

## 2 Script-Dateien und Funktionen

### 2.1 Script-Dateien

#### 2.1.1 Grundsätzliches

Das Arbeiten auf der Ebene der Kommandozeile, wie wir es bisher gepflegt haben, war bequem, weil wir ohne größere Überlegungen sofort mit der Arbeit beginnen konnten. Sicher haben Sie inzwischen auch schon bemerkt, dass das erneute Eintippen jedes Kommandos durchaus lästig werden kann, wenn eine Folge von Befehlen mehrfach genutzt werden soll. Es wäre sinnvoll, solche Kommandofolgen in einer Datei zu speichern und anschließend stets nur noch die Datei als Ganzes aufzurufen. Wir erwarten, dass dann die jeweilige Kommandofolge automatisch ausgeführt wird. Dies ist in allen Programmiersprachen wie Visual Basic, Fortran oder C üblich, und ähnliche Möglichkeiten werden auch von MATLAB, Octave oder Scilab zur Verfügung gestellt.<sup>1</sup>

Script-Dateien sind Textdateien. Sie enthalten Folgen von Befehlen, die als ausführbare Programme innerhalb des Hauptprogramms oder von der Kommandozeile aus mit einem einzigen Kommando aufgerufen und verwendet werden können.

Script-Dateien werden in MATLAB oder Octave mit der Dateierdung `.m` und in Scilab mit der Endung `.sce` oder `.sci` versehen, also zum Beispiel `test.m` beziehungsweise `nullstelle.sci`. MATLAB-Nutzer sprechen deshalb auch von M-Files.

Wir müssen *Script-Dateien* von *Funktionsdateien* unterscheiden. Bei Letzteren bleiben die Variablen „gekapselt“, sind also im aufrufenden Programm beziehungsweise auf der Kommandoebene nicht verfügbar. Auf sie kommen wir noch zurück.

Script-Dateien speichern hingegen ihre Variablen im Workspace von MATLAB, Octave oder Scilab ab. Die in der Script-Datei benutzten Variablen können

---

<sup>1</sup> Der Unterschied besteht darin, dass bei Scilab und Octave die Dateien von einem Interpreter Befehl für Befehl abgearbeitet werden. Bei Visual Basic, Fortran und C hingegen werden die Dateien zunächst von einem Compiler komplett in Maschinencode übersetzt und erst danach ausgeführt. Dadurch wird die Bearbeitung erheblich beschleunigt. MATLAB nimmt eine Zwischenstellung ein. Hier durchläuft der Programmcode einen sogenannten Parser, der den Code zumindest teilweise kompiliert und dabei optimiert.

also nicht nur innerhalb der Script-Datei selbst, sondern auch anschließend von der Kommandozeile aus aufgerufen werden. Script-Dateien in MATLAB und Octave *müssen* die Endung `.m` aufweisen. Script-Dateien in Scilab *sollten*, müssen aber nicht die Endung `.sce` besitzen. Der Aufruf einer Script-Datei erfolgt in MATLAB einfach durch Angabe des Dateinamens (ohne die Endung `.m`), in Scilab durch das Kommando `exec file_name.sce`<sup>2</sup>.

Die Scilab-Syntax zum Aufrufen einer Funktion sieht vor, dass neben dem Funktionsnamen auch der Suchpfad mit angegeben wird, und erlaubt darüber hinaus verschiedene Optionen. Die Funktion `dechex.sce` (die wir später noch benötigen) kann zum Beispiel aus dem Unterverzeichnis `test` wie folgt aufgerufen werden:

```
-->exec('test\dechex.sce');
```

Mit dem abschließenden Semikolon wird die Ausgabe der Zeilen nach dem Aufruf unterdrückt, so wie bei MATLAB. Andernfalls würden bei der Ausführung alle Zeilen dieser Datei angezeigt, einschließlich der Kommentare.

Um eine Script-Datei unter MATLAB zu schreiben, benutzen Sie am besten den zugehörigen Editor. Sie rufen ihn zum Beispiel mit dem Kommando `edit` auf oder mit dem Button `NewScript` unter dem Reiter `Home`. Im Editor werden die Befehle Zeile für Zeile in gleicher Weise eingetragen wie vorher auf der Kommandooberfläche. Anschließend wird das M-File gespeichert. Wenn die Datei dann ihre Arbeit verrichten soll, rufen Sie sie unter dem gleichen Namen, unter welchem sie gespeichert wurde, wieder auf. Dadurch werden die Befehle der Reihe nach automatisch ausgeführt. Alternativ können Sie die Ausführung auch aus dem Editorfenster heraus starten, indem Sie den grünen `Run`-Button oder die Taste `F5` drücken.

In Scilab kann ebenfalls auf einen Editor zugegriffen werden. Man findet ihn, von der Konsole ausgehend, unter `Datei > Eine Datei öffnen` oder gleichfalls mit dem Kommando `edit`. Im Prinzip kann jedoch auch jeder andere Texteditor verwendet werden.

An einem einfachen Beispiel wollen wir die Ausführung einer Script-Datei verdeutlichen.

---

<sup>2</sup> Der Dateiname kann beim Aufruf mit oder ohne Klammer sowie mit oder ohne Anführungszeichen stehen. Anstelle von Anführungszeichen kann der Name auch in Hochkommas eingeschlossen werden. Hier ist Scilab sehr tolerant.

Codetabelle 2.1

MATLAB, Octave	Scilab
<p>Die Script-Datei <code>testscript.m</code> führt folgende Operationen aus:<sup>3</sup></p> <pre>% testscript.m % Addition zweier Zahlen a und b a = 1; b = 2; a + b</pre>	<p>Die Script-Datei <code>testscript.sce</code> führt folgende Operationen aus:</p> <pre>// testscript.sce // Addition zweier Zahlen a und b a = 1; b = 2; a + b</pre>
<p>Sie wird ausgeführt nach Eingabe des Befehls <code>testscript</code>. In der Kommandozeile erscheint dann:</p> <pre>&gt;&gt; testscript ans =     3</pre> <p>Ein Semikolon nach dem Ausführungsbefehl bringt in MATLAB keine Veränderung:</p> <pre>&gt;&gt; testscript; ans =     3</pre>	<p>Sie wird ausgeführt nach Eingabe des Befehls <code>exec testscript.sce</code>. In der Kommandozeile erscheint dann:</p> <pre>--&gt;exec testscript.sce --&gt;// testscript.sce --&gt;// Addition zweier Zahlen a und b --&gt;a=1; --&gt;b=2; --&gt;a+b ans =     3.</pre> <p>Wird der Befehl <code>exec testscript.sce</code> mit Semikolon abgeschlossen, so wird nur das Ergebnis angezeigt:</p> <pre>--&gt;exec testscript; ans =     3.</pre>

Die Befehlsfolge einer Script-Datei stellt ein (einfaches) Programm im Sinne der Informatik dar. Es sollte deshalb (wenn es sich nicht um solch ein kurzes Testprogramm wie das soeben beschriebene handelt) üblicherweise mindestens drei Teile enthalten:

- Bereiche einer Script-Datei:
- Teil zum Vorgeben bzw. Einlesen von Werten (aus einer Datei oder von der Tastatur aus),
  - Rechenteil,
  - Ausgabeteil (Werte an Kommandoebene in eine Grafik oder in eine Datei).

Wir sollten auch nicht vergessen, unser Programm besser lesbar zu machen, indem wir sinnvolle Kommentare hinzufügen. So mancher Anwender hat sein kommentarloses Programm bereits nach wenigen Wochen nicht mehr verstanden.

<sup>3</sup> Die Inhalte von Script-Dateien kennzeichnen wir durch die Schriftart Arial Narrow.

Wie wir uns erinnern, werden Kommentarzeilen in MATLAB oder Octave mit dem Prozentzeichen, in Scilab mit doppeltem Schrägstrich eingeleitet. Octave akzeptiert auch ein Balkenkreuz anstelle des Prozentzeichens.

MATLAB-Kommentare, die unmittelbar am Anfang eines M-Files stehen, werden übrigens als Hilfe-Text ausgegeben.

■ Codetabelle 2.2

MATLAB, Octave	Scilab
<pre>&gt;&gt; help testscript testscript.m Addition zweier Zahlen a und b</pre>	Bei Scilab gibt es keine so einfache Möglichkeit, eine eigene Hilfe zu generieren.

Umfangreichere Aufgaben sollten, wie überall in der Informatik, in Teil- oder Unterprogramme aufgeteilt werden. Bei den kurzen Befehlsfolgen in unseren Beispielen ist das allerdings nicht nötig.

Mit dem Kommando `pause` wird die Ausführung einer Script-Datei unterbrochen. Dieser Befehl kann ausgesprochen nützlich für den Test eines Programms sein. Unter Scilab tritt an dessen Stelle der Befehl `halt`. Die Bearbeitung des Kommandos `pause` ist hier komplizierter. Darauf werden wir erst später zurückkommen.

### 2.1.2 Einrichten des Arbeitsverzeichnisses

Der einfachste Weg, unter Scilab eine Script-Datei arbeiten zu lassen, besteht darin, sie mittels Doppelklick auf das Dateisymbol aufzurufen. Dabei öffnen sich automatisch der Scilab-Editor und die Scilab-Konsole. Das aktuelle Arbeitsverzeichnis von Scilab liegt normalerweise in dem Verzeichnis, in dem Scilab gestartet wurde.

Ist dagegen Scilab bereits geöffnet, so müssen wir das Arbeitsverzeichnis unter Umständen erst suchen. Wir können es finden, indem wir im Menü den Befehl

`Datei > Aktuelles Verzeichnis anzeigen`

wählen. Zum Ändern wählen wir in der darüberliegenden Zeile:

`Datei > Aktuelles Verzeichnis ändern`

Außerdem besteht die Möglichkeit, zu diesem Verzeichnis per Kommandozeilenbefehl zu wechseln. Dazu ist das Kommando `cd` („Change directory“) geeignet:

```
-->cd D:\Daten\Numerik\MATLAB_Scilab\Script-Dateien
```

ans

```
cd D:\Daten\Numerik\MATLAB_Scilab\Script-Dateien
```

Anders als bei den sonstigen Windows-Programmen dürfen die Datei- und Verzeichnisnamen keine Leerzeichen enthalten.

Wie Sie wahrscheinlich wissen, lässt sich jedoch das Arbeitsverzeichnis eines Windows-Programms immer auch einstellen, indem wir mit der rechten Maustaste das (MATLAB- oder Scilab-)Icon anklicken und Eigenschaften aufrufen. Unter Ausführen in können Sie dann den gewünschten Verzeichnisnamen eingeben.

Bei MATLAB lässt sich das Arbeitsverzeichnis durch Mausklick wie in Abb. 2.1 gezeigt einstellen. In älteren MATLAB-Versionen gab es auch die Möglichkeit, wie bei Scilab im Menü File > Run Script... > *Dateiname* auszuwählen.

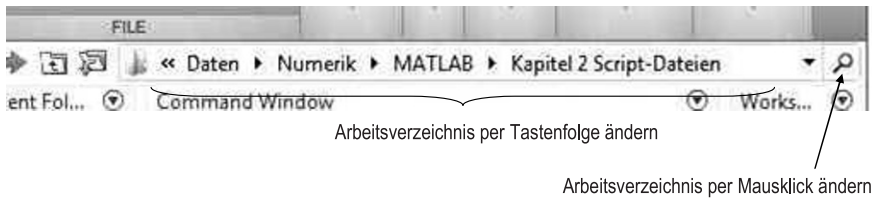


Abb. 2.1 Einrichtung des Arbeitsverzeichnisses unter MATLAB

Der aktuelle Suchpfad kann unter MATLAB oder Octave mit `path` angezeigt werden. In der Regel sind dann sehr viele Verzeichnisse zu sehen. Ein neuer Pfad wird in MATLAB über den Menübefehl File > Set Path angelegt. Alternativ kann man auch das Kommando `addpath` verwenden, zum Beispiel wie folgt:

```
addpath('C:\Programme\MATLAB\work\')
```

Wenn Octave ohne grafische Benutzeroberfläche verwendet wird, müssen alle Eingaben in der Kommandozeile ausgeführt werden. Dann ist man auf das Kommando `addpath` angewiesen,

```
addpath("C:/Programme/Octave/share/octave/3.0.1/m/phys")
```

*Datei auf Webseite*

Wie unter MATLAB oder Octave der Pfad zu allen Verzeichnissen geöffnet werden kann, die Sie von der Verlags-Webseite heruntergeladen haben, wurde bereits in der Einleitung erwähnt. Zu Beginn jeder Sitzung sollte dazu das Kommando `userpath_set` eingegeben werden, das wir auf der Verlags-Webseite anbieten. Damit wird ein Pfad zu Dateien in allen Unterverzeichnissen geöffnet. Leider funktioniert dies bei Scilab nicht in ähnlich einfacher Weise – hier führt kein Weg daran vorbei, das aktuelle Arbeitsverzeichnis stets neu aufzurufen.

### 2.1.3 Ein- und Ausgabekommandos

Sehr häufig ist es nötig, in eine Script-Datei während ihrer Bearbeitung Daten von der Kommandozeile aus einzugeben. Dies soll dem Benutzer durch ein geeignetes Kommando angezeigt werden. Hierfür steht in MATLAB, Octave und Scilab der Befehl `input` zur Verfügung. Damit werden die benötigten Werte eingelesen. Für Ausgaben kann `disp` verwendet werden.

Die formatierte Ausgabe erfolgt über `sprintf` bei MATLAB beziehungsweise `msprintf` bei Scilab. Dieses Kommando dient eigentlich dazu, eine *Zeichenkette* (einen *String*) aus den bereitgestellten Daten zu erzeugen. Eine solche Zeichenkette kann mittels `disp` im Kommandofenster angezeigt werden. Hier wollen wir uns jedoch über diese Anwendung hinaus noch nicht weiter mit Strings beschäftigen. Die verschiedenen Möglichkeiten der Formatierung sind anhand der Dateien hinreichend zu erkennen und werden anschließend noch einmal kurz zusammengefasst. Sie erinnern an äquivalente Ausgabemöglichkeiten in der Programmiersprache C. Auch neben der Verwendung in `disp` sind Strings auch sonst sehr vielseitig verwendbar.

Zur Illustration geben wir jetzt gleich mehrere Beispiele für Ausgabeformate an. Als Beispiel soll ein Demo-Programm zur Umrechnung von Radiant in Grad dienen. Der Wert des Winkels wird in diesen Script-Dateien in Radiant eingegeben, die Ausgabe erfolgt auf unterschiedliche Weise. Da die Kommandozeilen ziemlich lang sind, werden die MATLAB- und Scilab-Operationen hier nacheinander dargestellt.

■ Codetabelle 2.3

MATLAB, Octave	
Die Script-Datei <code>rad_grad.m</code> führt folgende Operationen aus	<a href="#">Datei auf Webseite</a>
<pre>% rad_grad.m % M-File zur Umwandlung von Radiant in Winkelgrad % keine Angabe von Winkelminuten und Winkelsekunden, sondern von Zehntel- % und Hundertstelgrad  % Werte eingeben % ***** disp('Umrechnung eines Winkels aus Radiant in Grad') alpha_rad = input('Winkel in Radiant eingeben: ');  % Rechnung % ***** alpha_grad = alpha_rad * 360/2/pi;  % Ausgabe von Text (formatierte Ausgabe) % ***** disp(' ') % erzeugt Leerzeile disp('Festkommaformat, 4 Stellen insgesamt, 3 Nachkommastellen');     string1 = sprintf('Winkel in Grad = %4.3f°\n', alpha_grad);     disp(string1); disp('Festkommaformat, 4 Stellen insgesamt, keine Nachkommastellen');     string2 = sprintf('Winkel in Grad = %4.0f°\n', alpha_grad);</pre>	

```

disp(string2);
disp('Dezimalformat:');
    string3 = sprintf('Winkel in Grad = %4.1d°\n', alpha_grad);
    disp(string3);
disp('Exponentialformat:');
    string4 = sprintf('Winkel in Grad = %4.3e°\n', alpha_grad);
    disp(string4);

```

Die Ausgabe zeigt dann Folgendes:

```

>> rad_grad
Umrechnung eines Winkels aus Radiant in Grad
Winkel in Radiant eingeben: 1.05

Festkommaformat, 4 Stellen insgesamt, 3 Nachkommastellen
Winkel in Grad = 60.161°

Festkommaformat, 4 Stellen insgesamt, keine Nachkommastellen
Winkel in Grad = 60°

Dezimalformat:
Winkel in Grad = 6.0e+001°

Exponentialformat:
Winkel in Grad = 6.016e+001°

```

### Scilab

Die Script-Datei `rad_grad.sce` führt folgende Operationen aus:

[Datei auf Webseite](#)

```

// rad_grad.sce
// Scilab-Script-File zur Umwandlung von Radiant in Winkelgrad
// keine Angabe von Winkelminuten und Winkelsekunden, sondern von Zehntel-
// und Hundertstelgrad

// Werte eingeben
// *****
disp('Umrechnung eines Winkels aus Radiant in Grad')
alpha_rad = input('Winkel in Radiant eingeben: ');

// Rechnung
// *****
alpha_grad = alpha_rad * 360/2/%pi;

// Ausgabe von Text (formatierte Ausgabe)
// *****
//disp(' ') // erzeugt Leerzeile
disp('Festkommaformat, 4 Stellen insgesamt, 3 Nachkommastellen');
    string1 = msprintf('Winkel in Grad = %4.3f°\n', alpha_grad);
    disp(string1);
disp('Festkommaformat, 4 Stellen insgesamt, keine Nachkommastellen');
    string2 = msprintf('Winkel in Grad = %4.0f°\n', alpha_grad);
    disp(string2);
disp('Dezimalformat:');
    string3 = msprintf('Winkel in Grad = %4.1d°\n', alpha_grad);
    disp(string3);
disp('Exponentialformat:');
    string4 = msprintf('Winkel in Grad = %4.3e°\n', alpha_grad);
    disp(string4);

```

Die Ausgabe zeigt dann Folgendes:

-->exec rad\_grad.sce;

Umrechnung eines Winkels aus Radiant in Grad

Winkel in Radiant eingeben: 1.05

Festkommaformat, 4 Stellen insgesamt, 3 Nachkommastellen

Winkel in Grad = 60.161°

Festkommaformat, 4 Stellen insgesamt, keine Nachkommastellen

Winkel in Grad = 60°

Dezimalformat:

Winkel in Grad = 60°

Exponentialformat:

Winkel in Grad = 6.016e+001°

Natürlich könnten wir auch unformatierte Daten ausgeben, indem wir wie bisher einfach das Semikolon hinter der Variablen fortlassen. Für Testzwecke ist dies auch vollkommen ausreichend. Die formatierte Ausgabe sieht allerdings eleganter aus.

Programmierpraxis mit Standardfunktionen	
MATLAB, Octave	Scilab
<ul style="list-style-type: none"><li>• Zeichenketten (Strings) werden in Hochkommas eingegeben, z.B. <code>s1 = 'Hallo'</code></li><li>• Mittels <code>disp</code> können Variablen oder Strings angezeigt werden, z.B. <code>disp(s1)</code></li><li>• Mit <code>sprintf</code> werden formatierte Daten einer Variablen <code>a</code> als String ausgegeben. In der Form <code>str = sprintf('Text %m,nf', a)</code> <code>f</code> bedeutet „Festkommaformat“: <code>m</code> ist die Gesamtzahl der ausgegebenen Stellen, <code>n</code> die Zahl der Nachkommastellen. Für andere Formate und weitere Möglichkeiten verweisen wir auf die Hilfe.</li></ul>	<ul style="list-style-type: none"><li>• Zeichenketten (Strings) werden in Hochkommas eingegeben, z.B. <code>s1 = 'Hallo'</code></li><li>• Mittels <code>disp</code> können Variablen oder Strings angezeigt werden, z.B. <code>disp(s1)</code></li><li>• Mit <code>msprintf</code> werden formatierte Daten einer Variablen <code>a</code> als String ausgegeben. In der Form <code>str = msprintf('Text %m,nf', a)</code> <code>f</code> bedeutet „Festkommaformat“, <code>m</code> ist die Gesamtzahl der ausgegebenen Stellen, <code>n</code> die Zahl der Nachkommastellen. Für andere Formate und weitere Möglichkeiten verweisen wir auf die Hilfe.</li></ul>

**Aufgabe 1** Funktionsverlauf eines Polynoms

a) Schreiben Sie ein Script-Programm, das den Funktionsverlauf eines beliebigen Polynoms in den Grenzen  $-x \leq x \leq 5$  grafisch wiedergibt.



Das Polynom soll wie in Abschnitt 1.9.1 durch seine Koeffizienten als Vektor in eckigen Klammern dargestellt werden. Lassen Sie die Koeffizienten vom Benutzer durch den Befehl `input` eingeben. Versehen Sie die Grafik mit Achsenbezeichnungen und einem geeigneten Titel.

b) Neben der Grafik sollen auch die Nullstellen des Polynoms über `roots` ausgegeben werden. ■

## Aufgabe 2 Logarithmische Darstellung einer Funktion

Im Zusammenhang mit Schwingungen und in der Regelungstechnik werden Funktionen des Typs

$$y = 1 / \sqrt{(1 - \omega^2)^2 + (2\delta\omega)^2}$$

untersucht. Dabei ist  $\omega$  die sogenannte Anregungsfrequenz und  $\delta$  die Dämpfung. Wir interessieren uns hier nicht für die physikalisch-technischen Zusammenhänge, wollen diese Funktion aber in einem doppelt-logarithmischen Maßstab grafisch darstellen.

Schreiben Sie also ein Script, welches die Funktion in den Grenzen  $10^{-1} \leq \omega \leq 10$  doppelt-logarithmisch dargestellt. Die Größe  $\delta$  soll über ein `input`-Kommando eingelesen werden.

*Zusatz:* Oft ist es nützlich, eine sogenannte Default-Eingabe vorzusehen. Falls nach dem `input`-Kommando vom Benutzer kein Zahlenwert eingegeben wird, kann man einen Vorgabewert vorsehen. Dafür lässt sich die folgende Konstruktion nutzen:

`if isempty(delta), delta = .1, end.` Vergleichen Sie hierzu auch die Syntaxbeschreibung am Ende von Abschnitt 4.2.3. ■

## 2.2 Funktionen in MATLAB, Octave und Scilab

### 2.2.1 Allgemeines über Funktionen

Neben den Script-Files gibt es auch Dateien, die Funktionen beinhalten.

Funktionen stellen im Gegensatz zu Script-Dateien Programme dar, die ihre Werte nur über festgelegte Ein- und Ausgabeschnittstellen mit dem aufrufenden Programmteil oder der Kommandoebene austauschen.

Grundsätzlich bestehen auch Funktionen aus den Teilen *Eingabe – Rechenteil – Rückgabe*. Im Unterschied zu Script-Dateien sind jedoch bei einer Funktion von außen die internen Variablen und Kommandos nicht sichtbar. Eine Funktion ist demnach eine Black Box, die nur über die Eingabe- und die Ausgabe-Schnittstellen Verbindung zum aufrufenden Programmteil aufnehmen kann (Abb. 2.2).

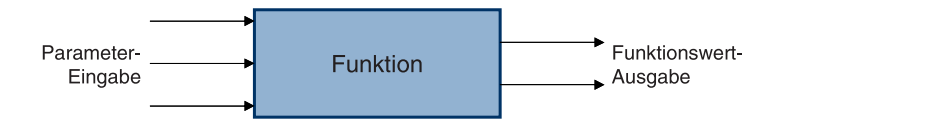


Abb. 2.2 Funktion als Black Box

Wir können zum Beispiel auf der Kommandoebene eine Variable `a` benutzen und innerhalb einer Funktion eine Variable gleichen Namens. Beide nehmen jedoch voneinander keinerlei Notiz, ihre Werte können ganz verschieden sein.

Variablen, die innerhalb der Funktion verwendet werden, sogenannte *interne (gekapselte) Variablen*, sind von außen (also auf der Kommandoebene oder im aufrufenden Programm) nicht verfügbar.

Zahlreiche vordefinierte Funktionen sind, wie bereits früher erwähnt, als Teile der Programmpakete von MATLAB, Octave oder Scilab enthalten (mitgelieferte Funktionen). Deren Namen stellen *Schlüsselwörter* dar, die nicht anderweitig verwendet werden sollten. Einige dieser Funktionen, die besonders häufig benötigt werden und dementsprechend optimierte Zugriffszeiten ermöglichen sollen, sind die sogenannten *Built-in-Funktionen* („eingebaute Funktionen“), in Scilab heißen sie *hart-codierte Funktionen*. Dazu gehören zum Beispiel `sin`, `sind`, `tan`, `tand`, `sqrt` usw. Ihre innere Codierung ist nicht sichtbar. Bei anderen mitgelieferten Funktionen kann der interne Code als Textdatei durchaus aufgerufen werden. Diese werden bei Scilab auch *Makros* genannt, ihre Codierung kann man mit dem Editor aufrufen. Hierzu gehört zum Beispiel die Funktion `sinh` (Hyperbelsinus). Darüber hinaus ist es dem Anwender möglich, eigene *anwenderdefinierte Funktionen* zu schreiben und zu verwenden.

Zuweilen wird unabsichtlich eines dieser Schlüsselwörter überschrieben, indem vielleicht eine Variable oder Datei gleichen Namens erzeugt wird. MATLAB weist aber im Gegensatz zu Scilab darauf leider nicht hin. Dadurch können Fehler entstehen, die nur recht schwer erkannt werden. Was passiert, wenn man (versehentlich) einmal die Sinusfunktion zu überschreiben versucht, wird mit der folgenden Kommandosequenz gezeigt.

■ Codetabelle 2.4

MATLAB, Octave	Scilab
<pre>&gt;&gt; sin = 3 sin =     3 &gt;&gt; sin(pi/4) ??? Subscript indices must either be real positive integers or logicals. &gt;&gt; clear sin</pre>	<pre>--&gt;sin = 3 Warnung: definiere Funktion neu: sin . Verwende funcprot(0), um diese Nachricht zu vermeiden  sin =     3.</pre>

Praktische Mathematik mit MATLAB, Scilab und Octave  
für Ingenieure und Naturwissenschaftler

Thuselt, F.; Gennrich, F.P.

2013, XI, 439 S. 140 Abb., 40 Abb. in Farbe., Softcover

ISBN: 978-3-642-25824-4