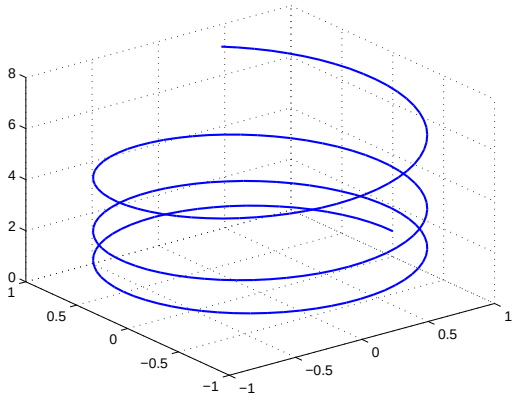
Im Matlab verwendet man zum Zeichnen von Raumkurven das Kommando

```
plot3(x,y,z)
```

Dabei sind x, y, z Vektoren, welche die Werte $x(t_i),\ y(t_i),\ z(t_i),\ i = 1, \ldots, n$ enthalten.

Beispiel

```
t=0:0.01:2;
x=cos(10.*t);
y=sin(10.*t);
z=exp(t);
plot3(x,y,z);
grid on;
```



Der Matlab-Befehl `grid on` bewirkt, dass ein Gitter gezeichnet wird. Dadurch ergibt sich eine bessere räumliche Darstellung der Kurve. Mit dem Kommando `grid off` wird dieses Gitter wieder entfernt.

Zeichnen des Graphen einer Funktion $f : D \subset \mathbb{R}^2 \to \mathbb{R}$

Eine Funktion $f : D \subset \mathbb{R}^2 \to \mathbb{R}$ kann als "Gebirge" über dem Definitionsbereich oder als Höhenkarte dargestellt werden. Für die Darstellung als Gebirge gibt es mehrere Varianten. Wir veranschaulichen dies exemplarisch für die Funktion

$$f(x, y) := 5\exp(-x^2 - (y-2)^2) + x^2 + (y-2)^2$$

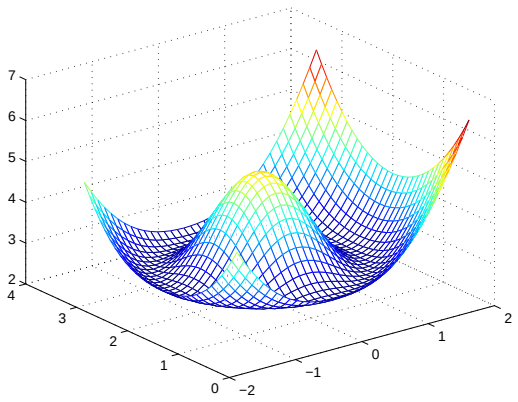
mit Definitionsbereich $D = [-1.5, 2] \times [0.5, 3.5]$.

Der Matlab-Befehl `meshgrid` wählt aus dem Definitionsbereich Gitterpunkte aus, in welchen die Funktion ausgewertet wird.

```
[X,Y] = meshgrid(-1.5:0.1:2,0.5:0.1:3.5);
Z = 5.*exp(-X.^2-(Y-2).^2)+X.^2+(Y-2).^2;
```

Für die graphische Darstellung kennt Matlab drei Varianten:

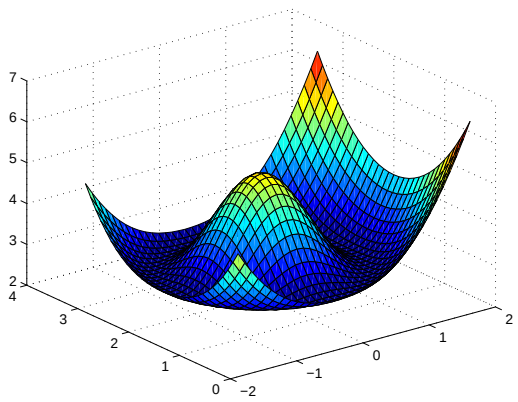
1. `mesh(X,Y,Z)`

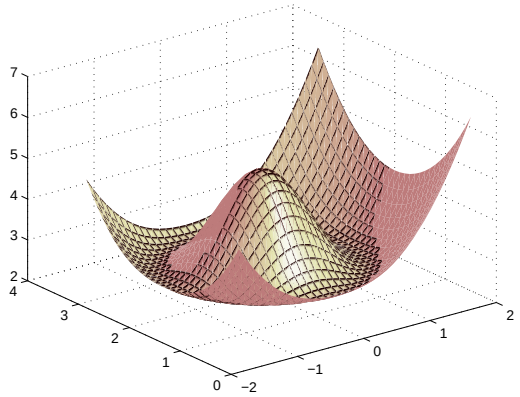


2.
```
surf(X,Y,Z);
colormap(jet);
```

Der 2. Befehl wählt eine Farbpalette aus.

3.
```
surfl(X,Y,Z);
shading interp;
colormap(pink);
```



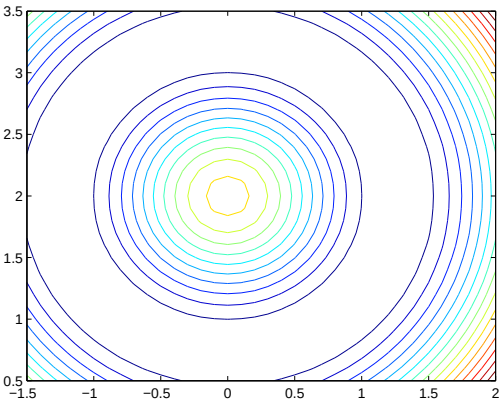


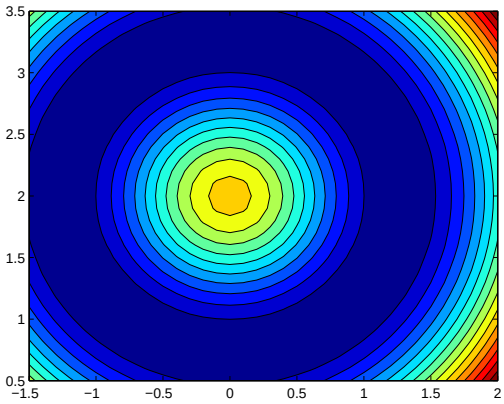
Eine weitere graphische Darstellung von $f : D \subset \mathbb{R}^2 \to \mathbb{R}$ ist die Höhenkarte. Auch dafür bietet Matlab Möglichkeiten:

Die folgenden Matlab-Befehle erstellen von der obigen Funktion eine Höhenkarte mit 15 Niveau-Linien

```
contour(X,Y,Z,15);
```

```
contourf(X,Y,Z,15);
```





Selbstverständlich kann man die einzelnen Höhenlinien auch mit dem entsprechenden Wert beschriften. Testen Sie mal das folgende Programm:

```
[X,Y] = meshgrid(-1.5:0.1:2,0.5:0.1:3.5);
Z = 5.*exp(-X.^2 - (Y-2).^2) + X.^2 + (Y-2).^2;
[c,h] = contour(X,Y,Z,15);
clabel(c,h,'manual');
```

Nach dem Start dieses Programms können Sie mit der Maus die Höhenlinien auswählen, die beschriftet werden sollen.

Das Matlab-Kommando

```
help graph3d
```

liefert weitere Information über die vielfältigen Möglichkeiten (etwa hinsichtlich Farbenwahl oder Schattierungen) bei 3D-Zeichnungen.

2.4.3 Graphik beschriften

Es gibt eine ganze Reihe von Möglichkeiten, um eine erstellte Graphik zu beschriften. Dabei können einige Elemente aus dem Paket LaTeX verwendet werden.

Achsen-Handling

Normalerweise wählt Matlab die Achsen und deren Skalierung automatisch so, dass die Graphik das Figure-Fenster gerade ausfüllt. Mit dem Befehl `axis` wird das Erscheinungsbild der Achsen beeinflusst:

`axis off`	Achsen werden entfernt
`axis on`	Achsen werden gezeichnet
`axis equal`	gleicher Maßstab für alle Achsen
`axis normal`	Maßstab so, dass Graphik raumfüllend
`axis square`	liefert ein quadratisches Bild

Der Befehl

```
axis([xmin xmax ymin ymax])
```

zeichnet die X-Achse von *xmin* bis *xmax* und die Y-Achse von *ymin* bis *ymax*.

Achsenbeschriftung

Die Achsen werden mit den Kommandos `xlabel, ylabel` bzw. `zlabel` beschriftet. Beispiele dazu sind

```
xlabel('Konzentration c [\mu Mol]');
ylabel('Geschwindigkeit \nu(c)');
```

Dabei sind auch einige Sonderzeichen aus der Mathematik möglich, z. B. griechische Buchstaben, welche wie in LaTeX eingegeben werden.

Überschrift

Durch den Matlab-Befehl `title` wird das Bild mit einer Überschrift versehen, z. B.

```
title('Logistische Kurve')
```

Legende

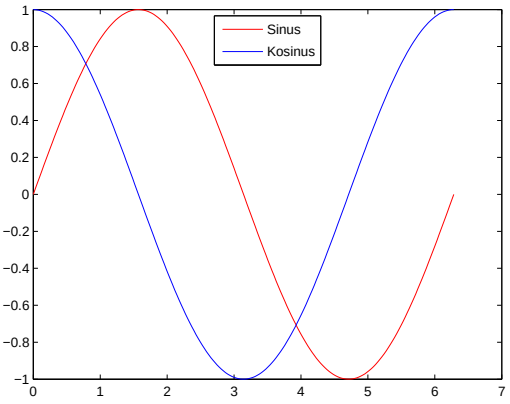
Die Graphik kann durch das Kommando

```
legend('String1','String2',...,'Location', <Ausrichtung>)
```

mit einer Legende versehen werden. Für `<Ausrichtung>` sind u.a. möglich: `'Best', 'West', 'North', 'NorthWest',...`

Beispiel

```
x = linspace(0,2*pi,200);
plot(x,sin(x),'r');
hold on;
plot(x,cos(x),'b');
legend('Sinus','Kosinus','Location','Best');
```



Text in der Graphik

An jeder beliebigen Stelle der Zeichnung kann Text platziert werden. Dazu gibt es zwei Möglichkeiten:

`gtext('a = 100')`	Text wird mit der Maus platziert
`text(10,20,'c = 1.5')`	Text wird an der Stelle (10,20) platziert

Schriftarten

Bei den obigen Kommandos können für den Text verschiedene Schriftarten gewählt werden. So wird zum Beispiel durch

```
title('{\bf Schaubild1: }{\it Käfer} über der Zeit');}')
```

das Wort **Schaubild1:** fett gedruckt und das Wort *Käfer* in der Schriftart Italics. Eine weitere Schriftart ist Slanted, welche mit `\sl` eingeleitet wird.

Schriftgröße

Die Schriftgröße wird durch weitere Parameter angegeben:

```
title('logistische Kurve','FontSize',15);
gtext('a = 300','FontSize',5);
xlabel(' Zeit \tau','FontSize',12);
```

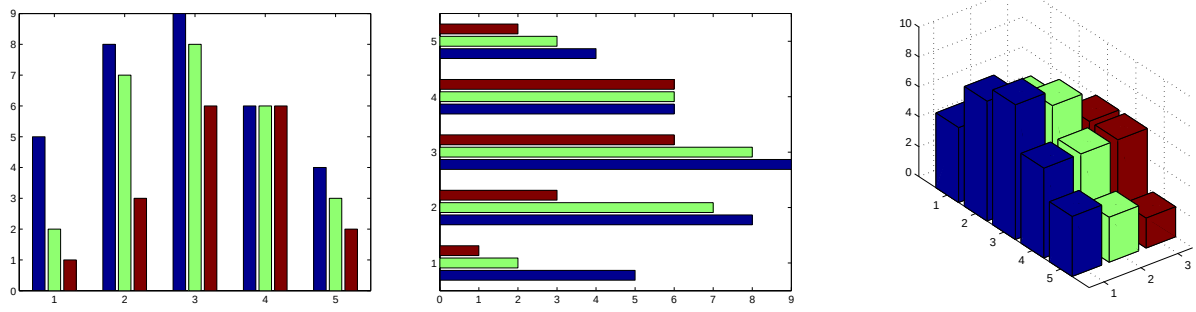
2.4.4 Balken- und Kuchendiagramme

Für Balkendiagramme müssen die zu zeichnenden Werte zunächst in einer Matrix abgelegt werden, z. B.

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

Danach liefern die Matlab-Kommandos `bar, barh` bzw. `bar3` die folgenden Diagramme:

`bar(Y);` `barh(Y);` `bar3(Y);`



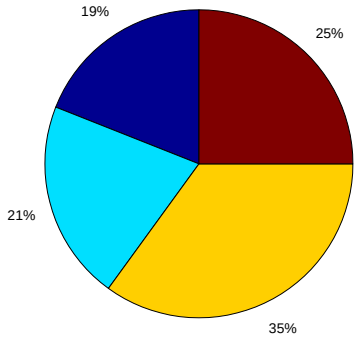
Möglich ist auch eine Achsenbeschriftung, z.B.

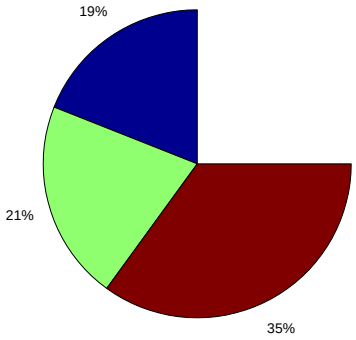
```
bar([2010 2011 2012 2013 2014],Y);
```

Für Kuchendiagramme werden die Anteile in einem Vektor zusammengefasst und dann mit dem Matlab-Befehl `pie` gezeichnet, z. B.

```
x = [0.19 0.21 0.35 0.25];
pie(x);
```

```
u = [0.19 0.21 0.35];
pie(u);
```





Statt der Prozente können auch andere Texte an diesen Stellen platziert werden, z. B.

```
pie(x,{'SPD','CDU','FDP','Grüne'});
```

Selbstverständlich können zur Beschriftung der Diagramme die in Abschnitt 2.4.3 genannten Kommandos verwendet werden.

2.4.5 Der `patch` - Befehl

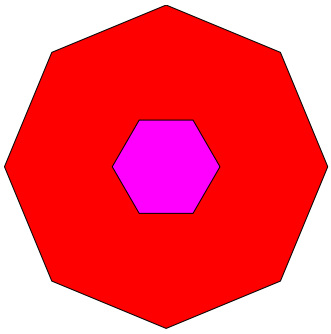
Ein geschlossener Polygonzug kann mit einer Farbe gefüllt werden. Das Matlab-Kommando dazu lautet

```
patch(x,y,color)
```

Die Vektoren `x` bzw. `y` enhalten die x- bzw. y-Koordinaten der Ecken des Polygonzuges.

Beispiel

```
axis equal; hold on;
t = 0:pi/4:2*pi;
x = 3*cos(t);
y = 3*sin(t);
patch(x,y,'r');
s = 0:pi/3:2*pi;
u = cos(s);
v = sin(s);
patch(u,v,[1 0 1]);
```



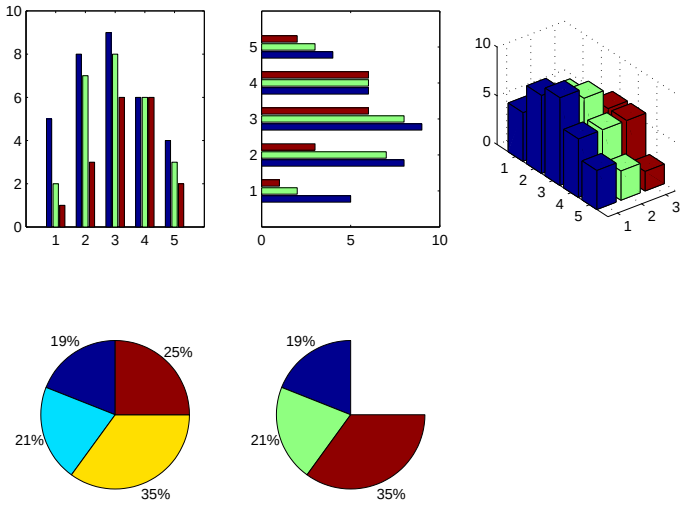
2.4.6 Unterbilder

Man kann mehrere Bilder in einem Graphik-Fenster darstellen. Dazu wird das Matlab-Kommando `subplot(m,n,r)` verwendet. Dies bedeutet, dass das Graphik-Fenster in insgesamt mn Unterbilder (mit m Zeilen und n Spalten) zerlegt wird und aktuell das r-te Bild angesprochen wird.

Beispiel

```
Y = [5 2 1; 8 7 3; 9 8 6; 6 6 6; 4 3 2];
subplot(2,3,1);
bar(Y);
subplot(2,3,2);
```

```
barh(Y);
subplot(2,3,3);
bar3(Y);
x = [0.19 0.21 0.35 0.25];
subplot(2,3,4);
pie(x);
subplot(2,3,5);
u = [0.19 0.21 0.35];
pie(u);
```



2.4.7 Graphik abspeichern

Eine erstellte Graphik kann in verschiedenen Formaten abgespeichert und so in andere Texte (z.B. LaTeX - Programme) eingebunden werden.

Dazu klicken Sie in dem Fenster `Figure` zunächst den Menüpunkt `File` und dann `Save As` an. Danach erscheint ein neues Fenster mit der Überschrift `Save As`. Hier wählen Sie den Dateityp und den Dateinamen aus.

Als Dateityp kommen vor allem in Frage:

```
EPS file (*.eps)
JPEG image (*.jpg)
Portable Document Format (*.pdf)
```

2.5 Programmsteuerung

2.5.1 Logische Ausdrücke

Beim Programmieren besteht häufig der Wunsch, eine bestimmte Anweisung von einer Bedingung abhängig zu machen. Zum Beispiel kann man die Quadratwurzel einer reellen Zahl a nur für $a \geq 0$ berechnen. Eine solche Bedingung nennt man einen *logischen Ausdruck*; er kann nur die Werte *wahr* (engl. *true*) oder *falsch* (engl. *false*) annehmen. In Matlab wird der Wert *wahr* durch 1 und der Wert *falsch* durch 0 dargestellt.

Logische Ausdrücke in der Mathematik sind zum Beispiel

$$\begin{aligned} x &< 10 \\ a+b &\geq y+5 \\ x^2+y^2 &= 5 \\ a &\neq \sqrt{x-y} \end{aligned}$$

Die Zeichen $< > \leq \geq \neq =$ nennt man *Vergleichsoperatoren.*

In Matlab gibt es diese Vergleichsoperatoren ebenfalls:

Matlab	math. Bedeutung
<	$<$
<=	$\leq$
>	$>$
>=	$\geq$
==	$=$
~=	$\neq$

In Matlab-Notation lauten die obigen Beispiele:

```
x < 10
(a + b) >= (y + 5)
(x.^2 + y.^2) == 5
a ~= sqrt(x-y)
```

Im Allgemeinen wirken die Vergleichsoperatoren auf Matrizen (derselben Größe). Der Vergleich wird dann elementweise durchgeführt.

Beispiel: Für

$$A = \begin{pmatrix} 2 & 7 & 5 & 3 \\ 4 & 6 & 8 & 0 \end{pmatrix} \quad , \quad B = \begin{pmatrix} 5 & 7 & 3 & 2 \\ 5 & 5 & 7 & 0 \end{pmatrix} \quad ,$$

liefert der Matlab-Befehl

```
A == B
```

das Ergebnis

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

Logische Ausdrücke kann man kombinieren durch die logischen Operatoren *und, oder* bzw. *nicht*; zum Beispiel

$$a \neq \sqrt{x-y} \qquad \text{und} \qquad x < 10 \quad .$$

In Matlab gibt es dafür die Operatoren

Matlab	math. Bedeutung
&	und
|	oder
~	nicht

Der logische Ausdruck aus obigem Beispiel lautet dann

```
(a ~= sqrt(x-y)) & (x < 10)
```

An dieser Stelle sind einige Worte zur Klammersetzung angebracht. Matlab hat gewisse Voreinstellungen, wenn runde Klammern fehlen. Dies kann dazu führen, dass nicht die von uns gewünschte Größe berechnet wird. Da zusätzliche (überflüssige) Klammerung (mit runden Klammern) nicht schadet, lautet die Empfehlung im Zusammenhang mit den logischen Ausdrücken: verwende generell eine Klammerung. Dies erhöht zum einen die Lesbarkeit des Matlabprogrammes, zum anderen braucht man sich dann die Matlab-Voreinstellungen nicht zu merken.

Beispiel
Das Matlab-Kommando zur Berechnung von $2^{2\cdot 3}$ lautet `2^(2*3)`. Dagegen liefert `2^2*3` ein falsches Ergebnis (berechnet $2^2 \cdot 3$).

Anmerkung: Ausführliche Information über die voreingestellte Auswertungsreihenfolge von Ausdrücken erhalten Sie durch Eingabe von

```
doc precedence
```
.

2.5.2 Verzweigungen

Verzweigungen dienen dazu, in Abhängigkeit vom Wert eines logischen Ausdruckes verschiedene Programmteile auszuführen. Dabei unterscheidet man

Einseitige Verzweigung
Diese haben in Matlab die Form

`if` <*logischer Ausdruck*>

Anweisungen (”if-Block”)

`end`

Bei der Programmausführung wird zunächst der logische Ausdruck ausgewertet. Hat dieser den Wert 1 (*wahr*), so werden die Anweisungen des if-Blockes ausgeführt, andernfalls wird in die Zeile nach `end` gesprungen.

Anmerkung: Ist `A` eine Matrix, so ist `if A < 1` nur dann *wahr*, falls **jedes** Element von `A` diese Bedingung erfüllt.

Als Beispiel berechnen wir die Quadratwurzel einer nichtnegativen reellen Zahl:

```
...
if a >= 0
  b = sqrt(a)
end
...
```

Zweiseitige Verzweigung

Hier hat man zusätzlich noch die Möglichkeit, einen Programmteil auszuführen, falls der *logische Ausdruck* den Wert 0 (*falsch*) hat. Die Form ist

```
if  <logischer Ausdruck>
   Anweisungen ("if-Block")
else
   Anweisungen ("else-Block")
end
```

Hat bei der Programmausführung der *logische Ausdruck* den Wert 1 (*wahr*), so wird der if-Block ausgeführt; hat er den Wert 0 (*falsch*), so wird der else-Block ausgeführt.

Als Beispiel berechnen wir wieder die Quadratwurzel einer reellen Zahl. Ist diese Zahl negativ, so soll nun eine Fehlermeldung ausgedruckt werden.

```
...
if a >= 0
   b = sqrt(a);
   text=sprintf('Die Quadratwurzel ist: %8.4f',b);
   disp(text);
else
   disp('Fehler: eingegebener Wert ist negativ');
end
...
```

Mehrfachverzweigungen

Auch Mehrfachverzweigungen lassen sich in Matlab realisieren. Diese haben die Struktur

```
if  <logischer Ausdruck 1>
   Anweisungen ("if-Block")
elseif  <logischer Ausdruck 2>
   Anweisungen ("elseif - Block")
else
   Anweisungen ("else-Block")
end
```

Bei der Programmausführung wird zuerst der *logische Ausdruck 1* ausgewertet. Hat dieser den Wert 1 (*wahr*), so wird der if-Block ausgeführt. Hat der *logische Ausdruck 1* den Wert 0 (*falsch*), dann wird als nächstes der *logische Ausdruck 2* berechnet; hat er den Wert 1 (*wahr*), so wird der elseif-Block ausgeführt, andernfalls der else-Block. In den else-Block gelangt man nur, wenn sowohl der *logische Ausdruck 1* als auch der *logische Ausdruck 2* den Wert 0 (*falsch*) haben.

Beispiel: Die Auswertung der Signumsfunktion

$$\text{sign}(x) := \begin{cases} 1 & \text{für } x > 0 \\ 0 & \text{für } x = 0 \\ -1 & \text{für } x < 0 \end{cases}$$

wird in Matlab durch folgendes Programm realisiert:

```
x = input(' x = ');
if x > 0
  sigma = 1;
elseif x == 0
  sigma = 0;
else
  sigma = -1;
end
text=sprintf(' sign(x) = %g',sigma);
disp(text);
```

Anmerkungen

1. Es sind auch mehrere elseif -Teile möglich.
2. In einem if-Block bzw. else-Block dürfen mehrere Anweisungen stehen, darunter selbst wieder if-Anweisungen.

Als Alternative zu solchen geschachtelten if-Anweisungen bei Mehrfachverzweigungen gibt es die switch-Anweisung, die folgenden formalen Aufbau hat (dabei muss <*ausdruck*> einen Skalar oder einen String ergeben):

`switch` <*ausdruck*>
`case` <*wert1*>
    Anweisungen (1. case-Block)
`case` <*wert2*>
    Anweisungen (2. case-Block)
  ⋮
`case` <*wertn*>
    Anweisungen (n. case-Block)
`otherwise`
    Anweisungen (sonst-Block)
`end`

Hat <*ausdruck*> den Wert <*wert1*>, so werden die Anweisungen des 1. case-Blockes ausgeführt; hat er den Wert <*wert2*>, so die des 2. case-Blockes usw. Hat <*ausdruck*> einen Wert, welcher von <*wert1*> bis <*wertn*> verschieden ist, so wird der sonst-Block ausgeführt. Der otherwise-Teil darf weggelassen werden.

Beispiel

```
n = input(' n = ');
switch n
  case 1
    disp('Januar');
  case 2
    disp('Februar');
  case 3
    disp('März');
  case {4,5,6}
    disp('Frühjahr');
  case {7,8,9,10,11,12}
    disp('2. Halbjahr');
  otherwise
    disp('falsche Eingabe');
end
```

Anmerkung

Sobald die Anweisungen eines case-Blockes ausgeführt werden, wird anschließend zu `end` gesprungen. Nachfolgende case-Bedingungen werden nicht mehr geprüft.

2.5.3 Schleifenbildung

Beim Programmieren besteht häufig der Wunsch, gewisse Programmteile mehrfach zu durchlaufen (mit verschiedenen Daten). Dazu dient das Konzept der Schleifenbildung. Matlab kennt verschiedene Möglichkeiten:

Die for-Schleife

Ist im Voraus bekannt, wie oft ein Programmteil (eine Schleife) durchlaufen werden soll, so verwendet man die sogenannte for-Schleife. Sie hat die Form

`for` *<index>* = *<start>*:*<inkrement>*:*<ende>*

Anweisungen

`end`

Ist das Inkrement 1, so darf es auch weggelassen werden. Das folgende Beispiel belegt einen Vektor.

```
for i = 1:10
   x(i) = 2*i
end
```

Selbstverständlich dürfen die Schleifen auch geschachtelt werden, zum Beispiel

```
for i = 1:5
  for j = 1:3
    A(i,j) = i+j-1
  end
end
```

Die while-Schleife

Hängt die Anzahl der Schleifendurchläufe von einer logischen Bedingung ab, so verwendet man die while-Schleife:

```
while  <logischer Ausdruck>
   Anweisungen
end
```

Beispiel: Es wird so lange eine reelle Zahl über den Bildschirm eingelesen, bis eine positive Zahl eingegeben wird. Erst dann wird mit der Berechnung von `log(x)` fortgefahren.

```
disp('Gib eine positive Zahl ein ');
x = input('x = ');
while x <= 0
   disp('Fehler: x <= 0');
   disp('Gib eine positive Zahl ein');
   x = input('x = ');
end
y = log(x)
```

Anmerkung: Matlab ist eine Sprache, die für den Umgang mit Matrizen und Vektoren entwickelt wurde, d.h. Operationen mit Matrizen und Vektoren werden schnell ausgeführt. Schleifen erfordern dagegen mehr Rechenzeit. Deshalb sollten Sie die Schleifenbildung möglichst vermeiden.

Vorzeitiges Beenden einer Schleife

Es gibt zwei Möglichkeiten:

1. `break` springt zu der Zeile nach `end`,
2. `continue` der aktuelle Schleifendurchlauf wird abgebrochen und die Schleifenabarbeitung mit dem nächsten Schleifendurchlauf fortgesetzt.

2.5.4 Umgang mit Funktionen

Funktionen

Selbstdefinierte Funktionen sind Matlab-Programme, welche Eingabe-Parameter akzeptieren und ein Ergebnis zurückliefern. Sie benutzen einen eigenen Workspace. Funktionen haben folgenden Aufbau:

```
function y = mittelwert(x)
% Kommentarzeilen
      Matlab-Anweisungen
y =  ....
```

Die erste Zeile beginnt immer mit dem Schlüsselwort `function`, danach kommt die Ausgabevariable (hier `y`), dann der Name der Funktion (hier `mittelwert`) und die Eingabeparameter. Das Programm ist in die Datei mit dem Namen <*Funktionsname*>`.m` (hier also `mittelwert.m`) zu speichern. In der zweiten Zeile folgt ein Kommentar, eingeleitet durch das Zeichen % (es sind auch mehrere Kommentarzeilen möglich). Danach folgen

die Berechnungen (Matlab-Anweisungen). Am Ende erfolgt eine Wertzuweisung auf den Ausgabeparameter.

Beispiel: Wir erstellen eine Funktion, welche zunächst die Länge eines Vektors feststellt und dann den Mittelwert berechnet.

```
function mw = mittelwert(u)
% Diese Funktion berechnet den Mittelwert eines Vektors
n = length(u);
mw = sum(u)/n;
```

Das Programm muss in eine Datei namens `mittelwert.m` gespeichert werden.

Im **Command Window** wird diese Funktion nun aufgerufen:

```
u = [1 2 3 4 5 6];
mittelwert(u)
z = [10 20 30 40 50];
mittelwert(z)
```

Mit dem Matlab-Kommando `help mittelwert` wird die Kommentarzeile auf dem Bildschirm ausgegeben.

Unterfunktionen

Jede Funktion, die Sie während Ihrer Matlab-Sitzung aufrufen wollen, muss sich in einer eigenen Datei befinden. Werden jedoch innerhalb der Funktion weitere Funktionen aufgerufen, so dürfen diese in derselben Datei stehen. Solche Funktionen werden als Unterfunktionen bezeichnet; sie können von anderen Matlab-Programmen jedoch nicht direkt aufgerufen werden.

Dazu betrachten wir folgendes Beispiel, welches in die Datei `auswert.m` geschrieben wird.

```
function [mittel,fehler] = auswert(u)
mittel = mwert(u);
fehler = quadrat(u,mittel);

function mw = mwert(u)
% Diese Funktion berechnet den Mittelwert eines Vektors
n = length(u);
mw = sum(u)/n;

function error = quadrat(u,mw)
error = sum((u-mw).^2)
```

Hier werden zwei Unterfunktionen (`mwert` und `quadrat`) definiert, die von der eigentlichen Funktion (`auswert`) intern aufgerufen werden. In einer Matlab-Sitzung würde man die Funktion `auswert` etwa so aufrufen:

```
w = [1 2 3 4 5 6 7];
[a,b] = auswert(w)
```

Funktionen als Parameter in anderen Funktionen

Eine Funktion darf als Parameter auch eine andere Funktion aufrufen. Dabei sind jedoch einige Dinge zu beachten, was anhand eines Beispiels deutlich wird: In die Datei `funkzeichne.m` schreiben wir die folgenden Matlab-Funktion:

```
function x = funkzeichne(funk,data)
werte = feval(funk,data);
plot(data,werte);
```

welche eine andere Funktion zeichnet. Dabei ist `data` ein Vektor, der die Werte aus dem Definitionsbereich enthält, an denen die Funktion ausgewertet wird. Der Parameter `funk` ist die zu zeichnende Funktion. Das Matlab-Kommando `feval` wertet die Funktion `funk` an den Punkten `data` aus.
In einer Matlab-Sitzung wird dann die Funktion `funkzeichne` wie folgt aufgerufen:

```
t = -3:0.01:3;
funkzeichne(@sin,t);
```

Dadurch wird die Sinus-Funktion im Intervall $[-3, 3]$ gezeichnet. Beachten Sie, dass die Übergabe der Funktion `sin` als Parameter `@sin` lautet, also mit vorangestelltem `@`-Zeichen. Entsprechend geht man bei selbst definierten Funktionen vor.

Ebenfalls möglich ist

```
H = @(x)[exp(-x.^2)];
u = -3:0.1:3;
funkzeichne(H,u);
```

Eine weitere Möglichkeit bilden die sogenannten *Inline-Funktionen*. Hier wird durch das `inline` - Kommando der Funktionsausdruck als Parameter übergeben:

```
t = -3:0.01:3;
funkzeichne(inline('(t.^3-t)./exp(t)'),t);
```

Im folgenden Beispiel wird dieser Funktionsausdruck über den Bildschirm eingelesen:

```
disp('Es wird eine Funktion gezeichnet'),
data = input('wertebereich: ');
ausdruck = input('gib die Funktion ein: ','s');
funkzeichne(inline(ausdruck),data);
```

Vorzeitiges Beenden eines Unterprogramms

Durch das Kommando

```
return
```

wird das (Unter)Programm sofort beendet.

Rekursive Funktionen

Funktionen können auch rekursiv definiert werden, z. B.

```
function y = fak(n)
% Rekursive Definition der Fakultät
if n == 0
   y = 1;
else
   y = n*fak(n-1);
end
```

Die Funktion `arrayfun`

Eine gewisse Vorsicht ist angebracht, wenn selbstdefinierte Funktionen auf Vektoren und Matrizen angewandt werden; insbesonde dann, wenn in diesen Funktionen Verzweigungen auftreten. In diesem Fall sollte man das Kommando

`arrayfun(<funktion>,x)`

verwenden, welches die Funktion auf jedes Element von x anwendet.

Beispiel

Wir zeichnen die Funktion $f(x) = \begin{cases} x & \text{für } x < 1 \\ 2 - x & \text{für } x \geq 1 \end{cases}$

im Intervall $[0, 2]$ und erstellen dafür zunächst eine Matlab-Funktion:

```
function y = f(x)
if x < 1
   y = x
else
   y = 2 - x
end

x = 0:0.1:2;
y = arrayfun(@f,x);
plot(x,y)
```

Ein falsches Ergebnis liefert das Programm

```
x = 0:0.1:2;
y = f(x);
plot(x,y)
```

2.6 Einige Ergänzungen

2.6.1 Komplexe Zahlen

Matlab kann auch mit komplexen Zahlen arbeiten. Als imaginäre Einheit wird `i` oder `j` verwendet. Es sind dann die aus der Mathematik bekannten Operationen mit komplexen Zahlen möglich, zum Beispiel

```
z = 2 + 3*i;
u = 5 - 6.3i;
v = z.*u;
r = abs(z);
```

Dabei ist `abs(z)` der Betrag einer komplexen Zahl. Die erste Zeile definiert nur dann eine komplexe Zahl, wenn `i` zuvor nicht als Variable definiert wurde.

Desweiteren liefern die Funktionen

`real(z)` bzw. `imag(z)`

den Realteil bzw. den Imaginärteil einer komplexen Zahl.

Sind `x` und `y` reelle Zahlen, so erzeugt

`z = complex(x,y)`

eine komplexe Zahl mit Realteil `x` und Imaginärteil `y`.

Viele Standard-Funktionen (wie cos, sin, exp, log) sind auch für komplexe Argumente definiert. Matlab erlaubt dies ebenfalls.

Beispiele:

```
exp(2+3i)  ⇝  -7.3151 + 1.0427i
log(2+3i)  ⇝  1.2825 + 0.9828i
cos(2+3i)  ⇝  -4.1806 - 9.1092i
abs(2+3i)  ⇝  3.6056
```

2.6.2 Pixelbilder

Pixelbilder (aus Digitalkameras) bestehen aus einer (großen) $m \times n$-Matrix X. Jedes Element enthält eine natürliche Zahl (z.B. zwischen 0 und 255) und stellt die Farbe des entsprechenden Punktes dar.

Zum Zeichnen einer solchen Matrix als Bild kennt Matlab den Befehl

`image(X)`

Die zu verwendende Farbpalette wird durch Kommando

`colormap(colorcube(N))`

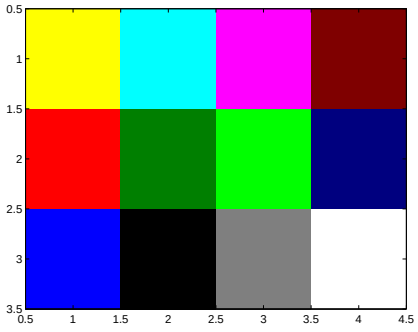
(wobei N die Anzahl der Farben bezeichnet) festgelegt.

`colorcube(N)`

erzeugt eine $N \times 3$-Matrix mit den RGB-Anteilen der Farben.

Beispiel

```
clear all;
X = [1 2 3 4; 5 6 7 8; 9 10 11 12];
colormap(colorcube(12));
image(X);
```



Speicherplatzbedarf

Standardmäßig verwendet Matlab für eine reelle Zahl (double) 8 Byte (= 64 Bit) Speicherplatz. Dies kann für große Pixelbilder schnell zu Speicherproblemen führen.

Um Speicherplatz zu sparen, gibt es für ganze Zahlen folgende Datentypen:

Typ	Zahlenbereich	Speicherplatz
`int8`	-128 bis 127	1 Byte
`uint8`	0 bis 255	1 Byte
`int16`	-32768 bis 32765	2 Byte
`uint16`	0 bis 65635	2 Byte

Für ein $M \times N$-Pixelbild mit höchstens 256 verschiedenen Farben (Farbtiefe) sollte

```
X = zeros(M,N,'uint8')
```

verwendet werden.

```
X = zeros(M,N)
```

braucht den 8-fachen Speicherbedarf.

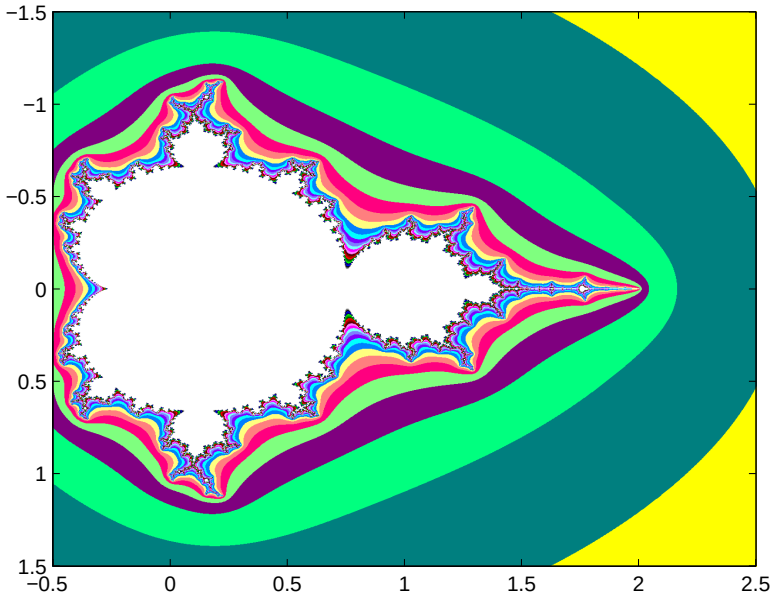
Beispiel: Es wird die *Mandelbrotmenge* gezeichnet. Diese beruht auf der komplexen Iteration

$$z_0 \;=\; 0\,, \qquad z_{k+1} \;=\; z_k^2 - c \quad (k = 0, 1, 2, \ldots) \text{ für } c \in \mathbb{C}.$$

In Anhängigkeit von der Wachstumsgeschwindigkeit dieser Iteration wird der Punkt c in der komplexen Ebene mit einer Farbe dargestellt. Für entsprechend viele Gitterpunkte durchgeführt erhält man so ein Pixelbild.

```
clear all;
tic
N = 32;
L1 = 600;
L2 = 600;
C = zeros(L1,L2,'uint8');
rmax = 15;
for j = 1:L1
  for k = 1:L2
     count = 0;
     z = complex(0,0);
     c = complex(-0.5 + 3.*j./L1, -1.5 + 3.*k./L2);
     r = 0;
     while ((r < rmax) & (count < N))
       z = z*z-c;
       r = abs(z);
       count = count+1;
     end
     C(j,k) = count;
  end
end
colormap(colorcube(N));
toc
image(C','XData',[-0.5,2.5],'YData',[-1.5,1.5]);
```

Beachten Sie, dass in `C(j,k)` der erste Index den Zeilenindex angibt, sich also auf die y-Achse bezieht. Deshalb ist beim Zeichnen die transponierte Matrix zu verwenden.
Dieses Programm liefert die folgende Graphik:



Anmerkung: Im obigen Programm treten noch die Befehle `tic` und `toc`. Sie dienen als Stoppuhr: die zwischen dem Aufruf von `tic` und `toc` verbrauchte Rechenzeit wird ermittelt und auf dem Bildschirm ausgegeben. Damit kann bei umfangreichen Programmen die benötigte Rechenzeit für einzelne Programmteile ermittelt werden.

Mit dem Kommando

```
C = imread('bild1.jpg')
```

wird eine JPEG-Datei (hier mit dem Namen `bild1.jpg`) einglesen und in der Matrix C gespeichert.

2.6.3 Graphiken einbinden in Latex

Es gibt inzwischen einige zusätzliche Pakete (usepackeges), um Graphiken in Latex-Texte einzubinden. Wir gehen hier auf zwei Möglichkeiten ein.

Graphiken im Postskript-Format

Graphiken im Postskript-Format erkennt man an dem Zusatz <*name*>`.ps` bzw. <*name*>`.eps`

In der Präambel muss die Anweisung

```
\usepackage{epsf}
```

stehen. Befindet sich z.B. die Graphik in einer Datei namens `mandelbrot.eps`, so wird sie im Text-Teil eingebunden durch

```
\epsfxsize 10cm  % oder \epsfysize 10cm
\epsfbox{mandelbrot.eps}
```

Dabei gibt `\epsfxsize` die Größe der Graphik in x-Richtung (Breite) an. Alternativ kann `\epsfysize` für die Größe in y-Richtung (Höhe) verwendet werden.

Das folgende Latex-Programm dokumentiert nochmal die Einbindung einer Postskript-Datei

```
\documentclass[12pt]{article}
\usepackage{german}

\usepackage{epsf}

\usepackage[utf8]{inputenc}
\usepackage{amsfonts}
\usepackage{amssymb}
\usepackage{amsmath}
\usepackage{latexsym}
\pagestyle{empty}
\topmargin 0cm
\textheight 25cm
\textwidth 16.0 cm
\oddsidemargin -0.2cm
\begin{document}
  \noindent
Bei der Mandelbrotmenge verwendet man (ausgehend von $c \in \mathbb{C}$)
 die komplexe Iteration
\[ z_0 = 0; \hspace{1em} z_{r+1} \; = \; z_r^2 - c \; \]
und erhält das folgende Schaubild:
\newline

\epsfxsize 8cm

\epsfbox{mandelbrot.eps}

\end{document}
```

Dieses Programm ist mit dem Linux-Kommando

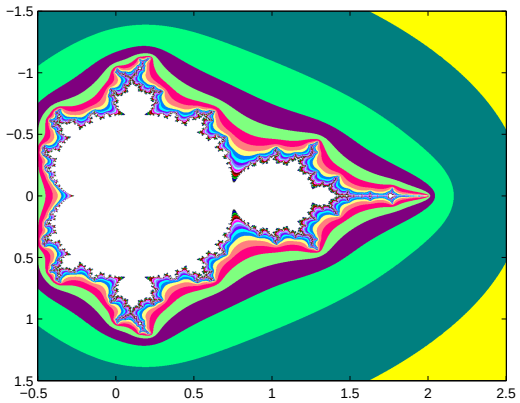
`latex` <dateiname>

(ohne den Zusatz `.tex`) zu bearbeiten und erzeugt folgenden Ausdruck:

Bei der Mandelbrotmenge verwendet man (ausgehend von $c \in \mathbb{C}$) die komplexe Iteration

$$z_0 = 0; \quad z_{r+1} \; = \; z_r^2 - c$$

und erhält das folgende Schaubild:



Graphiken im pdf- bzw. jpg-Format

Um Bilder im pdf- bzw. jpg-Format einzubinden, ist in der Präambel das Kommando

`\usepackage{graphicx}`

aufzunehmen. Ist die Graphik in der Datei `bild.pdf` (bzw. `bild1.jpg`) gespeichert, so wird diese im Text durch

`\includegraphics[height=10cm, width=8cm]{bild.pdf}` bzw.
`\includegraphics[height=10cm, width=8cm]{bild1.jpg}`

eingebunden. Hier werden die Bilder mit einer Höhe von 10 cm und einer Breite von 8 cm dargestellt. Die tatsächlichen Maße auf den Ausdruck können jedoch davon abweichen (abhängig von den Latex- und Druckereinstellungen).

Beispiel:

```
\documentclass[12pt]{article}
\usepackage{german}

\usepackage{graphicx}

\usepackage[utf8]{inputenc}
%\usepackage[latin1]{inputenc}
\usepackage{amsfonts}
\usepackage{amssymb}
\usepackage{amsmath}
\usepackage{latexsym}
\pagestyle{empty}
\textheight 25cm
\textwidth 16.0 cm
\begin{document}
  \noindent
Bei der Mandelbrotmenge verwendet man
(ausgehend von$c \in \mathbb{C}$) die komplexe Iteration
\[ z_0 = 0; \hspace{1em} z_{r+1} \; = \; z_r^2 - c \; \]
und erhält das folgende Schaubild: \newline

\includegraphics[height=10cm,width=10cm] {bild.pdf}

\newline Auch Bilder von einer Digitalkamera (jpg-Format) lassen sich
mit dem {\tt graphicx}-Paket einbinden: \newline

\includegraphics[height=6cm,width=8cm] {bild1.jpg}

\end{document}
```

Achtung: Dieses Programm ist mit dem Linux-Kommando

`pdflatex` <dateiname>

(ohne den Zusatz `.tex`) zu bearbeiten (nicht mit `latex` <dateiname>). Es wird dann sofort eine pdf-Datei mit dem Namen <dateiname>`.pdf` erstellt, welche mit dem Acrobat Reader bzw. mit dem Linux-Kommando `okular` betrachtet werden kann.

Anmerkungen:

1. Sie müssen sich für eine der beiden Möglichkeiten entscheiden. So können z.b. bei der zweiten Methode keine Postskript-Dateien eingebunden werden.

2. Linux stellt eine ganze Reihe von Kommandos zur Verfügung, um ein gewisses Dateiformat in ein anderes Format zu wandeln (dabei können jedoch Qualitätsverluste auftreten). So erzeugt zum Beispiel der Befehl

   `ps2pdf` <dateiname>`.ps`

   aus der Postskript-Datei eine pdf-Datei mit dem Namen <dateiname>`.pdf` .

2.6.4 Zufallszahlen

Matlab kann Zufallszahlen erzeugen (genau genommen handelt es sich um sogenannte Pseudo-Zufallszahlen).

Das Kommando

```
A = rand(m,n)
```

erzeugt eine $m \times n$-Matrix mit gleichverteilten Zufallszahlen aus dem Intervall $[0, 1]$.

Sind gleichverteilte Zufallszahlen aus dem Intervall [-1,1] gesucht, so lautet der Befehl

```
B = 2.*rand(m,n) - 1
```

Die Matlab-Funktion

```
randn(m,n)
```

liefert (standard)normalverteilte Zufallszahlen. Desweiteren erhält man durch

```
randperm(n)
```

eine zufällige Permutation der Zahlen $1, 2, 3, \ldots, n$.

2.6.5 Computeranimationen (Life-Graphik)

Matlab bietet auch Möglichkeiten für Animationen (bewegte Bilder). Dabei sind zwei Fälle zu unterscheiden.

1. Echtzeitsimulation

Das Rechnen und Zeichnen erfolgt gleichzeitig. Dies macht aber nur Sinn, wenn das Berechnen der Graphiken rasch geht, d.h. mindestens 20 Bilder pro Sekunde.

Beispiel: Wir betrachten die Brownschen Bewegung.

```
n=50; s=0.02;
x=rand(n,1)-0.5; y=rand(n,1)-0.5;

h = plot(x,y,'o','MarkerFaceColor','b');

axis([-1 1 -1 1]);
axis square;
while 1
  x = x + s*(rand(n,1)-0.5);
  y = y + s*(rand(n,1)-0.5);

  set(h,'Xdata',x,'Ydata',y);
  drawnow;

  % pause(0.1);
end
```

Der `plot` - Befehl erzeugt neben der Zeichnung ein sogenanntes *Graphics Handle*: eine Variable `h`, in der alle Informationen für die Graphik abgelegt sind. Mit dem Befehl

`get(h)`

werden die einzelnen Komponenten aufgelistet; mit

`set(h,`<komponente>`,`<wert>`)`

können einzelne Komponenten verändert werden. Durch das Kommando

`drawnow`

wird die (veränderte) Graphik gezeichnet. Läuft das Video zu schnell, so kann man mit

`pause(x)`

eine Zeitlupe einbauen: die Rechnung wird um `x` Sekunden verzögert.

2. Video erstellen und abspielen

Hier werden die erzeugten Graphiken in einer Variablen abgespeichert (Vektor, jede Komponente enthält ein Bild) und später als Film abgespielt. Dies Vorgehensweise ist bei rechenintensiven Aufgabenstellungen nötig.

Das Video wird in der Variablen `M` gespeichert. Der Befehl `M(i) = getframe` speichert der aktuelle Graphik in die i-te Komponente der Variabeln `M`.

Durch die Angabe von

`movie(M,n)` bzw. `movie(M,n,m)`

wird das Video `n` mal abgespielt (mit `m` Bildern pro Sekunde).

Im folgenden Beispiel wird ein Video erzeugt:

```
clear all;
N = 50;
for i=1:N
  x = cos(i.*2.*pi./N);
  y = sin(i.*2.*pi./N);
  plot(x,y,'o','Markersize',20,'Markerfacecolor','r');
  axis([-1.5 1.5 -1.5 1.5]);
  M(i) = getframe;
end
```

2.6.6 Strukturierter Vektor und Listenverarbeitung

Ein **strukturierter Vektor** ist ein Vektor, bei dem jede Komponente von unterschiedlichem Datentyp (Skalar, String, logische Größe, Vektor, Matrix oder selbst wieder strukturierter Vektor) sein darf.
Die einzelnen Komponenten erhalten einen Namen.

Beispiel: Die Variable `Student` bestehe aus

`name`	Typ String
`vorname`	Typ String
`matrikelnummer`	Typ Skalar
`studiengang`	Typ String
`note`	Typ strukt. Vektor

Der strukturierte Vektor `Note` bestehe aus

`ana1`	Typ Skalar
`ana2`	Typ Skalar
`la1`	Typ Skalar
`la2`	Typ Skalar
`coma`	Typ Skalar

Es gibt zwei Möglichkeiten, solche strukturierte Vektoren anzulegen:

1. Durch das Matlab-Kommando `struct`; z.B.

```
student = struct('name','Huber','vorname','Klaus','matrikelnummer',...
                 472745,'studiengang','BA Math',...
                 'note', struct('ana1',2.7,'ana2',0,'la1',3.3,...
                              'la2',2.7,'coma',1.3))
```

2. Wertzuweisung an die einzelnen Komponenten; z.B.

```
student.name = 'Huber' ;
student.vorname = 'Klaus';
student.matrikelnummer = 472745;
student.studiengang = 'BA Math';
student.note.ana1 = 2.7;
student.note.ana2 = 0;
student.note.la1 = 3.3;
student.note.la2 = 2.7;
student.note.coma = 1.3;
```

Aus 2. wird auch ersichtlich, wie man auf die einzelnen Komponenten in einem Matlab-Programm zugreift. So wird z.B. durch

```
(student.note.ana1 + studend.note.ana2)/2
```

die Durchschnittsnote aus ana1 und ana2 berechnet.

Anmerkung: Für den Vergleich von 2 Strings gibt es das Matlab-Kommando `strcmp`; z.B. `strcmp(student.studiengang,'BA Math')`.
Der Befehl `student.studiengang == 'BA Math'` liefert eine Fehlermeldung, falls `student.studiengang` nicht aus 7 Zeichen besteht.

Eine **Liste** ist ein Vektor, bei dem jede Komponente aus einem strukturierten Vektor besteht. Das folgende Matlab-Programm legt eine solche Liste an (Vgl. auch Übungen).

```
dateiname='Math_Stud';
liste=[];
weiter='j';
disp(' Eingabe der Daten');
while (weiter == 'j')
   student.name = input(' Name = ','s');
   student.vorname = input(' Vorname = ','s');
   student.matrikel = input(' Matrikelnummer = ');
   student.studiengang = input(' Studiengang = ','s');
   student.geb.jahr = input(' Geburtsjahr = ');
```

```
   student.geb.monat = input(' Geburtsmonat = ');
   student.geb.tag = input(' Geburtstag = ');
   student.note.ana1 = input('Note Analysis I = ');
   student.note.ana2 = input('Note Analysis II = ');
   student.note.la1 = input('Note Lineare Algebra I = ');
   student.note.la2 = input('Note Lineare Algebra II = ');
   student.note.coma = input('Note Coma = ');
   liste = [liste; student];
   weiter = input(' Weiter (j/n) ','s');
end
save(dateiname,'liste');
```

3 Numerisches Rechnen

3.1 Zahlen und ihre Darstellung

Grundlage der Analysis bilden die reellen Zahlen. Wir sind heute daran gewöhnt, eine reelle Zahl im Dezimalsystem als (unendlichen) Dezimalbruch darzustellen. Grundsätzlich kann jedoch als Basis statt der Zahl 10 jede natürliche Zahl $g \geq 2$ gewählt werden.

Sei $g \geq 2$ eine natürliche Zahl und $x \neq 0$ eine reelle Zahl. Dann gibt es eine Darstellung der Gestalt

$$x \;=\; \sigma \cdot g^N \cdot \sum_{\nu=1}^{\infty} x_\nu g^{-\nu} \tag{6}$$

mit $\sigma \in \{-1, +1\}$ (*Vorzeichen*), $N \in \mathbb{Z}$ (*Exponent*) und $x_\nu \in \{0, 1, 2, \ldots, g-1\}$ (*Ziffern*). Man nennt g die *Basis* des Zahlensystems.

Diese Darstellung ist eindeutig, wenn man zusätzlich noch verlangt, dass $x_1 \neq 0$ gilt und dass es zu jedem $n \in \mathbb{N}$ ein $\nu \geq n$ gibt mit $x_\nu \neq g - 1$. Wir sprechen dann von der *normalisierten Darstellung* und schreiben (6) kürzer in der Form

$$x \;=\; \sigma 0.x_1x_2x_3 \cdots \; g^N \tag{7}$$

Ohne diese Zusatzvoraussetzung hätte im Zehnersystem (d.h. $g = 10$) die reelle Zahl 1 die Darstellungen

$$\begin{aligned} 1 &= 0.1000... \cdot 10^1 \quad , \\ 1 &= 0.9999... \cdot 10^0 \quad , \\ 1 &= 0.0100... \cdot 10^2 \quad . \end{aligned}$$

Häufig verwendete Basen sind:

Name des Sytems	Basis g	Ziffern
Dualsystem	2	0,1
Dezimalsystem	10	0,1,2,3,4,5,6,7,8,9
Hexadezimalsystem	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Beispiel: Die reelle Zahl $x = 11.1875$ hat folgende normalisierte Darstellungen

dezimal:	$0.111875 \cdot 10^2$
dual:	$0.10110011 \cdot 2^4$
hexadezimal:	$0.\mathrm{B3} \cdot 16^1$

In Computern wird nicht das Dezimalsystem, sondern das Dualsystem verwendet. Dargestellt werden können dann nur die Null und die Zahlen der Form

$$x \;=\; \sigma \cdot 2^N \cdot \sum_{\nu=1}^{t} x_\nu 2^{-\nu} \;, \qquad x_\nu \in \{0, 1\},\; x_1 = 1 \tag{8}$$

mit einem festen $t \in \mathbb{N}$ und $N^- \leq N \leq N^+,\ N^-, N^+ \in \mathbb{Z}$. Dabei hängen $t,\ N^-$ und N^+ von der konkreten Rechenanlage bzw. von den verwendeten Programmpaketen ab. Diese Zahlen heißen *Maschinenzahlen*. Die Zahl

$$m\ :=\ \sum_{\nu=1}^{t} x_\nu 2^{-\nu}$$

heißt die *Mantisse* von x und t die *Mantissenlänge*. Daneben bezeichnen wir σ als *Vorzeichen* und N als *Exponent* der Zahl x. Man nennt Zahlen in der Dartellung (8) auch *Gleitkommazahlen*.
Matlab verwendet für die Darstellung von reellen Zahlen 64 Bit (= 8 Byte); diese teilen sich auf in

Vorzeichen	1 Bit
Exponent	11 Bit
Mantisse	52 Bit

3.2 Operationen mit Gleitkommazahlen

Die Menge M der Maschinenzahlen (bei festem $t,\ N^-,\ N^+$) ist natürlich endlich. Man spricht hier auch von einem diskreten Raum. Eine reelle Zahl x muss in der Regel durch eine Maschinenzahl $\bar{x}$ ersetzt (*approximiert*) werden. Diesen Prozess nennt man *Runden*; es entsteht ein Fehler. Dabei unterscheiden wir zwei Fehlerarten:

Definition: Es sei $x \in \mathbb{R}$ und $\bar{x}$ eine Näherung für x.
(i) $x - \bar{x}$ heißt *absoluter Fehler*.
(ii) Für $x \neq 0$ heißt $\frac{x-\bar{x}}{x}$ der *relative Fehler*.

Wir definieren folgende **Rundungsvorschrift:**
Sei $g \geq 2$ gerade (nur diese Fälle sind von praktischer Bedeutung), $t \in \mathbb{N}$ (fest vorgegebene Mantissenlänge) und $x \in \mathbb{R} \setminus \{0\}$ mit

$$x\ =\ \sigma \cdot g^N \cdot \sum_{\nu=1}^{\infty} x_\nu g^{-\nu} \quad (N^- \leq N \leq N^+\ ).$$

Dann setzt man

$$\mathrm{rd}(x)\ :=\ \begin{cases} \sigma \cdot g^N \cdot \sum\limits_{\nu=1}^{t} x_\nu g^{-\nu} \quad , & \text{falls } x_{t+1} < \frac{g}{2} \quad \text{(abrunden)}, \\ \sigma \cdot g^N \cdot \left(\sum\limits_{\nu=1}^{t} x_\nu g^{-\nu} + g^{-t} \right) \ , & \text{falls } x_{t+1} \geq \frac{g}{2} \quad \text{(aufrunden)}. \end{cases}$$

$\mathrm{rd}(x)$ heißt der auf t *Stellen gerundete Wert von* x. (Beachte: rd hängt auch von t ab; korrekter wäre deshalb $\mathrm{rd}_t(x)$. Wir verzichten darauf, da wir stets mit einer fest vorgegebenen Mantissenlänge arbeiten.)

Satz: (o.B., vgl. Hämmerlin-Hoffmann)
(i) Für den absoluten Fehler gilt: $|\mathrm{rd}(x) - x|\ \leq\ 0.5 g^{N-t}$
(ii) Für den relativen Fehler gilt: $\left|\frac{\mathrm{rd}(x) - x}{x}\right|\ \leq\ 0.5 g^{-t+1}$

Definition: Die Zahl $\tau := 0.5 g^{-t+1}$ heißt die *relative Rechengenauigkeit* der t-stelligen

Gleitkomma-Arithmetik.

Setzt man $\varepsilon := \dfrac{\mathrm{rd}(x) - x}{x}$, so gilt

$$\mathrm{rd}(x) \;=\; x(1+\varepsilon) \quad \text{mit } |\varepsilon| \le \tau$$

In Matlab ist die relative Rechengenauigkeit $\tau = 2^{-52}$. Diesen Wert erhält man durch Eingabe von `eps`.

Verknüpfung von Gleitkommazahlen
Es bezeichne nun $\diamond$ eine der Rechenoperationen $+, -, *, /$. Die Verknüpfung zweier Maschinenzahlen x und y durch eine dieser Operationen ist i.A. keine Maschinenzahl mehr; es muss gerundet werden. Das Ergebnis ist also

$$\mathrm{rd}(x \diamond y) \;=\; (x \diamond y)(1+\varepsilon) \quad \text{mit } |\varepsilon| \le \tau \;.$$

Beispiel: Sei $g := 10,\ t := 3$

$$\begin{array}{l} x = 0.123{\cdot}10^6\,, \quad y = 0.456{\cdot}10^2 \quad , \\ x + y = 0.1230456{\cdot}10^6 \quad , \\ \mathrm{rd}(x+y) = 0.123{\cdot}10^6 \quad . \end{array}$$

Sind nun x und y keine Maschinenzahlen, so müssen diese zunächst in Maschinenzahlen umgewandelt werden, bevor die Verknüpfung ausgeführt wird:

$$\begin{array}{rcll} \mathrm{rd}(x) & = & x(1+\varepsilon_1) & \text{mit } |\varepsilon_1| \le \tau \quad , \\ \mathrm{rd}(y) & = & y(1+\varepsilon_2) & \text{mit } |\varepsilon_2| \le \tau \quad . \end{array}$$

In einem ersten Schritt untersuchen wir nun, wie sich Rundungsfehler bei der Addition auswirken. Wird zusätzlich unterstellt, dass die Addition der gerundeten Zahlen auf dem Rechner exakt ausgeführt wird, so erhält man

$$\mathrm{rd}(x) + \mathrm{rd}(y) \;=\; x + x\varepsilon_1 \;+\; y + y\varepsilon_2$$

und somit für den relativen Fehler

$$\frac{\mathrm{rd}(x) + \mathrm{rd}(y) - (x+y)}{x+y} \;=\; \frac{x}{x+y}\,\varepsilon_1 \;+\; \frac{y}{x+y}\,\varepsilon_2 \;=:\; \delta\,.$$

Bei exakter Rechnung bewirken die relativen Fehler ε_1 bzw. ε_2 bei den Eingangsdaten x und y im Ergebnis einen relativen Fehler δ. Diesen Fehler bezeichnet man auch als *Fortpflanzungsfehler*. Falls x und y dasselbe Vorzeichen haben, so folgt sofort

$$|\delta| \;\le\; |\varepsilon_1| + |\varepsilon_2| \;\le\; 2\tau \quad .$$

Problematisch ist dagegen die Situation $x \approx -y$. Dann kann der relative Fehler bei der Addition sehr groß werden (trotz kleiner Fehler ε_1 und ε_2).

Fazit: Die Subtraktion zweier ungefähr gleich großer Zahlen ist **numerisch instabil**: kleine Fehler in den Summanden können zu einem großen Fehler im Ergebnis führen.

Nun betrachten wir die Situation, dass nach der Rechenoperation des Ergebnis zu einer Maschinenzahl gerundet werden muss. Dann erhält man

$$\begin{aligned} \mathrm{rd}(\mathrm{rd}(x) \diamond \mathrm{rd}(y)) &= (\mathrm{rd}(x) \diamond \mathrm{rd}(y))(1+\varepsilon_3) \quad \text{mit } |\varepsilon_3| \leq \tau \\ &= (x(1+\varepsilon_1) \diamond y(1+\varepsilon_2))(1+\varepsilon_3) \\ &= x \diamond y \ + \ F \quad . \end{aligned}$$

Verwendet man z.B. die Addition als Verknüpfung, so ergibt sich für den absoluten Fehler die Abschätzung (Beweis als Übung)

$$\begin{aligned} |F| &\leq |x|(\tau + \tau(1+\tau)) \ + \ |y|(\tau + \tau(1+\tau)) \\ &= (|x| + |y|)(\tau^2 + 2\tau) \quad . \end{aligned}$$

Die Untersuchung des Rundungsfehlers bei der Auswertung arithmetischer Ausdrücke nennt man *Rundungsfehleranalyse*. Sie wird bei größeren Ausdrücken sehr kompliziert.

3.3 Algorithmen

Eine Reihe (unendliche Summe) kann man im Computer nicht exakt berechnen, sondern muss zuerst durch eine endliche Summe ersetzt werden. Desweiteren ist dem Rechner auch mitzuteilen, in welcher Reihenfolge die Summe ausgewertet werden soll. Dies führt uns auf den Begriff des *Algorithmus*.

Definition: Ein *Algorithmus* ist eine Vorschrift, die aus einer Menge eindeutiger Regeln besteht. Diese spezifizieren eine endliche Aufeinanderfolge von Operationen, so dass deren Ausführung die Lösung eines Problems aus einer bestimmten Problemklasse liefert.

Ein Algorithmus ist also durch die folgenden 3 Punkte gekennzeichnet:

- Bestimmtheit: Jeder Rechenschritt ist eindeutig festgelegt. Jede mögliche Situation wird erfasst, und es muss genau spezifiziert werden, welcher Rechenschritt auszuführen ist.
- Endlichkeit: Der Rechenvorgang wird nach endlich vielen Schritten beendet.
- Allgemeingültigkeit: Die Rechenvorschrift soll auf eine ganze Problemklasse anwendbar sein, wobei die Lösung der verschiedenen Probleme der Klasse lediglich eine Änderung der Eingabe erfordert.

Komplexität von Algorithmen
Zur Lösung desselben Problems kann es verschiedene Algorithmen geben. Ein mögliches Vergleichskriterium ist dann der Rechenaufwand. Mit diesem Themenkreis beschäftigt sich die *Komplexitätstheorie*.

Definition: Sei A ein Algorithmus zur Lösung eines Problems mit $n \in \mathbb{N}$ Eingabedaten. Die Abbildung $T_A : \mathbb{N} \to \mathbb{N}$, die jeder Anzahl von Eingabedaten die Anzahl der Grundoperationen beim Ablauf des Algorithmus zuordnet, heißt *Komplexität* von A.

Die Komplexität gibt den Rechenaufwand an. Da eine Punktoperation ($*$, $/$) wesentlich mehr Aufwand erfordert als eine Strichoperation ($+$, $-$), werden bei Komplexitätsbetrachtungen häufig auch nur die Punktoperationen berücksichtigt.

Zur Beschreibung des Verhaltens der Komplexitätsfunktion verwendet man meistens das *Landau-Symbol* $\mathcal{O}$.

Definition: Es seien $f, g : D \subset \mathbb{R} \to \mathbb{R}$ zwei Funktionen mit $g(x) \neq 0$ für alle $x \in D$. f heißt *von der Ordnung "groß O" von g für* $x \to x_0$, wenn es eine Konstante $C > 0$ und ein $\delta > 0$ gibt, so dass für alle $x \in \mathrm{U}_\delta(x_0)$ mit $x \neq x_0$ die Ungleichung

$$\left|\frac{f(x)}{g(x)}\right| \leq C$$

gilt. Man verwendet dafür die Schreibweise $f(x) = \mathcal{O}(g(x))$ für $x \to x_0$.

Beispiel: Sei $f : \mathbb{N} \to \mathbb{N}$, $f(n) := \frac{1}{2}n(n+3)$. Dann gilt

$$f(n) = \mathcal{O}(n^2) \quad \text{für } n \to \infty$$

(d.h. $g(n) := n^2$), denn

$$\frac{f(n)}{n^2} = \frac{\frac{1}{2}n^2 + \frac{3}{2}n}{n^2} = \frac{1}{2} + \frac{3}{2n} \leq 1 \quad \text{für alle } n \geq 3 \ .$$

Hier kann man also $C = 1$ wählen.

Eigenschaften des Landau-Symbols

1. $f(x) = K \cdot \mathcal{O}(g(x))$ für ein $K \in \mathbb{R} \Rightarrow f(x) = \mathcal{O}(g(x))$
2. $f(x) = \mathcal{O}(g(x))$ und $g(x) = \mathcal{O}(h(x)) \Rightarrow f(x) = \mathcal{O}(h(x))$
3. $f_1(x) = \mathcal{O}(g_1(x))$ und $f_2(x) = \mathcal{O}(g_2(x)) \Rightarrow f_1(x) \cdot f_2(x) = \mathcal{O}(g_1(x) \cdot g_2(x))$

Beispiel zur Berechnung der Komplexität:
Gegeben sei ein Polynom vom Grad n:

$$p(t) := \sum_{i=0}^{n} a_i t^i \tag{9}$$

und $\xi \in \mathbb{R}$. Man bestimme den Funktionswert $p(\xi)$.

Der naive Algorithmus zur Polynomauswertung lautet:
Eingabe: $n \in \mathbb{N}$, $(a_0, a_1, \ldots, a_n) \in \mathbb{R}^{n+1}$, $\xi \in \mathbb{R}$.
Rechenvorschrift:

$p := a_0$
Für $i = 1, \ldots, n$:
  $b := a_i$
  Für $j = 1, \ldots, i$:
    $b := b * \xi$
  {Ende der j-Schleife}
  $p := p + b$
{Ende der i-Schleife}

Ausgabe: p.

Man berechnet also

$$a_0 + a_1 \cdot \xi + a_2 \cdot \xi \cdot \xi + a_3 \cdot \xi \cdot \xi \cdot \xi + \cdots$$

Dieser Algorithmus hat bei Zählung von Additionen und Multiplikationen die Komplexität

$$T_A(n) \;=\; \frac{1}{2}n(n+3) \;=\; \mathcal{O}(n^2) \quad (\text{für } n \to \infty).$$

(nämlich n Additionen und

$$\sum_{i=1}^{n} i \;=\; \frac{1}{2}n(n+1)$$

Multiplikationen.)

3.4 Das Hornerschema

Wir geben nun einen Algorithmus mit kleinerer Komplexität an (zur Berechnung eines Funktionswertes eines Polynoms). Gegeben sei wieder die Situation aus (9):

$$p(t) \;=\; \sum_{i=0}^{n} a_i t^i \;= \sum_{i=0}^{n} a_i^{(0)} t^i \quad .$$

Setze

$a_n^{(1)} := a_n^{(0)}$
Für $i = 1, \ldots, n$:
$\quad a_{n-i}^{(1)} := a_{n-i}^{(0)} + a_{n+1-i}^{(1)} * \xi$

Behauptung: Hier gilt $p(\xi) = a_0^{(1)}$ (Beweis als Übungsaufgabe).

Der Rechenaufwand beträgt in diesem Fall n Multiplikationen und n Additionen:

$$T_A(n) \;=\; 2n \;=\; \mathcal{O}(n) \quad (\text{für } n \to \infty).$$

Für die Handrechnung benutzt man folgendes Schema (*Hornerschema*):

$a_n^{(0)}$	$a_{n-1}^{(0)}$	$a_{n-2}^{(0)}$	$\ldots$	$a_1^{(0)}$	$a_0^{(0)}$	
–	$+a_n^{(1)}\xi$	$+a_{n-1}^{(1)}\xi$	$\ldots$	$+a_2^{(1)}\xi$	$+a_1^{(1)}\xi$	
$a_n^{(1)}$	$a_{n-1}^{(1)}$	$a_{n-2}^{(1)}$	$\ldots$	$a_1^{(1)}$	$a_0^{(1)}$	$= p(\xi)$

Beispiel: Es sei $p(t) := t^5 - 5t^3 + 4t + 1$ und $\xi := 2$.

1	0	–5	0	4	1	
–	2	4	–2	–4	0	
1	2	–1	–2	0	1	$= p(2)$

Wir können nun das Hornerschema mehrfach anwenden und erhalten dann

$a_n^{(0)}$	$a_{n-1}^{(0)}$	$a_{n-2}^{(0)}$	$\dots$	$a_1^{(0)}$	$a_0^{(0)}$	
$-$	$+a_n^{(1)}\xi$	$+a_{n-1}^{(1)}\xi$	$\dots$	$+a_2^{(1)}\xi$	$+a_1^{(1)}\xi$	
$a_n^{(1)}$	$a_{n-1}^{(1)}$	$a_{n-2}^{(1)}$	$\dots$	$a_1^{(1)}$	$a_0^{(1)}$	$= p(\xi)$
$-$	$+a_n^{(2)}\xi$	$+a_{n-1}^{(2)}\xi$	$\dots$	$+a_2^{(2)}\xi$		
$a_n^{(2)}$	$a_{n-1}^{(2)}$	$a_{n-2}^{(2)}$	$\dots$	$a_1^{(2)}$	$= p'(\xi)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$			
$\vdots$	$\vdots$	$\vdots$				
$a_n^{(n-1)}$	$a_{n-1}^{(n-1)}$	$a_{n-2}^{(n-1)}$				
$-$	$+a_n^{(n)}\xi$					
$a_n^{(n)}$	$a_{n-1}^{(n)}$					
$-$						
$a_n^{(n+1)}$						

Dieses Tableau wird als *vollständiges Hornerschema* bezeichnet.

Es gilt die Beziehung

$$a_\nu^{(\nu+1)} \;=\; \frac{p^{(\nu)}(\xi)}{\nu!} \qquad (\nu = 0, \dots, n)$$

(Beweis als Übungsaufgabe).

Anmerkung: Mit dem vollständigen Hornerschema können wir von einem Polynom die Taylor-Entwicklung um den Punkt ξ berechnen, ohne die Ableitungen explizit zu bestimmen.

Beispiel: Bestimmen Sie die Taylor-Entwicklung von

$$p(x) = 3x^4 + 2x - 10$$

um den Punkt $x_0 = -1$.

Hier erhalten wir das folgende vollständige Hornerschema

3	0	0	2	-10	
$-$	-3	3	-3	1	
3	-3	3	-1	-9	$= p(-1)$
$-$	-3	6	-9		
3	-6	9	-10	$= p'(-1)$	
$-$	-3	9			
3	-9	18	$= \frac{p^{(2)}(-1)}{2!}$		
$-$	-3				
3	-12	$= \frac{p^{(3)}(-1)}{3!}$			
$-$					
3	$= \frac{p^{(4)}(-1)}{4!}$				

Damit ergibt sich die Taylor-Entwicklung

$$p(x) = 3(x+1)^4 - 12(x+1)^3 + 18(x+1)^2 - 10(x+1) - 9\,.$$

Literatur: Hämmerlin; Hoffmann: Numerische Mathematik, 4. Auflage, Springer 1994.

4 Einführung in Maple

4.1 Grundlagen

4.1.1 Was ist Maple?

Maple ist ein kommerzielles Softwarepaket für das symbolische Rechnen; man spricht auch von einem Computeralgebra-System. Es kann z.B. algebraische Ausdrücke bearbeiten, allgemeine Lösungen von Gleichungssystemen finden, Grenzwerte von Funktionen bestimmen, Integrale auswerten oder (partielle) Ableitungen von Funktionen ausrechnen. In diesem Sinne erspart uns Maple das Nachschlagen in Formelsammlungen.

Maple rechnet mit ganzen Zahlen (bzw. rationalen Zahlen) exakt, indem es bei Bedarf mit beliebig vielen Stellen arbeitet. Dies kann allerdings zu Speicherplatz- und Rechenzeitproblemen führen.

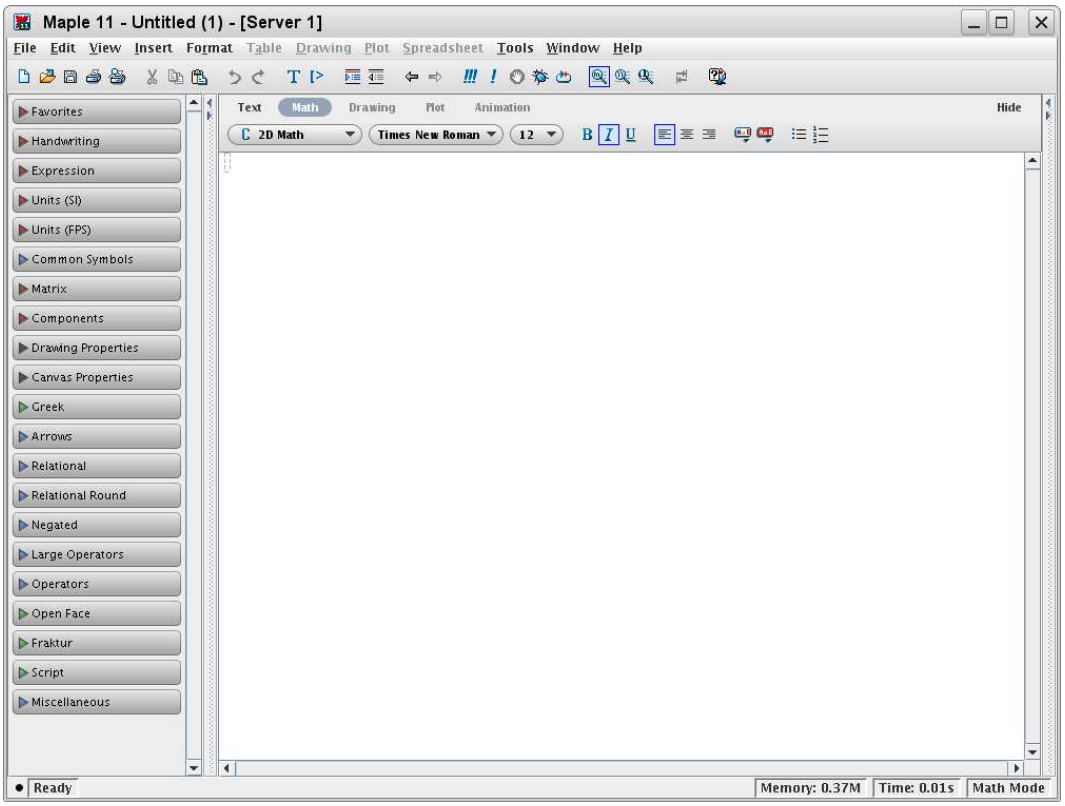
Maple besitzt auch eine gute Graphik; darauf wird hier jedoch nicht eingegangen (da wir für Graphiken Matlab verwenden).

4.1.2 Aufruf von Maple

Unter Linux startet man Maple durch Eingabe des Kommandos

```
xmaple &
```

und erhält dann ein neues Fenster:



Das Erscheinungsbild des Maple-Fensters hängt von der aktuellen Version ab, welche in der obersten Zeile (hier Maple 11) erkennbar ist.

In diesem Fenster (weiße Fläche) werden die Maple-Befehle eingegeben. Dabei können Sie mit

`View` ⟶ `Zoom Factor`

die Schriftgröße ändern.

Am linken Rand erkennen Sie einige Zusatz-Tools, z.B. `Greek` für die Eingabe von griechischen Buchstaben oder `Expressions` für die Eingabe von Integralen, Summen oder Produkten.

4.1.3 Arbeitsspeicher

Die während einer Sitzung eingegebenen Kommandos werden in einen Arbeitsspeicher (maple worksheet) geschrieben. Dieser kann vor dem Verlassen in eine Datei gespeichert werden. Dazu klicken Sie

`File` ⟶ `Save As...`

an und geben dann den Dateinamen <name> ein. Diese erhält den Zusatz `.mw` (mw = maple worksheet). Daran erkennt dann Maple, dass darin ein Maple-Sitzung gespeichert wurde und in einer späteren Maple-Sitzung durch

`File` ⟶ `Open...`

wieder geladen werden kann. Innerhalb einer Maple-Sitzung wird der Arbeitsspeicher durch

`restart`

gelöscht.

4.1.4 Maple beenden

Man beendet Maple durch Anklicken von

`File` ⟶ `Exit`

4.1.5 Online-Hilfe

Eine Online-Hilfe erhält man durch Anklicken von `help` in der Kopfzeile. In dem dann erscheinenden Menü können verschiedene Punkte ausgewählt werden. Für den Anfänger interessant ist vor allem

`Maple Help` .

Dadurch erscheint ein Inhaltsverzeichnis, und Sie können dann die entsprechenden Themen anklicken.

Informationen über einen bestimmten Befehl bekommen Sie durch Eingabe des Befehls mit vorangestelltem ?, zum Beispiel

`?diff` .

4.1.6 Eingabe von Maple-Kommandos

Maple arbeitet (ebenso wie Matlab) interaktiv, d.h. jeder Maple-Befehl wird nach Betätigen der Eingabetaste sofort ausgeführt.

Dabei ist zu beachten:

- Es wird zwischen Groß- und Kleinbuchstaben unterschieden.
- Das Zeichen # dient als Kommentarzeichen: Der Text rechts davon bis zum Zeilenende wird als Kommentar betrachtet (und somit nicht als Maple-Befehl interpretiert).
- Ein Maple-Kommando kann sich über mehrere Zeilen erstrecken, muss dann jedoch am Zeilenende mit dem Zeichen \ markiert werden. Alternativ kann auch `Shift + Enter` verwendet werden.
- Ein Maple-Kommando kann mit einem Doppelpunkt abgeschlossen werden. Dann wird das Ergebnis dieses Befehls nicht im Maple-Fenster angezeigt.
- In einer Zeile dürfen mehrere Maple-Befehle stehen. Diese müssen jedoch mit Strichpunkt (Ergebnis wird angzeigt) oder Doppelpunkt (Ergebnis wird nicht angezeigt) getrennt werden.
- ? vor einem Befehl aktiviert die Online-Hilfe zu diesem Kommando.
- % bezieht sich auf den zuvor berechneten Ausdruck.

Maple rechnet ganzzahlig und rational exakt (beliebig viele Stellen).

Beispiele:

```
> m := 5;
                                   m := 5

> p := m*m;
                                   p := 25

> q := 2^p;
                                  q := 33554432

> r := 5^(p^2);
r := 718212087483073508066162477348002474436463402065604320713967078077436641711074423037\
   6\
   725189970180627417964852836169339590849360778907259051520537600521703851886434335797\
   4\
   324172025963219925969594574290850486833464941042717610929437440253086127413158881119\
   1\
   108785652548707148071273055012503165342435647004891245928704549564107295773016903177\
   0\
   880096063325579268434790130162440139348161755465331089910013817600997754198033362627\
   0\
   294189453125

> s := r/(7^36);
s := 718212087483073508066162477348002474436463402065604320713967078077436641711074423037\
   6\
   725189970180627417964852836169339590849360778907259051520537600521703851886434335797\
   4\
   324172025963219925969594574290850486833464941042717610929437440253086127413158881119\
   1\
   108785652548707148071273055012503165342435647004891245928704549564107295773016903177\
   0\
   880096063325579268434790130162440139348161755465331089910013817600997754198033362627\
   0\
   294189453125/265173084585965347177902338160 1

>
```

Beachten Sie, dass in Maple `:=` als Wertzuweisungszeichen dient. `s` ist eine rationale Zahl. Suchen Sie den Bruchstrich!

Die Zahlen können auch in der Gleitpunktdarstellung ausgegeben werden:

```
evalf(s)
```

bzw.

```
Digits:=40
evalf(s)
```

liefert `s` als Gleitpunktzahl mit 10 Stellen (Voreinstellung) bzw. mit 40 Stellen.

4.1.7 Konstanten und Standardfunktionen

Maple kennt alle wichtigen Konstanten und Standardfunktionen. Wir geben hier nur eine kleine Auswahl an.

Konstanten

`I`	imaginäre Einheit (`I^2 = -1`)
`Pi`, `pi`	Kreiszahl π. Für (numerische) Berechnungen muss `Pi` verwendet werden. `evalf(Pi)` $\rightsquigarrow$ 3.141592654 `evalf(pi)` $\rightsquigarrow$ π
`infinity`	∞
`true`	logische Größe *wahr*
`false`	logische Größe *falsch*

Standardfunktionen

`cos(x)`	Kosinus
`sin(x)`	Sinus
`exp(x)`	Exponentialfunktion
`ln(x)`	natürlicher Logarithmus
`log10(x)`	Zehnerlogarithmus
`log[b](x)`	Logarithmus zur Basis b
`log[2](x)`	Zweierlogarithmus
`sqrt(x)`	Quadratwurzel
`root[n](x)`, `root(x,n)`	n-te Wurzel
`binomial(n,k)`	Binomialkoeffizienten $\binom{n}{k}$
`n!`	Fakultäten

Ausführliche Informationen über die Standardfunktionen erhalten Sie über die Onlinehilfe:

Help $\rightarrow$ *Maple Help* $\rightarrow$ *Mathematics* $\rightarrow$ *Basic Mathematics* $\rightarrow$ *Initially Known Functions*

4.1.8 Maple Programme

Man kann mehrere Befehle in eine Textdatei schreiben, deren Name frei wählbar ist (zum Beispiel `programm1.txt`), und dann in Maple ausführen durch das Kommando

```
read ‘programm1.txt‘
```

Achten Sie auf den richtigen Akzent. Die einzelnen Kommandos müssen dabei mit Se-

mikolon (Ergebnis erscheint auf dem Bildschirm) bzw. Doppelpunkt (keine Ausgabe auf den Bildschirm) abgeschlossen werden. Die in dieser Datei enthaltenen Maple-Befehle werden dann sequentiell abgearbeitet.

Beispiel: Die Datei `programm1.txt` habe folgenden Inhalt:

```
# Maple-Programm
x := Pi/4;
y := cos(x);
z := cos(x)^2 + sin(x)^2;
v := (x + y)/z;
evalf(v);
```

4.2 Symbolisches Rechnen

4.2.1 Ausdrücke bearbeiten

Maple bietet folgende Möglichkeiten:

`simplify(`<ausdruck>`)`	Ausdruck vereinfachen
`expand(`<ausdruck>`)`	Ausdruck ausrechnen

Beispiele:

```
z:=(5/3)^n*3^n*(x^2+2*x+1)/(x+1)
```

$$\frac{\left(\frac{5}{3}\right)^n 3^n \left(x^2+2\,x+1\right)}{x+1}$$

```
simplify(z)
```

$$(x+1)5^n$$

```
expand((x+1)^4*(x+2)^2)
```

$$x^6+8\,x^5+26\,x^4+44\,x^3+41\,x^2+20\,x+4$$

4.2.2 Faktorisierungen

Für die Faktorisierung (Primzahlzerlegung) von ganzen Zahlen gibt es das Kommando

`ifactor(`<zahl>`)` ,

zum Beispiel

```
ifactor(97518330)
```

$$(2)\,(3)^7\,(5)\,(7)^3\,(13)$$

Für die Faktorisierung von Polynomen bietet Maple die Kommandos

`factor(`<ausdruck>`)` ,
`factor(`<ausdruck>`,`<ueber>`)` .

Sind die Koeffizienten des Polynoms ganzzahlig, so wird über $\mathbb{Z}$ faktorisiert.

Beispiele:

```
factor(4*x^4-28*x^3+52*x^2+12*x-72)
```

$$4\,(x-3)^2\,(x+1)\,(x-2)$$

```
factor(x^3+3)
```

$$x^3+3$$

```
factor(x^3+3,3^(1/3))
```

$$(x^2-x\,3^{(1/3)}+3^{(2/3)})\,(x+3^{(1/3)})$$

```
factor(x^3+3,real))
```

$$(x+1.442249570)\,(x^2-1.442249570\,x+2.080083822)$$

```
factor(x^3+3,complex)
```

$$(x{+}1.442249570)\,(x{+}(-0.7211247852{+}1.249024766I))\,(x{+}(-0.7211247852{-}1.249024766I))$$

4.2.3 Summen, Reihen und Produkte

Maple kennt dafür die Kommandos

`sum(`<ausdruck>`,`<bereich>`)` ,
`product(`<ausdruck>`,`<bereich>`)` .

Beispiele:

```
sum(1/k,k=1..20)
```

$$\left[\ \text{berechnet}\ \sum_{k=1}^{20}\frac{1}{k}\ \right]$$

$$\frac{55835135}{15519504}$$

```
sum(1/(k!),k=0..infinity)
```

$$\left[\ \text{berechnet}\ \sum_{k=0}^{\infty}\frac{1}{k!}\ =\ e\right]$$

```
sum(k^2,k=1..n)
```

$$\left[\ \text{berechnet}\ \sum_{k=1}^{n}k^2\ \right]$$

$$\frac{1}{3}(n+1)^3-\frac{1}{2}(n+1)^2+\frac{1}{6}n+\frac{1}{6}$$

```
simplify(%)
```

$$\frac{1}{3}n^3+\frac{1}{2}n^2+\frac{1}{6}n$$

```
product(2^k,k=1..8)
```

$$\left[\ \text{berechnet}\ \prod_{k=1}^{8}2^k\ \right]$$

$$68719476736$$

Selbstverständlich dürfen diese Kommandos auch geschachtelt werden:

```
sum(product(1/i^2,i=1..k),k=2..6)
```

$$\left[\ \text{berechnet}\ \sum_{k=2}^{6}\prod_{i=1}^{k}\frac{1}{i^2}\ \right]$$

$$\frac{144937}{518400}$$

4.2.4 Grenzwerte

Die Berechnung von Grenzwerten erfolgt mit dem Befehl

`limit(`<ausdruck>`,`<stelle>`)`

Beispiele:

```
limit(sin(x)/x,x=0)
```

$$\left[\ \text{berechnet}\ \lim_{x\to 0}\frac{\sin(x)}{x}\ \right]$$

$$1$$

```
limit(exp(-x^2),x=infinity)
```

$$\left[\ \text{berechnet}\ \lim_{x\to\infty}\exp(-x^2)\ \right]$$

$$0$$

4.2.5 Umgang mit Funktionen

Funktionen werden in Maple wie folgt definiert:

```
f := x -> sin(x)/x
```

$$f := x \to \frac{\sin(x)}{x}$$

```
g := (x,y) -> sqrt(x^2 + 3*y^4)
```

$$g := (x,y) \to \sqrt{x^2+3y^4}$$

Beim Funktionsaufruf haben wir zwei Möglichkeiten: *symbolisch* oder *numerisch*.

`f(2)`	$\rightsquigarrow$	$\frac{1}{2}\sin(2)$
`f(2.0)`	$\rightsquigarrow$	0.4546487134
`evalf(f(2))`	$\rightsquigarrow$	0.4546487134
`g(2,1/2)`	$\rightsquigarrow$	$\frac{1}{4}\sqrt{67}$
`g(2.0,0.5)`	$\rightsquigarrow$	2.046338193

Auch eine Verkettung (Komposition) von Funktionen ist möglich, zum Beispiel

`(sin @ exp)(x)`	$\rightsquigarrow$	$\sin(\exp(x))$
`(ln @@ 2)(x)`	$\rightsquigarrow$	$\ln(\ln(x))$

4.2.6 Ableitungen

Für die Berechnung von (partiellen) Ableitungen bietet Maple zwei Möglichkeiten. In den folgenden Beispielen seien f und g die in obigem Abschnitt definierten Funktionen.

1. Verwendung des `diff` - Kommandos.

```
diff(f(x),x)
```
[berechnet $f'(x)$]

$$\frac{\cos(x)}{x} - \frac{\sin(x)}{x^2}$$

```
diff(f(x),x,x)
```
[berechnet $f^{(2)}(x)$]

$$-\frac{\sin(x)}{x} - \frac{2\cos(x)}{x^2} + \frac{2\sin(x)}{x^3}$$

```
diff(f(x),x$5)
```
[berechnet $f^{(5)}(x)$]

```
diff(exp(-x^2),x)
```

$$-2x\,e^{(-x^2)}$$

```
diff(g(x,y),x)
```
[berechnet $g_x(x,y)$]

$$\frac{x}{\sqrt{x^2+3y^4}}$$

```
diff(g(x,y),y,x)
```
[berechnet $g_{yx}(x,y)$]

Es gibt auch ein `Diff` - Kommando, das den zu differenzierenden Ausdruck hinschreibt:

```
Diff(sin(x),x) = diff(sin(x),x)
```

$$\frac{d}{dx}\sin(x) = \cos(x)$$

```
Diff(sqrt(x^2 + y^2),x) = diff(sqrt(x^2 + y^2),x)
```

$$\frac{\partial}{\partial x}(\sqrt{x^2+y^2}) = \frac{x}{\sqrt{x^2+y^2}}$$

2. Verwendung des Differentialoperators `D`

```
D(f)
```
[berechnet $f'(x)$]

```
D(f)(1)
```
[berechnet $f'(1)$]

```
(D@@3)(f)
```
[berechnet $f^{(3)}(x)$]

```
D(x->exp(x^2))
```

$$x \to 2x\,e^{(x^2)}$$

`D[1](g)` [berechnet $g_x(x,y)$]

`D[2](g)` [berechnet $g_y(x,y)$]

`D[1,2](g)` [berechnet $g_{xy}(x,y)$]

`D[1](g)(3,4)` [berechnet $g_x(3,4)$]

4.2.7 Integrale

1. Unbestimmte Integrale (Stammfunktionen)

```
int(exp(-t),t)
```
[berechnet Stammfunktion zu e^{-t}]

$$-e^{(-t)}$$

```
f := x -> sin(x)*cos(x)
int(f(x),x)
```

$$\frac{1}{2}\sin(x)^2$$

2. Bestimmte Integrale

```
int(exp(-t),t=0..2)
```
[berechnet $\int_0^2 e^{-t}\,dt$]

$$-e^{-2}+1$$

```
int(exp(-t),t=0.0..2.0)
```

$$0.8646647168$$

```
int(exp(-x^2),x=-infinity..infinity)
```
[berechnet $\int_{-\infty}^{\infty} e^{-x^2}\,dx$]

$$\sqrt{\pi}$$

```
Int(exp(-x^2),x=-infinity..infinity)
```

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx$$

Beachten Sie den Unterschied zwischen `int` und `Int`.

4.2.8 Taylor-Polynome

Die Taylor-Entwicklung einer (hinreichend oft differenzierbaren) Funktion erhält man durch

`taylor(`<ausdruck>`,`<stelle>`,`<ordnung>`)` .

Dabei enthält <stelle> den Entwicklungspunkt und die natürliche Zahl <ordnung> die Ordnung des Restglieds, d.h. der Grad des Taylor-Polynoms ist <ordnung>-1.

Beispiele:

```
taylor(exp(x),x=0,4)
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + O(x^4)$$

```
taylor(exp(x),x=pi,3)
```

$$e^{\pi} + e^{\pi}(x-\pi) + \frac{1}{2}e^{\pi}(x-\pi)^2 + O((x-\pi)^3)$$

Das Kommando

`convert(%,polynom)`

gibt den zuletzt berechneten Ausdruck als Polynom aus:

$$e^{\pi} + e^{\pi}(x-\pi) + \frac{1}{2}e^{\pi}(x-\pi)^2$$

Der Befehl

`convert(`<ausdruck>`,`<form>`)`

stellt einen Ausdruck in der gewünschten Form dar.

Beispiele:

```
convert(2011,binary)
```

$$11111011011$$

```
convert(2013,hexadecimal)
```

$$7DD$$

```
convert(101101,decimal,binary)
```

$$45$$

```
convert(AFFE,decimal,hex)
```

$$45054$$

```
convert(1.2345,fraction)
```

$$\frac{2469}{2000}$$

4.2.9 Gleichungen

Zur Lösung von Gleichungssystemen gibt es das Kommando

`solve(`<gleichungen>`,`<variablen>`)` .

Beispiele:

1. Die Lösungen von $4x^4 + 8x^2 = 10$ erhält man durch

```
pol := 4*x^4 + 8*x^2 = 10
solve(pol,x)
```

Möglich ist auch

```
solve(4*x^4 + 8*x^2 = 10,x)
```

$$-\frac{1}{2}\sqrt{-4+2\sqrt{14}},\ \frac{1}{2}\sqrt{-4+2\sqrt{14}},\ -\frac{1}{2}\sqrt{-4-2\sqrt{14}},\ \frac{1}{2}\sqrt{-4-2\sqrt{14}}$$

```
evalf(%)
```

$$-.9331820260, 0.9331820260, -1.694351998I, 1.694351998I$$

2. Die Lösungen des Gleichungssystems

$$\begin{aligned} x^2+y^2 &= 4 \\ (x-1)^4+(y+1)^2 &= 10 \end{aligned}$$

erhalten wir durch folgende Maple-Sequenz:

```
e1 := x^2 + y^2 = 4
e2 := (x-1)^4 + (y+1)^2 = 10
solve({e1,e2},{x,y})
evalf(%)
```

$$\{\, y = 2,\ x = 0 \,\},\ \{\, x = 2.928323177 + 0.1903946038\, I,\ y = 0.259774844 - 2.146231461\, I \,\}$$

4.2.10 Mengen

Mengen werden durch die üblichen Mengenklammern { } dargestellt:

```
A := {rot, blau, gelb}
B := {grün, braun, grau, gelb}
```

Vereinigung $A \cup B$, Durchschnitt $A \cap B$ und Differenz $A \setminus B$ erhalten wir durch

```
A union B
```

$$\{\, grau,\ braun,\ grün,\ blau,\ rot,\ gelb \,\}$$

```
A intersect B
```

$$\{\, gelb \,\}$$

```
A minus B
```

$$\{\, rot,\ blau \,\}$$

Möglich ist auch die Abfrage, ob ein Element zur Menge gehört. Das Ergebnis ist *true* oder *false*.

```
farbe := rot
member(farbe,A)
```

true

4.3 Vektoren und Matrizen

4.3.1 Vektoren

Für die Definition von Vektoren stellt Maple zwei Möglichkeiten zur Verfügung.

1. Möglichkeit:

```
v := <1,2,3>
```
(Spaltenvektor)

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
w := <1|2|3>
```
(Zeilenvektor)

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

2. Möglichkeit: Verwendung des Befehls `Vector`

```
v := Vector([1,2,3])
```

$$v := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
w := Vector[row]([1,2,3])
```

$$w := \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

```
Vector(3)
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

```
Vector[row](1..5,2)
```

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

4.3.2 Matrizen

Für Matrizen haben wir ebenfalls zwei Optionen.

1. Möglichkeit:

```
A := <<1|2|3>,<4|5|6>,<7|8|9>>
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

2. Möglichkeit: Verwendung des Befehls `Matrix`

```
A := Matrix([[1,2,3],[4,5,6],[7,8,9]])
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
B := Matrix(2,3)
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
C := Matrix(1..2,1..3,2)
```

$$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

Der Zugriff auf die einzelnen Elemente bzw. auf Teilbereiche der Matrix erfolgt durch

```
u : = A[2,3]
A[1..2,2..3] := Matrix([[1,1],[1,1]])
```

4.3.3 Umgang mit Matrizen

Für den Umgang mit Matrizen (Multiplikation, Eigenwerte, Determinanten,...) braucht man ein Zusatzpaket, welches durch das Kommando

```
with(LinearAlgebra)
```

bereitgestellt wird. Nach diesem Aufruf erhalten Sie eine Liste mit den nun zusätzlich verfügbaren Befehlen.

Beispiele:

`MatrixVectorMultiply(A,v)`	Av (Matrix mal Spaltenvektor)
`VectorMatrixMultiply(w,A)`	wA (Zeilenvektor mal Matrix)
`MatrixMatrixMultiply(A,B)`	AB (Matrixprodukt)
`Eigenvalues(A)`	Eigenwerte von A
`Eigenvectors(A)`	Eigenwerte und Eigenvektoren von A
`Determinant(A)`	Determinante von A
`Transpose(A)`	Transponierte A^T
`MatrixInverse(A)`	inverse Matrix A^{-1}

Zum Schluss noch ein kleines Maple-Programm. Testen Sie es mal.

```
#  Maple-Programm
with(LinearAlgebra);
A := <<1|2|3>,<0|4|5>,<0|0|6>>;
v := <1,2,3>;
c := MatrixVectorMultiply(A,v);
Eigenwerte := Eigenvalues(A);
Determinante := Determinant(A);
Transponierte := Transpose(A);
Inverse := MatrixInverse(A);
```