Kapitel 2

Basis Syntax in MATLAB

2.1 Variablen und Zuweisung von Werten

Neben der Möglichkeit MATLAB als eine Art überdimensionalen Taschenrechner zu benutzen

```
3*(5 + 8)
3 + sin(pi/3)
```

kann man Ergebnisse auch benannten Größen (Variablen) zuweisen. Das Zeichen für Zuweisung (assignment) ist das = Zeichen. Wird kein = verwendet, wird die Rechnung durchgeführt und das Ergebnis auf der Variablen `ans` (answer) gespeichert. Gespeicherte Größen können in der Folge für weitere Berechnungen herangezogen werden.

```
a = 3 * (5 + 8)
a = (a - 1) / 4
b = sin(0.5)
res_1 = (b + 1) / b - 1;      % Was ist der Unterschied
res_2 = (b + 1) / (b - 1);    % zwischen den zwei Zeilen?
```

Wichtig dabei ist, dass Größen die rechts vom = Zeichen stehen durch vorige Berechnungen bekannt sind und gültige Operatoren (z.B.: +, −, . . .), bzw. gültige Programmnamen (z.B.: `sin`) enthalten.

Wird ein Name wieder auf der linken Seite einer Zuweisung verwendet (`a` in der zweiten Zeile) wird zuerst die rechte Seite mit dem alten Wert von `a` berechnet und dann dieser Wert der Variablen zugewiesen. Die alte Bedeutung geht dabei dann verloren.

Der Strichpunkt am Ende einer Programmanweisung regelt nur die Ausgabe am Schirm und hat nichts mit der Berechnung an sich zu tun.

Insgesamt muss natürlich auch die Sprachsyntax (Regelwerk, Grammatik) korrekt sein. Insbesonders müssen alle Klammern geschlossen sein und Operatoren und Funktionen richtig verwendet werden.

Ein riesiger Vorteil von MATLAB ist, dass viele Befehle direkt auf ganze Felder bzw. Matrizen angewandt werden können.

So berechnet der Befehl

```
s  = sin( [0.0, 0.1, 0.2, 0.3] )
```

den Sinus aller vier Werte und speichert ihn in der Variablen `s`. Zusammen mit der entsprechenden Syntax für die Konstruktion von Feldern ermöglicht dies eine sehr elegante und auch effiziente Programmierung:

```
x   = [-1:0.1:1];
y_1 = sinh(x); y_2 = cosh(x);
plot(x,y_1,x,y_2)
```

Einige einfache Fehler und entsprechende Fehlermitteilungen von MATLAB, die immer in der Farbe rot im Kommando-Fenster ausgegeben werden, finden sich in der folgenden Tabelle.

FEHLER	FEHLERMELDUNG
`a = 3+`	`Error: Expression or statement is incomplete or incorrect.`
`a = (3+4`	`Error: Expression or statement is incorrect-possibly unbalanced (, \{, or \[.`
`a = sin()`	`Error: Error using ==> sin Not enough input arguments.`
`sin(1,2,3)`	`Error using ==> sin Too many input arguments.`
`sin(1]`	`Error: Unbalanced or misused parentheses or brackets.`
`a = six(1)`	`Undefined command/function "six".`
`s = 'Winfried`	`Error: A MATLAB string constant is not terminated properly.`
`3a = sin(1)`	`Error: Unexpected MATLAB expression.`
`3*a = sin(1)`	`Error: The expression to the left of the equals sign is not a valid target for an assignment.`
`a = b = 5`	`Error: The expression to the left of the equals sign is not a valid target for an assignment.`
`a = 5 .+ 2`	`Error: Unexpected MATLAB operator.`

Gültige Variablennamen in MATLAB müssen mit einem Buchstaben beginnen und dürfen nur Buchstaben, Zahlen und als einziges Sonderzeichen den Untersstrich _ verwenden. Groß- und Kleinbuchstaben werden zumindest unter LINUX unterschieden.

Die Verwendung von Umlauten ist unter WINDOWS möglich, sollte aber vermieden werden, da solche Programme unter LINUX dann nicht funktionieren. Äusserst ungeschickt ist es auch Namen von MATLAB-Funktionen und vordefinierten Konstanten 2.2 zu verwenden.

GÜLTIG	UNGÜLTIG	UNGESCHICKT
`a a3 a_3`	`3a 3_a a*`	`Maß ö3 ö_3`
`Sin SIN MAX`	`Sin( Sin() Max+`	`sin max abs`
`k l m`	`"k" k! k#`	`i j pi`
`Resultat_1`	`Resultat[1]`	`Lösung`

Werden MATLAB-Befehle als Namen für Variablen verwendet, schafft man ein Problem dadurch, weil diese neue Bedeutung in der Hirachie höher liegt als die ursprüngliche Bedeutung, d.h., dass die neue Bedeutung Vorrang hat. Schreibt man z.B. `sin=5`, verliert `sin` die Bedeutung als Sinus-Funktion und liefert immer den Wert `5`.

Manchmal ist die Gefahr, dass dies unendeckt bleibt sehr groß:
`sin(1)` liefert dann einfach `5` statt `0.8415`
Manchmal treten Fehler auf:

```
sin(2)
 Error: Index exceeds matrix dimensions.
sin(pi)
 Error: Subscript indices must either be real
 positive integers or logicals.
```

Behoben werden kann das Problem nur durch das Löschen der Variablen (`clear`):
`clear sin` oder `clear('sin')`

Gefährlich ist auch folgender Fall: Durch den Befehl `i=1` verliert die Variable `i` ihre Bedeutung als imaginäre Konstante und `3+2*i` liefert den falschen Wert `5`.

Diejenigen Fehler, die eine Beendigung des Programms zur Folge haben und die damit eine Error-Meldung liefern, sind grundsätzlich leichter zu finden und zu beheben als Fehler, die den Programmablauf nicht stoppen, sondern nur zur Folge haben, dass falsch weiter gerechnet wird.

2.2 Mathematische Konstanten

In MATLAB sind eine Reihe von mathematischen Konstanten vordefiniert. Ihre Namen und ihre Bedeutung findet sich in der nachfolgenden Tabelle. Sie sollen auf kei-

nen Fall überschrieben werden. Wenn Sie mit komplexen Zahlen rechnen, vermeiden Sie daher unbedingt die Verwendung von `i` und `j` als Zählvariablen in Schleifen.

KONSTANTE	BEDEUTUNG	WERT
`eps`	Relative Genauigkeit von Fließkommarechnungen.	$2.2204 \cdot 10^{-16}$
`i`, `j`	Imaginäre Einheit.	$\sqrt{-1}$
`inf`, `Inf`	Unendlich.	∞
`nan`, `NaN`	Not A Number - Keine Zahl im herkömmlichen Sinn. Entsteht z.B. durch $\frac{0}{0}$, $\frac{\infty}{\infty}$, $0 \cdot \infty$ und durch jede Operation mit `nan`.	nan
`pi`	Verhältniss zwischen Umfang des Kreises und seines Durchmessers.	3.14159
`realmax`	Größte positive Fließkommazahl.	$1.79769 \cdot 10^{+308}$
`realmin`	Kleinste positive Fließkommazahl.	$2.22507 \cdot 10^{-308}$
`intmax`	Größte ganze Zahl (`int32`).	2147483647
`intmin`	Kleinste ganze Zahl (`int32`).	-2147483648

Die Eulersche Zahl e ist in MATLAB nicht definiert. Braucht man sie, dann muss man sie durch `e=exp(1)` selbst definieren. Für weitere Rechnungen benötigt man das aber nicht wirklich, da die Berechnung von e^x in MATLAB besser mit `exp(x)` und nicht mit mit einer `Potenz` von e (`exp(1).^x`) erfolgt.

Die vorliegende Liste ist beschränkt auf die am häufigsten verwendeten numerischen Datentypen in MATLAB, `double` und `int32` (32 Bit Integer mit Vorzeichen).

Für Fließkommazahlen gibt es neben dem Datentyp `double` (8 Byte) auch den Datentyp `single` (4 Byte). Will man die Fließkommazahlen im Datentyp `single`, muss man schreiben `eps('single')`, `inf('single')`, `nan('single')`, bzw. `realmax('single')` und `realmin('single')`.

Für `pi` und andere Zahlen muss man die Befehlsform von `single`, nämlich `single(pi)` verwenden. Die numerischen Datentypen sind in Kapitel 8 zusammengefasst. Die ganzzahligen Datentypen gibt es von `int8` (1 Byte) bis `int64` (8 Byte). Die richtige Verwendung hier wäre also z.B., `intmax('int8')`.

2.3 Wichtige Befehle

In der folgenden Tabelle sind einige für das Arbeiten mit MATLAB wichtige bzw. paraktische Befehle aufgelistet.

BEFEHL	BEDEUTUNG
`quit`	Beendet MATLAB.
`exit`	Beendet MATLAB.
`clc`	Löscht MATLAB-Kommandofenster.
`home`	Löscht MATLAB-Schirm und positioniert den Cursor links oben (man kann aber den Balken verwenden um alte Inhalte zu sehen.
`diary`	Schreibt Befehle und Ergebnisse in einem File mit.
`save`	Speichert Inhalte des Workspaces in einem File.
`format`	Beeinflusst Outputformat.
`system`	Führt Befehl im zugrundeliegenden Betriebssystem aus.
`clear`	Löscht Variable im Workspace (z.B.: `clear all`).
`close`	Schließt Graphikfenster (z.B.: `close all`).
`who`, `whos`	Listed Variablen im Workspace auf.
`exist`	Überprüft, ob ein Name bereits existiert.

2.4 Möglichkeiten für Hilfe in MATLAB

MATLAB bietet eine Reihe von Möglichkeiten Hilfe zu Befehlen bzw. zur Syntax der Programmiersprache zu finden. Hier sind einige wichtige aufgelistet, die sich als äussert praktisch erwiesen haben.

BEFEHL	BEDEUTUNG
`helpbrowser`	Startet den Browser für das ausführliche Helpsystem von MATLAB.
`help`	Zeigt MATLAB-Hilfe im Kommandofenster. `help help`: Hilfe über den Hilfebefehl. `help sin`: Hilfe für die Funktion Sinus.
`doc`	Öffnet Hilfeseite im Browser. `doc sin`: Hilfe für die Funktion Sinus.
`lookfor`	Sucht nach speziellen Schlüsselwörtern im Hilfesystem. `lookor hyperbolic`: Listet alle Befehle, wo in der Hilfe das Schlüsselwort vorkommt.

2.5 Spezielle Zeichen - Special Characters

Um eine Programmiersprache wie MATLAB korrekt benutzen zu können, muss man die Bedeutung von speziellen Zeichen kennen.

2.5.1 Klammern

2.5.1.1 Runde Klammern - Parenthesis

Runde Klammern erfüllen in MATLAB im Wesentlichen eine Abgrenzungsfunktion zwischen arithmetischen Ausdrücken (Gliederung), zwischen Feldname und Indices (Indizierung), bzw. zwischen Funktionsnamen und Argumenten. Einige Beispiele sind in folgender Tabelle zusammengefasst.

KLAMMER	BEDEUTUNG	BEISPIEL
`()`	Gliederung arithmetischer Ausdrücke	`3*(a+b)`

	Indizierung von Feldern
	Ein Element (Zeile, Spalte)
	Mehrere Elemente
		`a(3)`
		`a(1,3)`
		`a([1,3,5])`

	Abgrenzung von Argumenten
	Übergabeparameter an Funktionen
		`sin(a)`
		`plot(x,y1,x,y2)`
		`plus(3,4)`

2.5.1.2 Eckige Klammern - Brackets

Mit Hilfe von eckigen Klammern werden in MATLAB Vektoren und Matrizen erzeugt bzw. zusammengefügt. Ausserdem werden sie bei der Rückgabe von Funktionswerten verwendet, wenn es mehrere Ergebnisse gibt. Einige Beispiele sind in folgender Tabelle zusammengefasst.

KLAMMER	BEDEUTUNG	BEISPIEL

`[]`	Erzeugen von Vektoren
	(auch ohne Beistrich)
	Bereich (von - bis)
	Schrittweite
	Leeres Feld
		`a=[1,2,3,4,5]`
		`a=[1 2 3 4 5]`
		`a=[1:5]`
		`b=[1:2:5]`
		`c=[]`

	Erzeugung von Matrizen
	Nebeneinander
	Übereinander
	Zeichenketten
		`[1,2,3;4,5,6]`
		`[a,a]`
		`[a;a]`
		`['ich',' ','bin']`

	Mehrere Ausgabewerte	`[a,b,c]=func(x,y,z)`

2.5.1.3 Geschwungene Klammern - Curly Braces

Geschwungene Klammern werden in MATLAB für die Erzeugung und Indizierung von Zellen verwendet. Zellen sind Felder, die an jeder Stelle beliebige Elemete (Felder, Zeichenketten, Strukturen) und nicht nur Skalare enthalten können.

KLAMMER	BEDEUTUNG	BEISPIEL
`{}`	Erzeugen von Zellen Leere Zelle	`z={[1:3],'string'}` `l={}`
	Zugriff auf Zellelemente Zuweisung	`a=z{1}` `z{3}=[1,2;3,4]`

2.5.2 Punkt - Dot

Punkte haben eine vielfältige Bedeutung in MATLAB, wobei die wichtigste wohl der Dezimalpunkt ist:

ZEICHEN	BEDEUTUNG	BEISPIEL
`.`	Dezimalpunkt auch in Fließkommazahlen $1.5 \cdot 10^{-5}$	`p=3.14` `1.5e-5`
`.`	Zugriff auf Strukturelemente	`s.f` `s.('f')`
`..`	Übergeordnetes Verzeichnis	`cd ..`
`...`	Fortsetzungszeile	`m=[1,2; ...` `3,4]`
`.* ./ .\ .^`	Operator für alle Elemente z.B.: Quadrieren	`[1,2,3].*[4,5,6]` `[1,2,3].^2`
`.'`	Transponieren 2.5.5	M.'

2.5.3 Komma und Strichpunkt - Comma and Semicolon

Komma und Strichpunkt fungieren im Wesentlichen als Trennzeichen:

ZEICHEN	BEDEUTUNG	BEISPIEL
`,`	Trennzeichen - Spalte	`[1,2,3]`
`;`	Trennzeichen - Zeile Spaltenvektor	`[1,2,3;4,5,6]` `[1;2;3]`
`,`	Trennzeichen - Index höhere Dimension	`a(3,4)` `a(m,n,o,p,q)`
`,`	Trennzeichen - Funktion auch bei Output	`plus(3,4)` `[a,b]=func(x,y)`
`,`	Trennzeichen - Kommando mit Ausgabe	`a=[1,2], b=5` `a=3,b=a,c=a`
`;`	Trennzeichen - Kommando ohne Ausgabe	`a=[1,2]; b=5;` `a=3;b=a;c=a;`

Hier gibt es eine interessante Fehlermöglichkeit, nämlich die Verwechslung von . (Punkt) mit , (Komma) als Dezimalzeichen. Das MATLAB-Kommando

```
a = 3,4
```

liefert keine Fehlermitteilung, sondern setzt `a=3`, zeigt es wegen des Kommas am Schirm an, setzt die Variable `ans=4` und zeigt sie ebenfalls am Schirm an.

Anmerkung: Die Variable `ans` wird immer für das letzte Resultat verwendet, wenn keine explizite Zuweisung erfolgt.

MATLAB kann in diesem Fall keine Fehlermitteilung anzeigen, da es sich um eine korrekte Eingabe handelt, die "nur" etwas anderes berechnet, als sich der Benutzer vielleicht erwartet.

Fehler, die sich auch öfters ergeben, sind hier mit Fehlermitteilungen angeführt:

FEHLER	FEHLERMELDUNG
`a = [1,2;3,4`	`Error: "]" expected, "End of Input" found.`
`a = [1,2;3]`	`Error using ==> vertcat` `All rows in the bracketed expression must have the same number of columns.`

Die letzte Fehlermitteilung beruht darauf, dass das Feldelement `a(2,2)` fehlt, und ein Feld in allen Positionen besetzt sein muss. Will man markieren, dass an dieser Stelle der Matrix eigentlich kein "richtiger" Wert steht, kann man sich der Zahl `nan` (Not a Number) bedienen. Richtig müsste es also heissen:

```
a = [1, 2; 3, 4]; b = [1, 2; 3, nan];
```

2.5.4 Doppelpunkt - Colon

Die `Doppelpunktnotation` ist eine der mächtigsten Bestandteile von MATLAB. Sie kann einerseits zur Konstruktion von Vektoren (Kapitel 3), aber auch zum Zugriff auf Teile von Matrizen (Index, Kapitel 3) verwendet werden. Alle Details dazu findet man in Kapitel 3 . Hier werden nur elementare Beispiele angeführt:

```
m = [1:5]       % [1,2,3,4,5]
m = [1:2:5]     % [1,3,5]
m = [5:-1:1]    % [5,4,3,2,1]
x = [0:0.1:2]   % [0,0.1,0.2, ...., 1.9,2.0]
```

2.5.5 Hochkomma - Quotation Mark

Das Hochkomma wird zur Definition von Zeichenketten (Strings) verwendet:

```
str1 = 'Winfried'; str2 = 'Kernbichler';
str3 = 'Resultat: ';
str4 = num2str( sin(1) );
disp([str3,str4])
```

Details zu diesen Beispielen findet man im Abschnitt Kapitel 10 . Die hier verwendeten Befehle `num2str` und `disp` dienen zur Umwandlung von Zahlen in Zeichenketten und zur Darstellung von Ergebnissen im MATLAB-Kommandofenster.

Eine weitere Verwendung findet das Hochkomma als Operator für das Transponieren und das komplex konjugierte Transponieren von Matrizen. Wenn `M` eine Matrix ist, kann man den Operator wie folgt anwenden:

```
M1 = M.'       % transpose
M2 = M'        % conjugate complex transpose
```

Diese Anwendung ist ident mit den Befehlen `ctranspose`, bzw. mit `transpose`. Details zum Bearbeiten von Matrizen findet man im Abschnitt Kapitel 6 .

2.5.6 Prozent und Rufzeichen - Percent and Exclamation Point

Mit Hilfe des Prozentzeichens `%` können Kommentare in MATLAB-Programme eingefügt werden. Macht man das am Anfang des Files, z.B. wie

```
% Program: func
% Aufruf:  [a,b] = func(x,y)
% Beschreibung ......
% Input:   x: Beschreibung
%          y: ....
% Output:  a: ....
%          b: ....
% Autor:   Winfried Kernbichler
% Datum:   01-03-2004
```

kann man diese Kommentare als Programmdokumentation verwenden, die mit dem Befehl `help` einfach betrachtet werden kann.

In weiterer Folge kann man dann Programmzeilen kommentieren,

```
% Abstandsberechnung
d = sqrt(x.^2 + y.^2);
[ds,ind] = sort(d); % Sortierung nach Groesse
```

wobei dies in eigenen Zeilen oder am Ende von Zeilen gemacht werden kann.

Mit Hilfe des Rufzeichens können Systembefehle an das Betriebssystem übergeben werden, die dann ausserhalb von MATLAB abgearbeitet werden:

```
!cp file1 file2      % Kopieren
!mv file1 file2      % Verschieben
```

Das Rufzeichen ist eine Kurzform des MATLAB-Befehls `system`, der auch die Rückgabe der Ergebnisse auf Variable ermöglicht.

Einige wichtige Systembefehle sind aber auch direkt in MATLAB vorhanden:

BEFEHL	BEDEUTUNG
`dir`	Verzeichnis Listing.
`pwd`	Anzeige des aktuellen Verzeichnisses.
`cd`	Wechsel des Verzeichnisses.
`mkdir`	Anlegen von Verzeichnissen.
`rmdir`	Löschen von Verzeichnissen.
`delete`	Löschen von Files.
`copyfile`	Kopieren von Files.
`movefile`	Verschieben von Files.
`type`	Ausgabe von Files am Schirm.
`fileattrib`	Setzen von Fileattributen (Rechte).

2.5.7 Operatoren

Eine Reihe von Zeichen sind für Operatoren reserviert, die hier nur kurz angeführt werden sollen. Details findet man in den jeweiligen Verweisen:

TYP	VERWEIS	ZEICHEN
Arithmetisch - Matrizen	Kapitel 6	`+ - * / \ ^`
Arithmetisch - Elemente	Kapitel 4	`+ - .* ./ .\ .^`
Transponieren	Kapitel 6	`' .'`
Vergleich	Kapitel 4	`== ~= < <= > >=`
Logisch	Kapitel 4	`~ & |`

2.6 Schlüsselwörter - Keywords

In MATLAB sind eine Reihe von Schlüsselwörtern definiert, die im Wesentlichen zu Steuerkonstrukten Kapitel 7 gehören. In alphabetischer Reihenfolge sind dies:

`break`	`case`	`catch`	`continue`	`else`
`elseif`	`end`	`for`	`function`	`global`
`if`	`otherwise`	`persistent`	`return`	`switch`
`try`	`while`			

2.7 MATLAB-Skripts und MATLAB-Funktionen

MATLAB kennt zwei Typen von Programmeinheiten, Skripts und Funktionen, die im Detail im Abschnitt Kapitel 9 besprochen werden.

MATLAB-Skripts sind Programme, die im MATLAB-Arbeitsbereich (Workspace) ablaufen und keine Übergabeparameter kennen. Ihnen sind alle definierten Variablen im Workspace (`who`) bekannt, zwei unterschiedliche Skripts können also wechselseitig Variablen benutzen oder überschreiben. Das kann einerseits praktisch sein, birgt aber andererseits auch große Gefahren unerwünschter Beeinflussung. Will man sicher sein, dass Skripts in einem "reinen" Workspace ablaufen, muss man den Befehl `clear all` verwenden.

MATLAB-Funktionen Kapitel 9 hingegen werden in einem eigenen Arbeitsbereich abgearbeitet. Hier gibt es also keine unerwünschten Querverbindungen. Ihre Kommunikation mit Skripts (oder anderen Funktionen) erfolgt durch sogenannte Übergabeparameter,

```
function [out1,out2,out3] = test(in1,in2,in3)
```

wobei die Position innerhalb der Klammern die Zuordnung bestimmt. Ein Aufruf der Funktion `test` in folgender Art,

```
[a,b,c] = test(x,y,z)
```

führt innerhalb der Funktion dazu, dass `in1` den Wert von `x`, `in2` den Wert von `y` und `in3` den Wert von `z` zugewiesen bekommt.

Nach Ablauf aller Programmschritte innerhalb der Funktion `test`, wobei die Werte für `out1`, `out2` und `out3` berechnet werden müssen, bekommt ausserhalb der Funktion die Variablen `a` den Wert von `out1`, `b` den Wert von `out2` und `c` den Wert von `out3`.

Der lokale Arbeitsbereich einer Funktion ist bei jedem Aufruf leer. Nach dem Aufruf sind also nur die Input-Parameter bekannt. In Funktionen können natürlich genauso wie in Skripts alle MATLAB-Befehle und eigene Programme verwendet werden.

Zwei Regeln müssen bei Funktionen unbedingt eingehalten werden. Erstens, die Funktion muss in einem gleichnamigen MATLAB-File abgespeichert werden, d.h., die Funktion `test` muss im File `test.m` gespeichert werden und steht dann unter dem Namen `test` zur Verfügung. Dabei soll man keinesfalls existierende MATLAB Funktionsnamen (`exist`) verwenden, da sonst deren Zugänglichkeit blockiert ist. Zweitens, muss die Funktion eine sogenannte Deklarationszeile enthalten, die den Funktionsnamen und die Namen (und somit die Anzahl) der Übergabeparameter enthält. Diese Zeile muss mit `function` beginnen (siehe oben).

2.7.1 Einfache Beispiele

Zur Einführung werden hier zwei einfache Beispiele gezeigt, wobei man weiterführende Beispiele im Abschnitt Kapitel 9 findet.

Eine einfache Funktion mit zwei Input- und zwei Output-Parametern könnte so aussehen:

```
function [a,b] = test_fun1(x,y)
% TEST_FUN1 - Test Function
%   Syntax:   [a,b] = test_fun1(x,y)
%   Input:    x,y - Array (same size)
%   Output:   a,b - Array (same size as x and y)
%             a = sqrt(x.^2 + y.^2);
%             b = exp(-a.^2)
a = x.^2 + y.^2;  % ist hier eigentlich a.^2
                  % Variable darf aber nicht a^2 heissen
b = exp(-a);      % dafuer kann man jetzt einfacha verwenden
a = sqrt(a);      % und zum Schluss die Wurzel ziehen
```

Sie besteht aus einer Deklarationszeile, einer Reihe von Kommentarzeilen, die mit dem Befehl `help` angezeigt werden können, und drei Programmzeilen zur Berechnung der Output-Parameter. Beachten Sie bitte die Verwendung des Operators `.^`, da es sich bei den Übergabegrößen um Felder handeln kann (elementweise Berechnung).

Das entsprechende Skript zum Aufruf der Funktion könnte so aussehen:

```
% Test-Skript for test_fun1.m
% Winfried Kernbichler 08.03.2004
z_max = 1; z_n   = 101;  % Prepare input
z_1 = linspace(-z_max, z_max, z_n);
```

```
[d, e] = test_fun1(z_1, -z_1);  % Call function
figure(1); % Plot output
plot(z_1,d,'r',z_1,e,'b:');
xlabel('z_1'); ylabel('f(z_1,-z_1)');
legend('Distance','Exponent');
title('Result of test_fun1');
```

Im aufrufenden Skript werden typischerweise die Input-Parameter vorbereitet und die Ergebnisse dargestellt. Man trennt damit das "Umfeld" von der eigentlichen Berechnung.

Will man mit dem Benutzer des Programmes kommunizieren, kann man zur Eingabe von `z_max` und `z_n` auch den Befehl `input` eventuell in folgender Form verwenden:

```
z_max = input('Bitte geben Sie z_max ein: ');
```

Manchmal möchte man eine Funktion auch für die Erledigung unterschiedlicher Aufgaben verwenden. Dazu bietet sich die Verwendung der Steuerstruktur `switch` an. Bei dieser Steuerstruktur wird eine Schaltvariable `switch` benutzt. Für verschiedene Werte dieser Schaltvariablen können dann Fälle (`case`) und entsprechende Aktionen programmiert werden.

```
function [a,b] = test_fun2(typ,x,y)
% TEST_FUN2 - Test Function
%   Syntax:   [a,b] = test_fun2(typ,x,y)
%   Input:    typ - String
%             x,y - Array (same size)
%   Output:   a,b - Array (same size as x and y)
%             a = sqrt(x.^2 + y.^2);
%             b = exp(-a.^2)  [typ: 'exp']
%             b = sech(-a.^2) [typ: 'sech']
a = x.^2 + y.^2;
switch typ
 case 'exp'
  b = exp(-a);
 case 'sech'
  b = sech(-a);
 otherwise
  error('Case not defined!')
end
a = sqrt(a);
```

Je nach Wert der der Variablen `typ` kann die Funktion nun zwei verschiedene Aufgaben erledigen. Eine genaue Beschreibung von Steuerstrukturen finden Sie im Abschnitt Kapitel 7 , Details zum Befehl `switch` finden Sie im Abschnitt Kapitel 7 .

Der Aufruf schaut nun ein wenig anders aus (`typ`):

```
% Test-Skript for test_fun2.m
% Winfried Kernbichler 08.03.2004
z_max = 1; z_n   = 101;  % Prepare input
typ   = 'exp';  % oder 'sech'
z_1 = linspace(-z_max, z_max, z_n);
[d, e] = test_fun2(typ,z_1, -z_1);  % Call function

figure(1); % Plot output
plot(z_1,d,'r',z_1,e,'b:');
xlabel('z_1'); ylabel('f(z_1,-z_1)');
legend('Distance',typ);
title('Result of test_fun2');
```

Anmerkung: Will man mit der Funktion `input` den Wert der Variablen `typ` abfragen, empfiehlt es sich, sie in dieser Form

```
typ = input('Bitte geben Sie den Typ ein: ','s')
```

zu verwenden. Damit kann man einfach `exp` anstelle von `'exp'` eingeben und MATLAB erkennt trotzdem, dass es sich um einen String handelt.

Anmerkung: Will man die Eingabe des Typs noch weiter erleichtern (Groß- oder Kleinschreibung, nur Anfangsbuchstabe(n), kann man die `switch`-Konstruktion verbessern. Man kann z.B. alle Zeichenketten mit dem Befehl `lower` in Kleinbuchstaben verwandeln. Details zur Verwendung von Zeichenketten findet man im Abschnitt Kapitel 10 . Die Konstruktion

```
switch lower(typ)
 case 'exp'
  ...
end
```

würde nun auch mit Werten wie `Exp` oder `EXP` funktionieren. Will man nur den Anfangsbuchstaben der Zeichenkette auswerten, kann man mit Hilfe der Indizierung auf den ersten Buchstaben zugreifen. Dies könnte so aussehen:

```
switch lower(typ(1))
 case 'e'
  ...
end
```

2.8 Einfache MATLAB-Skripts

Zur Vorbereitung auf die Übung gibt es hier noch zwei einfache Beispiele, wobei sich das eine eher mit Skalaren und das andere eher mit Vektoren beschäftigt.