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

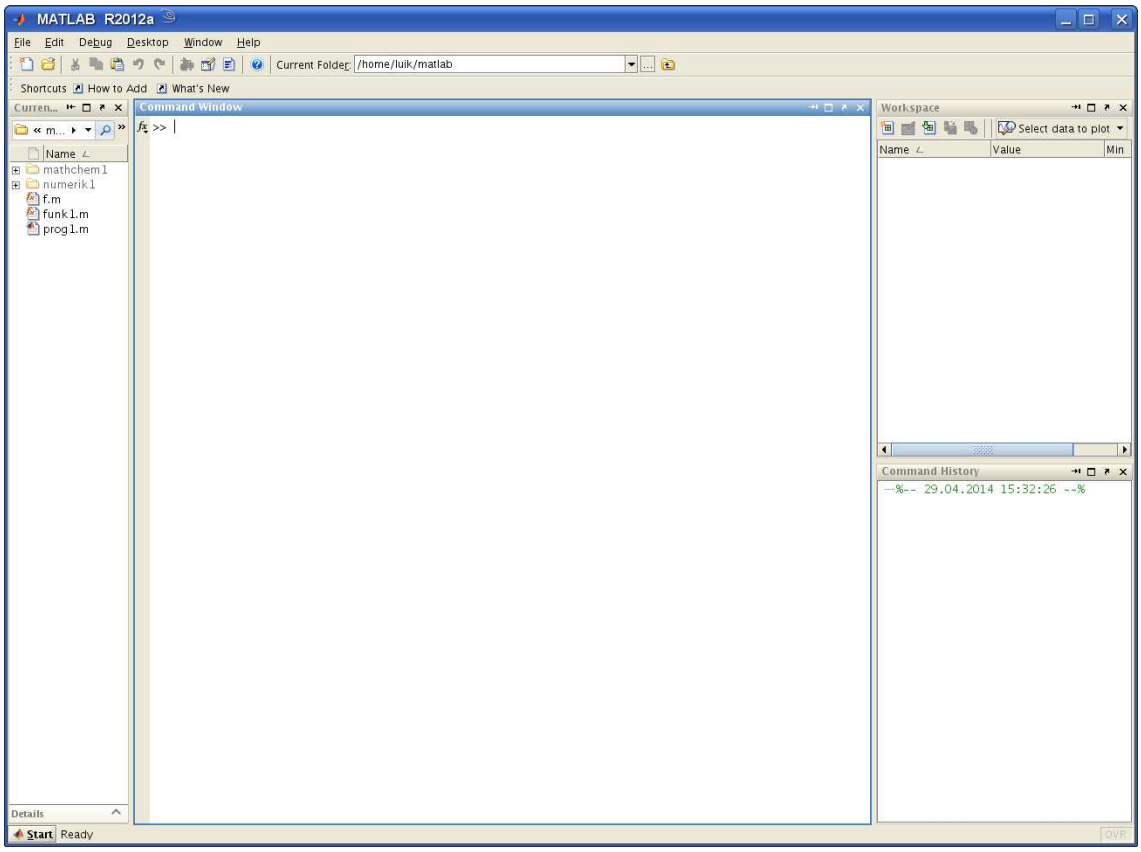
Command Window

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 den Menüpunkt **File** an und wählen dann (mit der linken Maustaste) in dem erscheinenden Untermenü den Punkt **Exit MATLAB** 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

```
helpdesk
```

im Command Window. Es wird dann ein neues Fenster **Help** eröffnet. Um dieses Fenster wieder zu schließen, klicken wir mit der linken Maustaste den Menüpunkt **File** und dann den Unterpunkt **Close Help** an.

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 aus-

geführt (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.

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 **File**. In dem erscheinenden Untermenü wird zuerst **New** und dann **M-file** angeklickt. Dadurch wird eine neues Fenster eröffnet, und wir können nun das Programm eintippen. Soll ein bereits existierendes Matlab-Skript geändert werden, so klicken wir statt **New** den Menüpunkt **Open...** an und geben dann den Dateinamen ein.

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

   `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

File $\longrightarrow$ **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 im linken oberen 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 **File** 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 **File** 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 **Edit** 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 $x = (0, 0.1, 0.2, \ldots, 1.9, 2.0)$.

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 mit der Überschrift **Array Editor** 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. Zum Beenden klickt man

File $\longrightarrow$ **Close Array Editor** .

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 dabei die reelle Zahl 2 zu einer Matrix, welche dieselbe Größe wie A hat, und deren Elemente alle den Wert 2 haben.

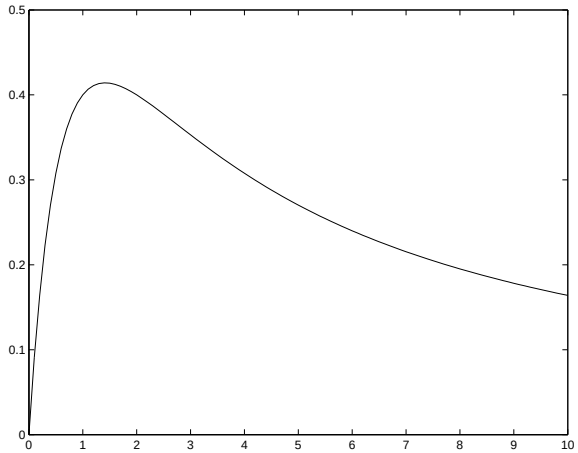
Beispiel: Für die Funktion

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

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

`max(x)`	$(= \max\{x_i : i = 1, \ldots, n\})$
`min(x)`	$(= \min\{x_i : i = 1, \ldots, n\})$
`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)`	$(\|x\|_\infty = \max\{\|x_i\| \ : \ i = 1, \ldots, n\})$

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

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

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.

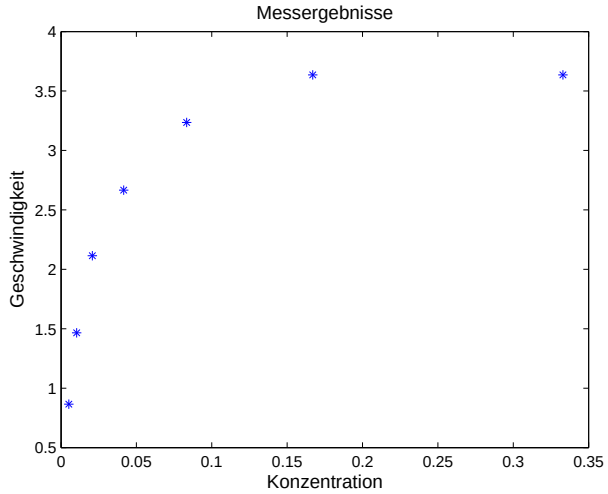
Beispiel: Die Datei `ergebnisse.dat` habe folgende Gestalt:

```
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 also 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

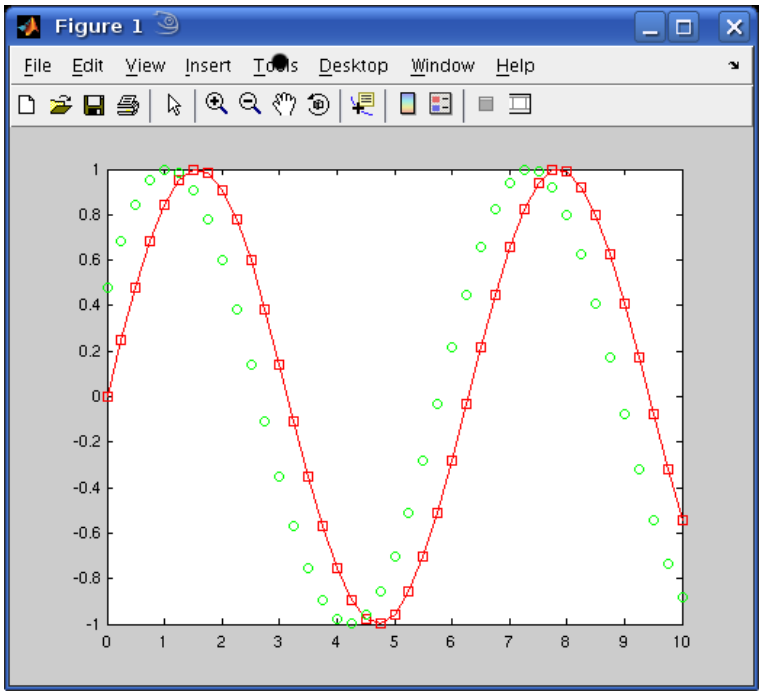
Farben
Die Auswahl der Farbe erfolgt durch:

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.

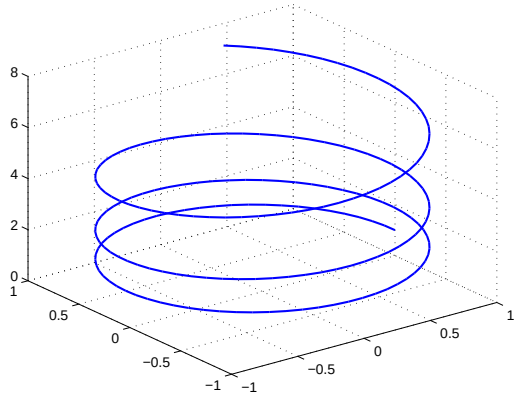
Im Matlab verwendet man zum Zeichnen von Raumkurven das Kommando

```
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

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

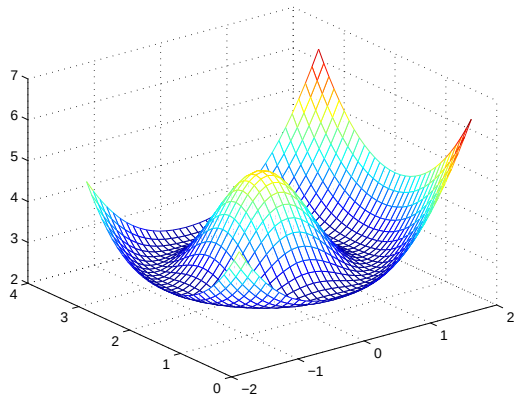
mit Definitionsbereich $D = [-1.5, 2] \times [0.5, 3.5]$.

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:

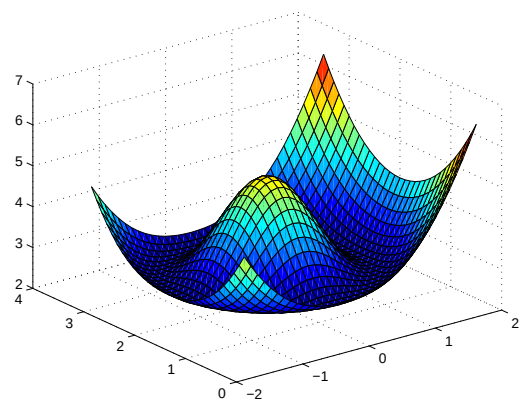
1. `mesh(X,Y,Z)`

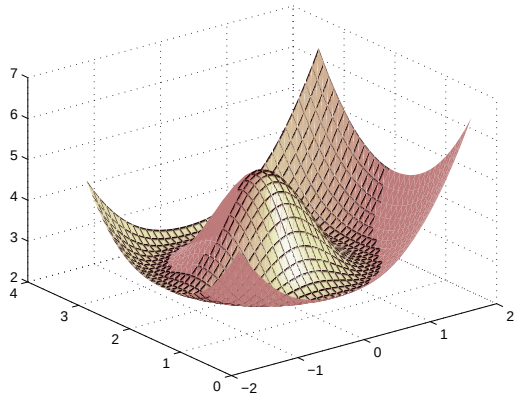


2.
```
surf(X,Y,Z);
colormap(jet);
```

Der 2. Befehl wählt eine Farbpalette aus.

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



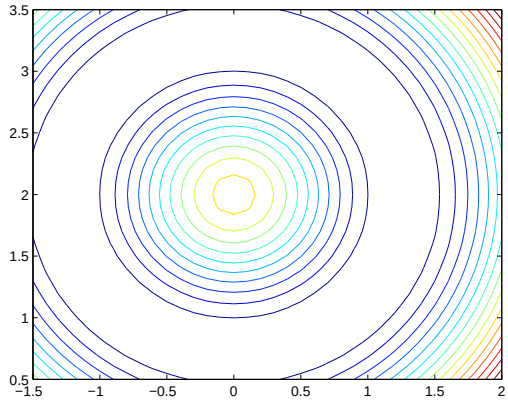


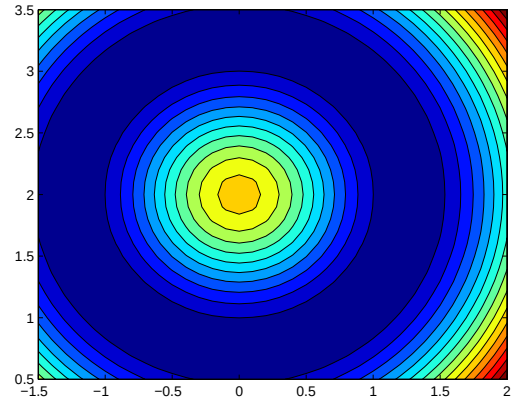
Eine weitere graphische Darstellung von $f : D \subset \mathbb{R}^2 \to \mathbb{R}$ ist die Höhenkarte. Auch dafür bietet Matlab Möglichkeiten:

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')
```

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);
```

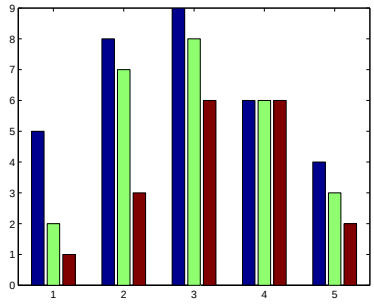
2.4.4 Balken- und Kuchendiagramme

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

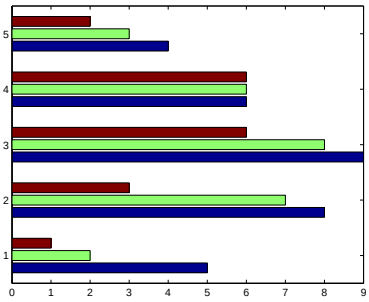
```
Y = [5 2 1; 8 7 3; 9 8 6; 6 6 6; 4 3 2];
```

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

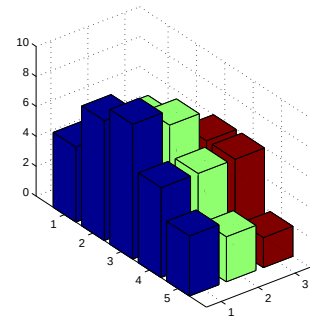
```
bar(Y);
```



```
barh(Y);
```

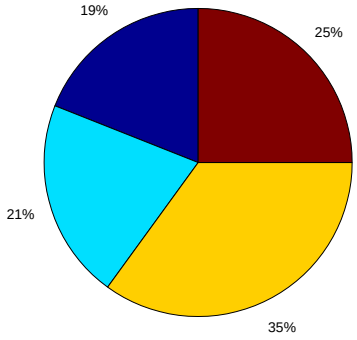


```
bar3(Y);
```

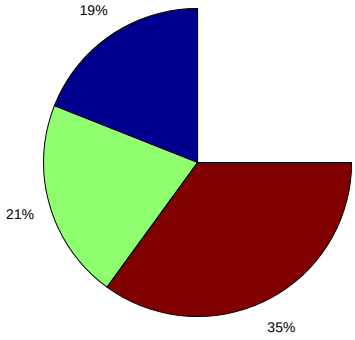


Für Kuchendiagramme werden die Anteile in einem Vektor zusammengefasst und dann mit dem Matlab-Befehl `pie` gezeichnet, z. B.

```
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

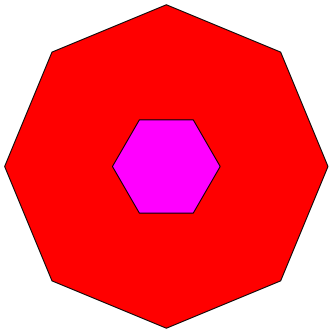
Ein geschlossener Polygonzug kann mit einer Farbe gefüllt werden. Das Matlab-Kommando dazu lautet

```
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]);
```

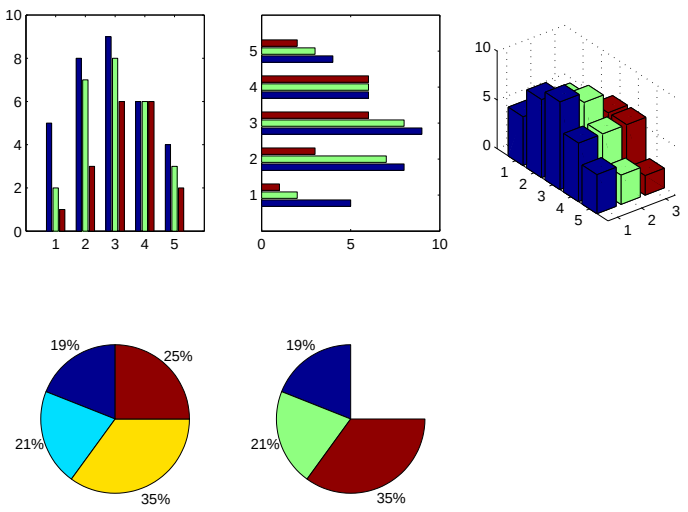


2.4.6 Unterbilder

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 in 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 über

`helpdesk` $\longrightarrow$ `search` .

In dem dann erscheinenden Feld *search for* wird

```
precedence level
```

eingegeben.

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(' sigma(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
```

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

Dabei ist `abs(z)` der Betrag einer komplexen Zahl. Die zweite 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
```

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))
```

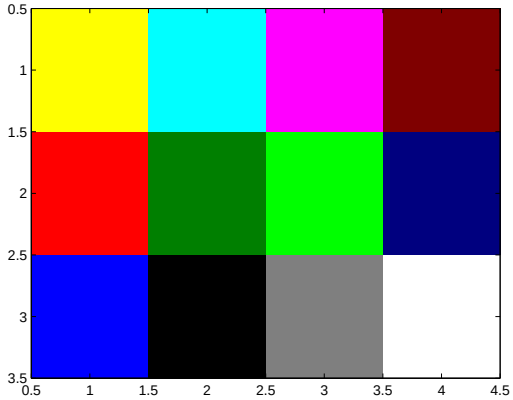
(wobei N die Anzahl der Farben bezeichnet) festgelegt.

```
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

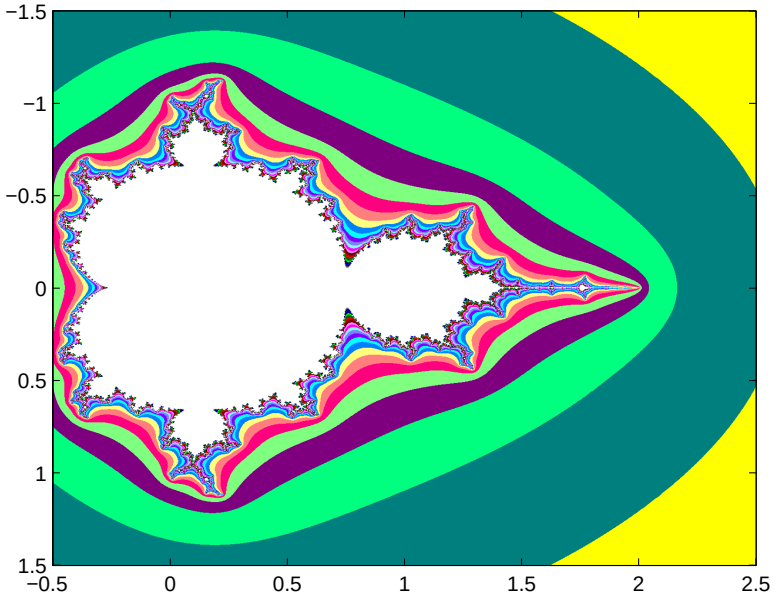
$$z_0 \;=\; 0\,, \qquad z_{k+1} \;=\; z_k^2 - c \quad (k = 0, 1, 2, \ldots) \text{ für } c \in \mathbb{C}.$$

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.

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

```
% Einbinden von eps-Dateien
\documentclass[12pt]{article}
\usepackage{german}

\usepackage{epsf}

\usepackage[utf8]{inputenc}
%\usepackage[latin1]{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

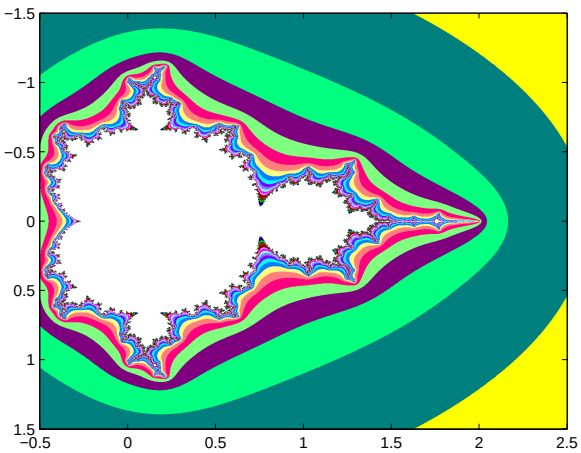
`latex` <dateiname>

(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 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
```