



Leseprobe

Ulrich Stein

Programmieren mit MATLAB

Programmiersprache, Grafische Benutzeroberflächen, Anwendungen

ISBN (Buch): 978-3-446-44299-3

ISBN (E-Book): 978-3-446-44391-4

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44299-3>

sowie im Buchhandel.



Vorwort zur 5. Auflage

Dieses Lehrbuch wendet sich an Studentinnen und Studenten der Ingenieurwissenschaften in den ersten Semestern und orientiert sich in der Stoffauswahl an der Vorlesung „Angewandte Informatik“ am Department „Maschinenbau und Produktion“ der Hochschule für Angewandte Wissenschaften, am Berliner Tor in Hamburg.

Unser Weg durch die Informatik beginnt mit den elementaren Prinzipien der Datenverarbeitung und vermittelt dann anhand der in MATLAB® integrierten Programmiersprache systematisch die Grundkenntnisse des Programmierens. Darauf aufbauend vertiefen Anwendungen das Wissen und führen in die weitergehenden numerischen Verfahren von MATLAB ein. Wo für spezielle Probleme das Basismodul von MATLAB nicht mehr ausreicht, wird für eine intensivere Nutzung auf die MATLAB-Toolboxen™ verwiesen – deren Installation zum Verständnis dieses Buches jedoch nicht notwendig ist.

Anwendungsbeispiele, zahlreiche Abbildungen und Übungsaufgaben zum Stoff fördern das Verständnis. Das Werk ist sowohl für Programmierneulinge als auch für Umsteiger von anderen Programmiersprachen wie C geeignet.

In diesem Buch wird in MATLAB programmiert. Im Vordergrund steht dabei aber nicht primär das Programm MATLAB, sondern die Vermittlung von Programmierkenntnissen, wie sie auch für andere Sprachen nützlich sind. Das bedeutet, dass nicht immer Wert darauf gelegt wurde, den Programm-Code in Bezug auf die Stärken von MATLAB zu optimieren. Auch ist dies kein Referenz-Handbuch für MATLAB. Die MATLAB-Funktionen werden oft nur so weit vorgestellt, wie es für die aktuelle Aufgabenstellung nötig ist. Für eine vollständige Definition der Funktionen sei auf die MATLAB-Hilfe verwiesen.

Warum wird das Programmieren bei uns nicht anhand von C vermittelt?

In der Industrie habe ich einige Jahre Programme in C beziehungsweise C++ geschrieben. Nahezu alles, was das Programmiererherz begehrt, ist mit C möglich. Aber diese Sprache enthält auch viele, böse Fallen. Um sich einigermaßen fehlerfrei in C ausdrücken zu können, muss man lange damit gearbeitet haben, besonders wenn man grafische Oberflächen und Grafiken erstellen will oder sich im Bereich COM/OLE tummeln möchte. Die letzten beiden Punkte konnte ich deshalb in meiner zweisemestrigen C-Vorlesung höchstens erwähnen. Und Hand aufs Herz – nur ein sehr kleiner Teil der Ingenieur-Absolventen hat heutzutage im Beruf noch mit C zu tun. MATLAB jedoch wird in vielen Bereichen eingesetzt, sowohl im Fortgeschrittenen-Studium, zum Beispiel in der Regelungstechnik oder in der Simulation von Motoren, als auch in Industrie und Forschung.

Vor einigen Jahren habe ich den Versuch gestartet, Programmieren anhand von MATLAB zu lehren, mit ähnlichen Lernzielen und Inhalten wie vorher mit C. Und es hat geklappt. Besser noch – mit MATLAB war vieles einfacher. Ich kam schneller voran und konnte mehr Inhalte bringen. Und es machte Spaß, sowohl den Studenten als auch mir.

Dieses Buch soll das Programmieren lehren, aber nicht zum Informatiker ausbilden. Deshalb fehlen Themen wie Automatentheorie, formale Sprachen, Petrinetze, Maschinensprache, Netzwerke und auch die beliebten Umwandlungen vom Dezimal- zum Dualsystem. So auf Du und Du mit Prozessor, Speicher oder serieller Schnittstelle sind Ingenieure heute meist nicht mehr. Leider mussten auch die verketteten Listen wegfallen – dazu fehlen in MATLAB die Zeiger. Informationen zu diesen Themen finden Sie in Büchern zur Informatik, wie im „Informatik Handbuch“ von P. Rechenberg und G. Pomberger.

Frau Dipl.-Ing. Erika Hotho machte mir vor sechs Jahren den Vorschlag, ein Buch über das Programmieren mit MATLAB zu schreiben. Vielen Dank für Ihr Engagement und Dank auch an Frau Franziska Jacob, M.A und Frau Mirja Werner, M.A., die mich inzwischen kompetent betreuen, und an Frau Dipl.-Ing. (FH) Franziska Kaufmann, die mir beim Layout zur Seite stand. Dank auch an alle Kollegen, die mich zu diesem Projekt ermutigten und mir hilfreiche Tipps gaben, wie Prof. Dr.-Ing. habil. Jürgen Dankert, Prof. Dr.-Ing. Bernd Kost, Prof. Dr.-Ing. Wolfgang Schulz, Prof. Dr. rer. nat. Bernd Baumann, Prof. Dr.-Ing. Thomas Frischgesell. Nicht zu vergessen die Mitarbeiter des Rechenzentrums Berliner Tor, die mir in Fragen der MATLAB-Installation zur Seite standen.

2007 erschien die erste Auflage dieses Buches. Danke für die vielen Zuschriften und für die nahezu einhellig positive Reaktion der Leser. Für die fünfte Auflage habe ich mehrere Bereiche vollständig überarbeitet, zum Beispiel die neuen Methoden der objektorientierten Programmierung eingebaut und für die Prozess-Kommunikation den Kontakt zum Internet ausgewählt. Auch wurden einige weitere numerische Verfahren aufgenommen. Damit das Buch dadurch nicht noch dicker und teurer wird, habe ich im Gegenzug längere Programm-Listings entfernt und auf meine Homepage gepackt.

Die im Buch beschriebenen und abgebildeten Abläufe beziehen sich auf die Bedienoberfläche der Version **MATLAB 2014**. Andere MATLAB-Versionen präsentieren sich dem Anwender zum Teil mit einer leicht abgewandelten Oberfläche. Lassen Sie sich deshalb nicht verwirren. Die vorgestellten Programme wurden jedoch mit verschiedenen Versionen von MATLAB 7.0.1 bis MATLAB 2014 getestet. Erweiterungen und die Lösungen der Aufgaben finden Sie auf meiner Homepage

www.Stein-Ulrich.de/Matlab/

Ich wünsche den Lesern, dass Ihnen das Programmieren neben der Lernarbeit auch Spaß macht und dass Ihnen möglichst viel vom hier präsentierten Stoff im wirklichen Leben bei Problemlösungen nützt. Und nicht verdrängen oder vergessen: Informatik kann auch Schaden anrichten. Deshalb sollte jeder, der programmiert, sich überlegen, ob er sein Tun verantworten kann und will.

Hamburg, im Januar 2015

Ulrich Stein



Inhalt

1	Einführung	14
1.1	Hello, world	14
1.2	Datenverarbeitung	16
1.2.1	Hardware	16
1.2.2	Software	17
1.2.3	Datentypen	19
1.2.4	Editieren	20
1.2.5	Programmausführung	20
1.3	Erster Kontakt mit MATLAB	21
1.3.1	Der MATLAB-Desktop.....	21
1.3.2	MATLAB als Taschenrechner.....	22
1.3.3	Zahlen- und Textdarstellung.....	24
1.3.4	Variablen und Datentypen	26
1.3.5	Vektoren und Matrizen.....	29
1.3.6	MATLAB aufräumen	31
1.3.7	Zusammenfassung	31
1.3.8	Aufgaben	32
2	Programmstrukturen	34
2.1	Funktionen	34
2.1.1	Eine Black Box.....	34
2.1.2	Eingangs- und Rückgabeparameter	35
2.1.3	Funktionen in MATLAB	36
2.1.4	Funktionsbeispiel: Umfang.....	38
2.1.5	Stack, Funktionsparameter	40
2.1.6	Ablaufprotokoll	41
2.1.7	MATLAB-Arbeitsverzeichnis	42
2.1.8	Zusammenfassung	44
2.1.9	Aufgaben	45
2.2	Ein- und Ausgabe.....	46
2.2.1	I/O-Kanäle	46
2.2.2	Einfache Ausgabe.....	46
2.2.3	Formatierte Ausgabe	47
2.2.4	Einfache Eingabe.....	49
2.2.5	Ein-/Ausgabe-Beispiel: UmfangInput	51
2.2.6	Zusammenfassung	52

2.2.7	Aufgaben	52
2.3	Ablaufstrukturen	53
2.4	Verzweigungen	54
2.4.1	Bedingungen	54
2.4.2	Vergleiche	54
2.4.3	Logische Verknüpfungen	55
2.4.4	Alternative	56
2.4.5	if-else-Beispiele	59
2.4.6	Fallunterscheidung	61
2.4.7	Zusammenfassung	63
2.4.8	Aufgaben	63
2.5	Schleifen	64
2.5.1	Schleifenbedingung	64
2.5.2	Zählschleife	65
2.5.3	Summen- und Produkt-Bildung	68
2.5.4	Iteration und Rekursion	71
2.5.5	Verschachtelte Schleifen	72
2.5.6	Wiederholschleife	74
2.5.7	while-Beispiel: e-Funktion	76
2.5.8	Schleifen verlassen	78
2.5.9	Zusammenfassung	79
2.5.10	Aufgaben	79
2.6	Felder	81
2.6.1	Matrizen	81
2.6.2	Matrix-Beispiel: sinPlot	84
2.6.3	Matrizen erzeugen	86
2.6.4	Der :-Operator und linspace	87
2.6.5	Analyse von Feldern	88
2.6.6	meshgrid	91
2.6.7	Matrix-Operatoren	93
2.6.8	Verknüpfungen	95
2.6.9	Cell-Arrays	95
2.6.10	Zusammenfassung	97
2.6.11	Aufgaben	97
2.7	Grafik	98
2.7.1	Grafiktypen	98
2.7.2	2D-Grafik	99
2.7.3	3D-Grafik	105
2.7.4	Mehrere Plots in einer figure	108
2.7.5	3D-Kurven	109
2.7.6	Grafik-Handle	110
2.7.7	Zusammenfassung	113
2.7.8	Aufgaben	114
2.8	Strukturen	114
2.8.1	Strukturierte Daten	114
2.8.2	Datenfelder	115

2.8.3	struct	116
2.8.4	Suchen	118
2.8.5	struct ändern	120
2.8.6	struct-Beispiel: CAD-Drahtmodell	122
2.8.7	Objektorientierte Programmierung	125
2.8.8	Zusammenfassung	130
2.8.9	Aufgaben	130
2.9	Dateien	131
2.9.1	Dateizugriff	131
2.9.2	Dateien lesen	132
2.9.3	Dateien schreiben	133
2.9.4	Excel-Dateien	133
2.9.5	MAT-Files	135
2.9.6	Zusammenfassung	136
2.9.7	Aufgaben	136
2.10	Strings	137
2.10.1	Character-Arrays	137
2.10.2	String-Funktionen	138
2.10.3	String-Evaluation	140
2.10.4	Zusammenfassung	141
2.10.5	Aufgaben	142
3	GUI	144
3.1	Grafische Benutzeroberfläche	144
3.1.1	Das große Warten – Callbacks	144
3.1.2	Einführung in GUIDE	146
3.1.3	Zusammenfassung	149
3.1.4	Aufgaben	149
3.2	GUI-Elemente	150
3.2.1	Fenster und Maus	150
3.2.2	GUIDE-M-File	151
3.2.3	Text-Ausgabefeld	154
3.2.4	Text-Eingabefeld	157
3.2.5	GUI-Rückgabewert	159
3.2.6	GUI-Grafikobjekt	163
3.2.7	Pop-up-Menü	164
3.2.8	Zusammenfassung	167
3.2.9	Aufgaben	167
3.3	GUI-Menüs	168
3.3.1	Menu Bar	168
3.3.2	Context Menu	172
3.3.3	Zusammenfassung	174
3.3.4	Aufgaben	174
3.4	Standarddialoge	174
3.4.1	Standarddialog-Typen	174
3.4.2	Aufgaben	177

3.5	Callback-Interaktionen.....	178
3.5.1	Maus-Interaktion	178
3.5.2	Tastatur-Interaktion.....	182
3.5.3	Zusammenfassung.....	184
3.5.4	Aufgaben.....	184
4	Anwendungen	186
4.1	Akustik: Signalverarbeitung.....	186
4.1.1	Schwingungen.....	186
4.1.2	Fourier-Transformation.....	190
4.1.3	Audio-Funktionen	193
4.1.4	Zusammenfassung.....	195
4.1.5	Aufgaben.....	195
4.2	Bildverarbeitung	196
4.2.1	RGB-Farbmodell.....	197
4.2.2	Grafikformate.....	197
4.2.3	Bilder einlesen.....	198
4.2.4	Bilder bearbeiten	201
4.2.5	Hoch- und Tiefpass	205
4.2.6	Zusammenfassung.....	209
4.2.7	Aufgaben.....	209
4.3	Spiel: Projekt Labyrinth	210
4.3.1	Projektstruktur.....	210
4.3.2	Datenbasis	211
4.3.3	Spiel laden.....	212
4.3.4	Spielfeld zeichnen	218
4.3.5	Spielablauf	220
4.3.6	Zusammenfassung.....	222
4.3.7	Aufgaben.....	222
4.4	Mathematik: Funktionen.....	223
4.4.1	Polynome	223
4.4.2	Kurvendiskussion.....	224
4.4.3	Polynom-Fit, Lineare Regression.....	226
4.4.4	Datenauswertung.....	229
4.4.5	Nullstellen	231
4.4.6	Newton-Verfahren.....	235
4.4.7	Numerische Integration	238
4.4.8	Vektorfelder	240
4.4.9	Zusammenfassung.....	242
4.4.10	Aufgaben.....	243
4.5	Physik: Differentialgleichungen.....	244
4.5.1	Federschwingung	244
4.5.2	Differentialgleichungen.....	245
4.5.3	Numerische Lösung.....	246
4.5.4	Ungedämpfte Schwingungen.....	251
4.5.5	Gedämpfte Schwingungen	254

4.5.6	Erzwungene Schwingungen.....	257
4.5.7	Randwertproblem	261
4.5.8	Zusammenfassung	266
4.5.9	Aufgaben	266
4.6	Technische Mechanik.....	268
4.6.1	Zentrales Kraftsystem.....	268
4.6.2	Lineare Gleichungssysteme.....	269
4.6.3	Zusatzaufgabe.....	271
4.6.4	Multivariate Regression.....	273
4.6.5	Zusammenfassung	275
4.6.6	Aufgaben	275
4.7	Regelungstechnik	277
4.7.1	Stehpendel	277
4.7.2	Stabilität	281
4.7.3	Eigenwerte und Eigenvektoren.....	281
4.7.4	Regelung.....	286
4.7.5	Control System Toolbox™.....	289
4.7.6	Simulink®	292
4.7.7	Zusammenfassung	297
4.7.8	Aufgaben	297
4.8	Prozess-Kommunikation, Internet.....	298
4.8.1	COM, OLE und ActiveX.....	299
4.8.2	Kontakt zum Internet Explorer.....	299
4.8.3	Java Virtual Machine (JVM).....	304
4.8.4	Zusammenfassung	308
4.8.5	Aufgaben	308
4.9	MEX – C in MATLAB	309
4.9.1	C.....	309
4.9.2	DLL.....	311
4.9.3	C-Beispiel.....	313
4.9.4	Parameterübergabe	315
4.9.5	Zusammenfassung	318
4.9.6	Aufgaben	318
5	Programmierhilfen	320
5.1	Das Programm läuft nicht!	320
5.2	Der Debugger	325
5.3	Weitere MATLAB-Tools.....	326
5.3.1	M-Lint Code Checker.....	327
5.3.2	Profiler.....	327
5.3.3	Dependency Report.....	327
5.3.4	Help Report	327
5.3.5	File Comparison Report.....	327
5.4	Zusammenfassung.....	328

12	_____	Inhalt
6	Befehlsübersicht.....	330
	Literatur	339
	Index	343

2

Programmstrukturen



2 Programmstrukturen

In Kapitel 1 haben wir ein wenig in MATLAB herumgeschnüffelt, und Sie haben auch bereits einige Möglichkeiten des Programms kennen gelernt. Nun, in Kapitel 2, sollen systematisch die zentralen Bereiche der MATLAB-Programmiersprache vorgestellt werden. Für eine vollständige Beschreibung aller Möglichkeiten, die MATLAB bietet, reicht der Umfang dieses Buches jedoch bei Weitem nicht aus. Deshalb wurden für dieses Kapitel speziell die Aspekte ausgewählt, die typischerweise auch in anderen Programmiersprachen, wie in C oder Java, die Basis der Programmierung bilden. Dazu gehören

- die zentralen Ablauf-Konstrukte: Funktion, Verzweigung, Schleife
- die Kommunikation mit der Außenwelt: Ein- und Ausgabe von Zahlen und Text, Grafikerstellung, Dateioperationen
- die erweiterten Datentypen: Feld, Struktur, String.

2.1 Funktionen

2.1.1 Eine Black Box

Sie (ich spreche mal die männlichen Leser an) besuchen Ihre Freundin und bringen Eier, Milch, Mehl, Zucker, Butter und Salz mit. Ihre Liebste verschwindet mit den Sachen in der Küche, während Sie im Zimmer warten müssen. Nach einiger Zeit kommt Ihre Freundin mit frischen Pfannkuchen zurück.

Diesen Vorgang kann man als „Black Box“ ansehen – eine „schwarze Schachtel“, in der abgedunkelt etwas passiert, was Sie von außen nicht beobachten können.

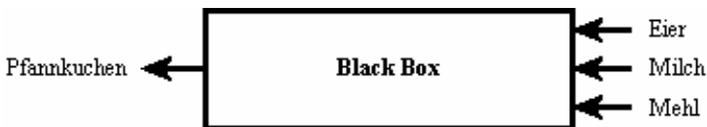


Abbildung 2.1 Black Box Pfannkuchen

Die Black Box hat *Eingangswerte*, alles was in die Box hineingelangt, in unserem Fall Eier, Milch, Mehl, Zucker, Butter und Salz – und *Rückgabewerte*, die Pfannkuchen, die in der Box auf geheimnisvolle Art entstehen (der Zutritt zur Küche wurde Ihnen nach leidvollen Erfahrungen bei der letzten Party verboten).

Auch Funktionen sind so eine Art Black Box. Nur beschickt man sie nicht mit Eiern, sondern mit Zahlen oder Texten, also unseren Daten. Aus dem Mathematik-Unterricht sollten Ihnen bereits mathematische Funktionen bekannt sein, wie Sinus, Kosinus oder die Wur-

zelfunktion, die man in Programmiersprachen oft mit *sqrt* (Square Root) bezeichnet. Wie bei der Black Box für die Pfannkuchen gibt es auch bei den Funktionen Eingangs- und Rückgabewerte – beispielsweise kann man die Zahl 9 als Eingabe an die *sqrt*-Funktion geben, die dann als Rückgabe hoffentlich die Zahl 3 liefert.



Abbildung 2.2 Black Box *sqrt*-Funktion

In MATLAB lautet dieser Aufruf, wie wir bereits gesehen haben, folgendermaßen:

```
>> sqrt( 9 )  
ans = 3
```

oder falls man sich das Ergebnis in der Variablen *y* merken möchte:

```
>> y = sqrt( 9 )  
y = 3
```

Wie MATLAB oder eine andere Programmiersprache die Wurzel von 9 berechnet, das wird dem Anwender im Allgemeinen nicht mitgeteilt. Auch hier dürfen wir die Küche der Programmierer nicht betreten und müssen darauf vertrauen, dass MATLAB einen zuverlässigen Algorithmus zur Wurzelberechnung verwendet hat.

2.1.2 Eingangs- und Rückgabeparameter

Bleiben wir noch etwas bei den Eingangs- und Rückgabewerten bzw. den dafür vorgesehenen formalen Parametern. Die meisten Funktionen haben eine fest vorgegebene Zahl von Eingangs- und Rückgabeparametern, bei der *sqrt*-Funktion sind es ein Eingangs- und ein Rückgabeparameter. Daneben sind die *Reihenfolge* der Parameter und deren Datentyp wichtig. Es gibt aber auch Funktionen, bei denen die Zahl und der Datentyp der Eingangswerte (Argumente) variabel ist und erst beim Aufruf der Funktion, also zur Laufzeit, bestimmt wird – durch den so genannten *varargs-Mechanismus*.

Daneben gibt es Funktionen, die gar *keine Eingangsparameter* haben – die Analogie wäre der Fall des Pfannkuchenbackens, wobei Sie selbst keinerlei Zutaten mitbringen, sondern auch das der Freundin überlassen. Diese Black Box hätte dann nur den Rückgabeparameter *Pfannkuchen*. Analog gibt es Funktionen, die *keine Rückgabeparameter* haben, was auf den ersten Blick kurios klingt: Was soll eine Funktion, die nichts zurückgibt. Diese Funktion wird aber normalerweise trotzdem etwas tun, aber wir, als Auftraggeber, bekommen nichts zurück – die Analogie wäre zum Beispiel der Auftrag, während unseres Urlaubs die Haustiere zu füttern, wovon wir selbst jedoch keine Rückmeldung bekommen. In zuverlässigen Projekten sollte aber jede Funktion etwas zurückgeben, zum Beispiel eine E-Mail, dass die Aufgabe erledigt wurde – Vertrauen ist gut, Kontrolle des Vorgangs besser.

2.1.3 Funktionen in MATLAB

In MATLAB haben Funktionen folgende allgemeine Form:

```
% H-Line (Hilfe zur Funktion)
function y = fname( x )
    % Anweisungen, Verarbeitung der Eingangsparameter
    ... x
    % Zuweisung des Rückgabewerts an den Rückgabeparameter
    y = ...
```

Zeilen im Funktions-Code, die mit einem *%-Zeichen* beginnen, sind *Kommentare*, die bei der Programmausführung nicht berücksichtigt werden. Sie dienen alleine als Hilfe für die Programmierer. Von spezieller Bedeutung ist dabei eine Kommentarzeile, die vor der Funktionsdeklaration steht. Diese so genannte *H-Line* enthält einen Hilfetext zur Funktion und wird von MATLAB angezeigt, wenn Sie im Command-Window die Anweisung „*help fname*“ eintippen.



In der H-Line dürfen vor dem *%-Zeichen* keine Leerzeichen stehen! Ansonsten kann die H-Line auch mehrere Zeilen lang sein, solange sie nicht durch eine Leerzeile geteilt ist. Die H-Line darf auch unterhalb der *function*-Zeile stehen.

Der eigentliche Funktions-Code beginnt in MATLAB mit dem Schlüsselwort *function*. Diese Zeile, der *Funktionskopf*, enthält der Deklaration der Funktion. Hier werden der Name der Funktion und die Reihenfolge der Rückgabe- und Eingangsparameter festgelegt. Der *Funktionsname* *fname* muss mit einem Buchstaben beginnen. Darauf kann jede beliebige Kombination von Buchstaben, Zahlen oder Unterstrichen („_“) folgen. Gibt es mehrere *Eingangsparameter* (Funktionsargumente), dann werden diese durch Kommas getrennt, zum Beispiel (*x,y,z,...*). Bei mehreren *Rückgabeparametern* verwendet man anstelle der einen Variablen *y* eine MATLAB-Liste, zum Beispiel [*a,b,c,...*], wie wir sie bereits von den Zeilenvektoren kennen. Gibt es keine Rückgabeparameter, dann entfällt der vordere Teil „*y = ...*“ in der Funktionsdeklaration, oder man ersetzt *y* durch eine leere Liste, also beispielsweise „*[] = fname(...)*“.

Unterhalb der Zeile mit dem Funktionskopf folgt der *Funktionskörper*. Hier beginnt eine beliebige Anzahl von Anweisungen, in denen die Eingangsparameter verarbeitet werden. Hat die Funktion einen Rückgabewert, muss in der Funktion irgendwo eine Anweisung erscheinen, die den Rückgabewert setzt, also zum Beispiel in der Form „*y = ...*“. Der Programmablauf innerhalb einer Funktion erfolgt *sequentiell*, das heißt, der Programm-Code wird Zeile für Zeile nacheinander abgearbeitet – außer man trifft auf einen Verzweigungspunkt (Auswahl), eine Schleife (Iteration) oder auf eine Unterfunktion.

Für jede selbst geschriebene MATLAB-Funktion, die Sie vom Command-Window aus aufrufen möchten, erzeugen Sie im Arbeitsverzeichnis eine separate Datei. Diese muss den **gleichen Namen** tragen wie die Funktion selbst und die Endung „.m“ besitzen. Der so genannte *M-File* kann nach der Hauptfunktion zusätzlich noch weitere Funktionen (*private Funktionen*) enthalten, die aber nur innerhalb dieser Datei bekannt sind. Von außen, zum Beispiel vom Command-Window aus, können private Funktionen eines M-Files nicht aufgerufen werden.

Zum *Ausführen* Ihrer MATLAB-Funktion tippen Sie den Namen der Funktionsdatei (ohne die Endung „.m“) im Command-Window ein. In Klammern folgen die Parameter, zum Beispiel „ $y = \text{sqrt}(9)$ “. Wie bereits erwähnt, kann man einer Funktion als *Argumente* beliebige *mathematische Ausdrücke* übergeben, in denen auch Variablen erlaubt sind. Diesen Variablen müssen Sie vorher natürlich einen Wert zugewiesen haben:

```
>> sqrt( 4 + 5 )
ans = 3

>> a = 15;
>> sqrt( a + 1 )
ans = 4
```

Wenn Sie sich den *Rückgabewert* einer Funktion merken wollen, darf auf der linken Seite des Gleichheitszeichens jedoch nur eine *Variable* (L-Value) stehen, aber kein Ausdruck:

```
>> r = sqrt( 4 + 5 )
r = 3

>> s + 1 = sqrt( 4 + 5 )
??? s + 1 = sqrt( 4 + 5 )
```

```
|
Error: The expression to the left of the equals sign
       is not a valid target for an assignment.
```



Im Funktionskopf „function $y = \text{fname}(x)$ “ wurde die Variable x als Übergabeparameter festgelegt. Diese Variablen im Funktionskopf bezeichnet man als „Formalparameter“, um sie von den Werten (Argumenten) zu unterscheiden, mit denen die Funktion später aufgerufen wird. Die Argumente beim Aufruf nennt man „Aktualparameter“.

Programmieren besteht hauptsächlich darin, Funktionen zu schreiben. Die Funktionen dienen zur Verarbeitung von Daten, zum Beispiel dem Addieren von Zahlen oder der Ausgabe von Text auf dem Bildschirm. Doch woher kommen diese Daten bzw. wohin werden sie nach der Bearbeitung geschickt?

Man kann Daten direkt in der Funktion definieren, etwa durch die Zuweisung $x = 5$, die den Speicherplatz x mit der Zahl 5 belegt. Oder man übergibt die Daten beim Aufruf an die Funktion mittels Eingangsparameter, wie die Zahl 9 beim Aufruf $y = \text{sqrt}(9)$. Daten können

auch zur Laufzeit vom Benutzer abgefragt werden, beispielweise von der Tastatur. Dieses Thema werden wir im Abschnitt Ein- und Ausgabe behandeln.

Zur Frage, wohin die Daten gehen: Daten können von einer Funktion zurückgegeben werden, zum Beispiel an die Variable y beim Aufruf $y = \text{sqrt}(9)$. Oder man schickt Daten an ein Ausgabemedium, wie den Bildschirm, den Drucker, oder speichert sie in einer Datei auf der Festplatte.

Ein Computerprogramm, ganz gleich wie lange es auch sein mag, besteht aus solchen einzelnen Operationen, die durch Verzweigungen und Schleifen im Ablauf strukturiert werden. Eine größere Aufgabe zerlegt man dabei in kleinere Häppchen, typischerweise in der Größe von einer Textseite Programm-Code. Theoretisch könnte man ein großes Projekt auch als lange Abfolge einzelner Anweisungen anlegen, so genannten „*Spaghetti-Code*“ programmieren, folgende Überlegungen sprechen jedoch dagegen:

- Einzelne, kurze Funktionen sind besser zu testen.
- Einzelne Funktionen können in weiteren Projekten (wieder-)verwendet werden.
- Nur die Zerlegung eines Projektes in Funktionen erlaubt ein sinnvolles Concurrent-Engineering, das Zusammenarbeiten mehrerer Mitarbeiter im Projekt.

Das Anlegen einer Funktion als Black Box ist insbesondere beim *Concurrent-Engineering* wichtig – man definiert nur Eingangs- und Rückgabeparameter und die gewünschte Funktionalität. Der Mitarbeiter, der die Funktion programmiert, hat dann relativ freie Hand bei deren Gestaltung. In der Systemdokumentation ist der Funktionsablauf zwar exakt beschrieben, die meisten Anwender der Funktion werden sich für die Einzelheiten jedoch nicht interessieren – genauso wenig, wie es für uns wichtig war, mit welchem Algorithmus der MATLAB-Programmierer die sqrt -Funktion implementiert hat.

2.1.4 Funktionsbeispiel: Umfang

Jetzt wird es aber Zeit für ein lauffähiges Beispiel einer MATLAB-Funktion. Dazu brauchen wir einen Funktionsnamen, die Eingangs- und Rückgabeparameter und natürlich die Beschreibung, was die Funktion tun soll:

- *Aufgabe* der Funktion: Die Funktion soll nach folgender Formel aus dem Kreisradius den Umfang berechnen: $Umfang = 2 * \pi * Radius$.
- *Name* der Funktion: Umfang.
- *Eingangsparameter*: r , der Radius, eine reelle Zahl.
- *Rückgabeparameter*: u , der berechnete Umfang.

Als *Hilfetext* wählen wir:

```
u = Umfang(r): Berechnung des Kreisumfangs u aus dem Kreisradius r
```

Der Funktions-Code muss in eine ASCII-Datei geschrieben werden. Hierzu verwenden wir den in MATLAB integrierten *Editor*, der im MATLAB-Menü unter „File+New+Script“ gestartet wird. Tippen Sie dort hinein den Programm-Code unserer Funktion Umfang:

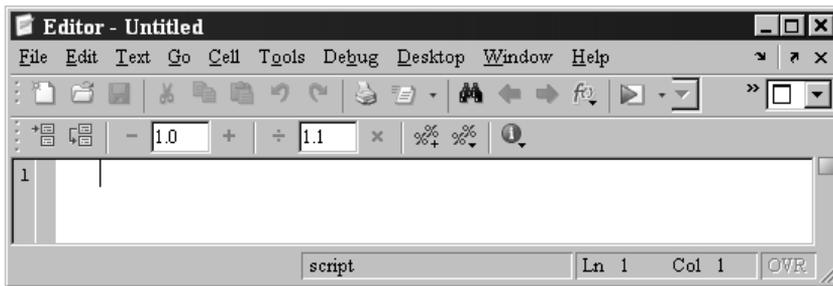


Abbildung 2.3 Der MATLAB-Editor

Listing 2.1 function Umfang

```
% u = Umfang(r):
% Berechnung des Kreisumfangs u aus dem Kreisradius r
function u = Umfang( r )
    erg = 2*pi*r;    % Berechnung des Umfangs aus Eingangswert r
    u = erg;        % Zuweisung an Rückgabewert u
```

Speichern Sie den M-File mit dem Programm-Code unter dem Namen *Umfang.m*.

In der ersten Zeile steht der Hilfetext, dann folgt der Funktionskopf mit der Deklaration des Funktionsnamens und der Parameter. Die dritte Zeile enthält die erste ausführbare Anweisung des Programms: „ $erg = 2 \cdot \pi \cdot r$ “. Hier steht rechts der Ausdruck ($2 \cdot \pi \cdot r$) und links die Variable *erg*. Diese *Wertzuweisungen* werden von *rechts nach links* interpretiert. Zuerst wird der Ausdruck ($2 \cdot \pi \cdot r$) auf der rechten Seite berechnet, dies jedoch in der Reihenfolge von links nach rechts. Die Zahl 2 wird also mit der MATLAB-Konstanten *pi* multipliziert und dann das Ergebnis mit dem übergebenen Radius aus der Variablen *r* multipliziert. Dieser berechnete Wert wird anschließend der neu definierten Variablen *erg* auf der linken Seite zugewiesen. Das Semikolon am Zeilenende dient dazu, dass MATLAB bei der Programmausführung die Protokollierung unterdrückt.

In der letzten Zeile „ $u = erg$ “ greift die Funktion wieder auf den in der Variablen *erg* gespeicherten Wert zu und kopiert ihn in den Rückgabeparameter *u*. Ohne diese Zeile würde die Funktion zwar den Wert des Umfangs berechnen, aber der Aufrufer der Funktion bekäme ihn nicht zu sehen.

Um die einzelnen Operationen, Berechnung und Rückgabe, klarer herauszuarbeiten, wurden sie in diesem Beispiel in getrennten Zeilen ausgeführt. Sie können die beiden Zeilen aber auch zu einer einzigen *zusammenfassen* und auf die Zwischenvariable *erg* verzichten:

```
% Berechnung und Rückgabe des Umfangs
u = 2*pi*r;
```

Im Command-Window von MATLAB *testen* wir die neue Funktion:

```
>> rad = 1.0;
>> umf = Umfang( rad );
>> umf
umf = 6.2832
```

Die Semikolons am Ende der beiden ersten Zeilen dienen wieder dazu, die Kontrollausgabe zu unterdrücken. Durch die dritte Zeile wird der Wert von *umf* ausgegeben.



Als Formalparameter für die Übergabe haben wir in der Funktionsdefinition von Umfang die Variable *r* verwendet. Beim Aufruf wurde das Argument *rad* mit dem Wert 1.0 als Aktualparameter an die Funktion übergeben. Wir hätten die Funktion auch direkt mit dem Wert 1.0 aufrufen können, also „*umf* = Umfang(1.0);“. Für die Rückgabe wählten wir ebenfalls unterschiedliche Namen für die formalen und die aktuellen Variablen.

2.1.5 Stack, Funktionsparameter

Es macht also keine Probleme, wenn wir beim Funktionsaufruf für Eingangs- und Rückgabebvariablen (hier *rad* und *umf*) *andere Namen* verwenden, als im Funktionskopf des M-Files Umfang.m angegeben sind. Dies ist eine typische Eigenschaft für alle Funktionen. Für den *Funktionsaufruf* (hier „>> *umf* = Umfang(*rad*);“) brauchen Sie weder zu wissen, wie der Umfang in der Funktion berechnet wird, noch welche Namen der Programmierer innerhalb der Funktion für die Eingangs- und Rückgabeparameter verwendet hat. Und Sie müssen nicht notwendigerweise Variablen für die Eingangsparameter der Funktion anlegen, sondern können die Daten auch direkt übergeben.

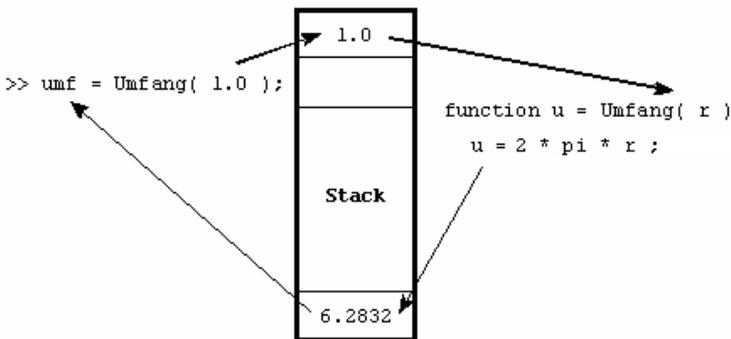


Abbildung 2.4 Stack zur Wertübergabe

Beim Funktionsaufruf werden zur Übergabe der Eingangsparameter an die Funktion gar keine Variablen verwendet. Die Daten werden stattdessen in der angegebenen Reihenfolge nacheinander in einen speziellen Speicherbereich gepackt – den *Stack*. Die aufgerufene Funktion liest die Daten aus dem Stack aus, verarbeitet sie und schreibt das Ergebnis der

Berechnungen ebenfalls in einen bestimmten Teil des Stacks zurück, von wo der Rückgabewert nach dem Funktionsaufruf abgeholt werden kann.



Wenn Sie von MATLAB aus andere MATLAB-Funktionen aufrufen, müssen Sie sich um die Art der Parameterübergabe nicht kümmern, solange Sie die korrekten Datentypen verwenden. Wie bereits erwähnt, ist MATLAB nicht typsicher. Es gibt also keinen automatischen Check, der verhindert, dass Sie beim Funktionsaufruf beispielsweise einem `double`-Parameter einen String zuweisen. Mehr noch – der Datentyp eines Funktionsparameters wird bei der Funktionsdefinition nicht einmal vereinbart.

Sie können von den meisten Programmierumgebungen aus auch Funktionen aufrufen, die in einer anderen Programmiersprache, beispielsweise in C, Java oder FORTRAN, geschrieben sind und als übersetzte Bibliotheks-Funktionen vorliegen (siehe Abschnitt 4.9 „MEX – C in MATLAB“). Hierbei müssen Sie sich aber ein paar mehr Gedanken zur Datenübergabe machen, insbesondere im Fall von komplexeren Datentypen wie mehrdimensionalen Arrays oder Strukturen.

In der Funktion `Umfang` haben wir eigene Variablen erzeugt, zum Beispiel die Variable `erg` bei der Berechnung des Umfangs. Kommt es da nicht zu Problemen, wenn auch in anderen Funktionen oder im Workspace eine Variable `erg` verwendet wird? Nein, denn auf die Variablen einer Funktion, so genannte „lokale Variablen“, kann man nur innerhalb dieser Funktion zugreifen. Ihr „*Scope*“ ist die Funktion. Eine andere Variable `erg`, die zum Beispiel im Workspace definiert ist, hat einen eigenen Speicherbereich und wird von MATLAB separat adressiert und verwaltet. Für die Funktionsparameter, hier `r` und `u`, gilt eine analoge Aussage. Ihr *Scope* ist ebenfalls auf die Funktion beschränkt. Hat das Programm eine Funktion verlassen, sind die Variablen und die Parameter der Funktion nicht mehr verfügbar. Deshalb können Sie Werte, die Sie in einer Funktion berechnet haben, nur dann außerhalb der Funktion weiter verwenden, wenn Sie diese über Rückgabeparameter aus der Funktion „exportieren“, in unserem Beispiel durch die Zuweisung des Wertes in `erg` an den Rückgabeparameter `u`, der dann über den Stack an die Variable `umf` im Workspace gelangt. Eine Ausnahme gibt es allerdings: Sie können eine Variable als *global* definieren, wie es beispielsweise im Spielprojekt im Kapitel 4 geschieht. So eine globale Variable kann dann in beliebigem Kontext verwendet werden, führt aber zu einem langsameren Programmablauf.

2.1.6 Ablaufprotokoll

Um den Ablauf unserer Funktion genauer zu verstehen, wollen wir die Zeilen der Funktion einmal einzeln durchgehen und unser Augenmerk darauf richten, wo welche Daten anfallen und gespeichert werden. Dazu dient das Ablaufprotokoll, das protokolliert, wie ein Computer unser Programm ausführen würde – nacheinander, Zeile für Zeile. Zur Orientierung haben wir die Zeilen des Funktions-Codes mit Nummern versehen. Der Aufruf der Funktion mit dem Wert 1.0 wurde der Vollständigkeit halber als Zeile 0 hinzugefügt, da hier die Übergabe an den Stack stattfindet.



Index

' – Transposition 94
% – Kommentar 36
/ – Rechts-Division 94
:-Operator und linspace 87
@ – Function Handle 231
\ – Links-Division 94
\n – Neue Zeile 47

A

Abklingkoeffizient 254
Ablaufprotokoll 41
Ablaufstrukturen 53
abs 77
Abtasttheorem 191
Achsenbeschriftung 112, 164
acker 289
Ackermann-Funktion 289
ActiveX 299
actxserver 299
Aktualparameter 37
Akustik: Signalverarbeitung 186
Alternative 56
Alternative, einfache 57
Amplitude 187
Analyse von Feldern 88
Anfangsbedingungen 245
Anfangswertproblem 246
angle 196
Antwort unterdrücken 25
Anwenderprogramme 18
Anwendungen 186
Aperiodischer Grenzfall 255

Arbeitsverzeichnis 42
Argumente 35
Array 30
ASCII-Tabelle 19
Aspect Ratio 200
Audio-Funktionen 193
audioinfo 194
audioread 194
audiorecorder 194
audiowrite 194
Ausgabe, einfache 46
Ausgabe, formatierte 47
Ausgleichsgerade 226
Äußere Schleife 72
Automatische Typzuweisung 28
axes 111, 163, 178
axis 109
axis image 200

B

Backslash 47
Bedingte Auswahl 56
Bedingungen 54
Befehlsübersicht 330
Benutzeroberfläche, grafische 144
Bessel-Filter 209
Betriebssystem 18
Bibliotheken 22
Bilder bearbeiten 201
Bilder einlesen 198
Bildverarbeitung 196
Binden 311

BIOS 16
Bit 19
Black Box 34
Bluetooth 17
bmp-Format 197
Bogenmaß 270
Boot-Vorgang 18
break 78
Breakpoint 325
Bus 17
Butterworth-Filter 209
BVP 261
bvp4c 263
bvpinit 262
Byte 19

C

C 14, 309
C++ 125, 309
CAD-Beispiel 122, 180
Callback 144, 153, 158, 170, 179
Callback-Interaktionen 178
cart2pol 242
cart2sph 242
cast 27, 80, 204
C-Beispiel 313
C-Compiler 310
cd 43
CD-Laufwerk 17
ceil 230
cell 95
cell2mat 96
Cell-Array 95, 166
char 27
Character-Array 25, 30, 137
Check Box 166
cla 213
class 127
clc 31
clear all 31
clf 105
CloseRequestFnc 161
color style marker 102

colorbar 108
colormap 108
COM 299
COM-Client 299
Command History 32
Command-Window 150
Compiler 20
COM-Server 299
Concurrent-Engineering 38
Context Menu 168, 172
continue 79
contour3 108
contourf 108
Control System Toolbox 289
conv 226
cos 270
cosd 270
CPU 16
curl 241
Current Character 182
Cursor-Tasten 183

D

date 140
Dateien 131
Dateien lesen 132
Dateien schreiben 133
Dateinamen 37, 42
Dateizugriff 131
Daten, strukturierte 114
Datenauswertung 229
Datenbasis 211
Datenfelder 115
Datenkapselung 129
Datentypen 19, 27, 115
Datentyp-Kennzeichner 48
Datenverarbeitung 16
datevec 140
DCOM 299
Debugger 325
delete 162, 171, 304
Dependency Report 327
det 94

Detail-Bild 205
deval 254
DGL 245
DGL, numerische Lösung 246
diag 86
Dialoge, modale 159
diff 241
Differentialgleichungen 245
Differentialgleichungen, gewöhnliche 246
Differentialgleichungen, partielle 246
Digitalisierung 187
DirectX 299
disp 46
divergence 241
DLL 311
doc-Funktion 23
double 27
Drahtmodell 122
3D-Grafik 105
3D-Kurven 109
DVD-Laufwerk 17
Dynamic Link Library 311

E

Edit Text 157
Editieren 20
Editor 38
e-Funktion 76
eig 281
Eigenvektoren 281
Eigenwerte 281
Ein-/Ausgabe-Beispiel: UmfangInput 51
Einführung 14
Eingabe, einfache 49
Eingangsparameter 35
elseif 58
eps 55
error 79
errorldg 175
Erzwungene Schwingungen 257
eval 140
Excel 29
Excel-Dateien 133

Exceptions 323
exe-Version 20
eye 86

F

Fadenkreuz 180
Fallunterscheidung 61
Farben invertieren 203
Fast Fourier Transform 191
fclose 132
Federschwingung 244
Fehlermeldung 24, 321
Felder 81
Fenster 150
ferror 131
Festplatte 17
fft 191
fft2 205
fgetl 132
fgets 132
FIG-File 150
figure 100, 104, 108, 111
File Comparison Report 327
File-Identifizier 131
Filter 205
findobj 111
floor 230
Flussdiagramm 53
fopen 131
for 65, 87
Formalparameter 37
format 47
Format-Anweisung 47
Formatierte Ausgabe 47
Fourier-Integral 190
Fourier-Reihe 190
Fourier-Transformation 190
fplot 99
fprintf 47, 133
fread 132
Frequenz 186
fscanf 132
function 36

Function Handle 231
Funktion 34, 53
Funktionen in MATLAB 36
Funktionsaufruf 37, 40
Funktionsbeispiel: Umfang 38
Funktionskopf 36
Funktionskörper 36
Funktionsname 36, 42
Funktionsparameter 40
Funktionsrumpf 312
fwrite 133
fzero 231

G

gca 111
gcf 111
gco 111
Gedämpfte Schwingungen 254
get 111, 158, 302
Gewöhnliche Differentialgleichungen 246
ginput 179
Gleichungen, quadratische 224
Gleichungssysteme, lineare 269
global 28, 41, 211
Grad 270
gradient 241
Grafik 98, 103
Grafiken, mehrere 103
Grafik-Export 100
Grafikformate 197
Grafik-Handle 110
Grafikkarte 17
Grafiktypen 98
Grafische Benutzeroberfläche 144
Graufilter 203
grid on 84
Grundton 188
GUI 144
guidata 153, 160
GUIDE-Einführung 146
GUIDE-M-File 151
GUI-Elemente 150
GUI-Grafikobjekt 163

GUI-Menüs 168
GUI-Rückgabewert 159

H

handles 152, 160
Hard Disc 17
Hardware 16
Header-Datei 312
hello, world 14, 30, 47, 310
Help Report 327
help-Funktion 22
Hertz 186
hidden on/off 107
H-Line 36
hObject 152
Hochpass 205
hold on/off 104
Hüllkurve 193

I

I/O 46
I/O-Kanäle 46
I/O-Karten 17
if-Abfrage 56
if-else-Abfrage 57
if-else-Beispiele 59
if-else-Strukturen, verschachtelte 58
iff2 207
imag 25
image 200, 219
Imaginäre Zahlen 25
imfinfo 197
imread 198
imwrite 201
Info 171
Innere Schleife 72
input 49
inputdlg 175
int2str 139
int32 27
Interfaces 299
Internet 298
Internet Explorer 298

Interpretierte Programme 20
inv 94
invoke 301
ischar 51, 139
isnumeric 51
isreal 224
Iteration 71
Iterationsverfahren 236

J

Java in MATLAB 317
Java Virtual Machine 304
jpg-Format 197
JVM 304

K

Kammerton a 187
KeyPressFcn 182, 220
Klasse 127
Kommentare 36
Kompilieren 311
Kompilierte Programme 20
Komponenten 116
Konstruktor 127
Kontrollstrukturen 78
Kraftsystem, zentrales 268
Kriechfall 255
Kurvendiskussion 224

L

LAN 17
lasterr 324
Layout-Editor 146
legend 104
length 89, 139
Lineare Gleichungssysteme 269
Lineare Regression 226, 273
Linken 311
Links-Division 94, 270, 274
linspace 88
Lint 327
Listbox 166
Literale 24

load 135
logical 27
Logische Verknüpfungen 55
Logischer Ausdruck 54
loglog 105
lower 139
ls 43
L-Value 37

M

MAT-Files 135
Mathematik: Funktionen 223
MATLAB 15
MATLAB als Taschenrechner 22
MATLAB aufräumen 31
MATLAB Compiler 20
MATLAB, Funktionen 36
MATLAB-Arbeitsverzeichnis 42
MATLAB-Desktop 21
MATLAB-Editor 39
MATLAB-Hilfe 23
MATLAB-Tools 326
Matrix 29, 81, 83
Matrix-Beispiel: sinPlot 84
Matrix-Funktionen 86
Matrix-Operatoren 93
Matrizen 29, 81
Matrizen erzeugen 86
Maus 150
Maus-Interaktion 178
max 229
mean 229
Memory 16
Menu Bar 168
Menu Editor 168
Menu Label 169
Menu Tag 169
mesh 107
meshgrid 91, 106
Message-Loop 144
Methoden 126, 299
MEX 309
mexFunction 312

M-File 37
min 229
Mittelwert 229
M-Lint 327
mod 230
Modale Dialoge 159
movie 267
msgbox 174
Multivariate Regression 228, 273

N

Namen 24
Netzwerkkarte 17
New Context Menu 172
New Menu 168
New Menu Item 169
Newton-Verfahren 235
Nichtmodale Dialoge 159
niz 89
norm 94
Nullstellen 231
num2cell 96
num2str 51, 139
numel 89
Numerische Integration 238
Numerische Lösung der DGL 246
Numerische Verfahren 231, 235

O

Obertöne 188
Objekte 125
Objektorientierte Programmierung 125
ODE 246
ode45 249
OLE 299
ones 86
OOP 125
OpeningFcn 152, 156, 159, 163
Optimization Toolbox 230
orderfields 122
Ordnung 246
OutputFcn 153, 160
Output-Vektor 286

P

Parameterübergabe 315
Partielle Differentialgleichungen 246
PDE 246
Peripheriegeräte 16
persistent 28
Phase 187
Physik: Differentialgleichungen 244
place 290
Platzhalter 47
plot 84, 100
plot3 109
Plots, mehrere 108
pol2cart 242
Pol-Vorgabe 290
polyder 225
polyfit 226
Polygonzugverfahren 246
polyint 226
Polynome 223
Polynom-Fit 226
polyval 223
Pop-up-Menü 164
Präfix 210
Präprozessor 312
Private Funktionen 37
Profiler 254, 327
Programm läuft nicht 320
Programmausführung 20
Programme, interpretierte 20
Programme, kompilierte 20
Programmierhilfen 320
Programmstart 18
Programmstrukturen 34
Projektstruktur 210
Prompt 22
properties 125
Property Inspector 155, 173, 179
Prozess-Kommunikation 298
Prozessor 16
Punkt-Operator 116
Push-Button 146
pwd 42

Q

quad 239
Quadratische Gleichungen 224
questdlg 175
quiver 240

R

Radio Button 166
RAM 16
rand 86, 243
randn 86, 243
Randwertproblem 261
real 25
Rechteck-Funktion 189
Rechteckregel 238
Rechts-Division 94
Regelung 286
Regelungstechnik 277
regress 275
Reibungskraft 254
Rekursion 71
Resonanz 259
return 79
RGB-Farbe 177
RGB-Farbmodell 197
rmfield 121
ROM 16
root 111
roots 224, 231
round 230
RPC 298
Rückgabeparameter 35
Runge-Kutta-Verfahren 249

S

Sample-Rate 188, 193
save 135
Schleife 53, 64
Schleife, äußere 72
Schleife, innere 72
Schleifen verlassen 78
Schleifen, verschachtelte 72
Schleifenbedingung 64

Schleifenzähler 77
Schranke eps 77
Schrittweite 247
Schwingfall 255
Schwingungen 186
Schwingungen, erzwungene 257
Schwingungen, gedämpfte 254
Schwingungen, ungedämpfte 251
Schwingungsdauer 186
Semikolon ; 25
semilogx 105
semilogy 105
Sequenz 36, 53
Services 299
set 112, 156, 180
short-circuit 56
Show Subfunctions 327
sign 196, 230
Signal Processing Toolbox 209, 230
Signalverarbeitung 186, 209, 230
Simulink 292
sin 270
sind 270
size 88
Skalarfeld 240
Skalarprodukt 94
Slider 166
Software 17
Solver 249
sort 230
Sortieren 230
sound 193
Spaghetti-Code 38
Spalte 81
Spaltenrenner 29, 82
Spaltenvektor 29, 82
Speicherbausteine 16
Speicherklasse 28
sph2cart 242
Spiel: Projekt Labyrinth 210
Sprechende Namen 27
sprintf 138
sqrt 22, 35

scanf 139, 217
Stabilität 281
Stack 40
Standardabweichung 230
Standarddialoge 174
Standarddialog-Typen 174
Static Text 154
Statistics Toolbox 275
std 230
Stehpendel 277
str2double 139
str2num 50
strcmp 314
Strg c 321
String-Evaluation 140
String-Funktionen 138
Strings 25, 137
strlen 139
strncmp 139
struct 116
struct ändern 120
struct-Beispiel: CAD-Drahtmodell 122
Strukturen 114
Strukturierte Daten 114
Stützstellen 247
subplot 108
Suchen 118
Summen- und Produkt-Bildung 68
surf 92, 105
switch 61
Syntax-Highlight 20, 321
Systemtakt 16

T

Table 166
Tastatur-Interaktion 182
Technische Mechanik 268
text 110
Text-Ausgabefeld 154
Textdarstellung 24
Texte 25, 137
Text-Editor 20
Text-Eingabefeld 157

Tiefpass 205, 208
title 104
Toggle Button 166
Toolbox V, 21, 209, 230, 239, 277, 292, 311
Totschleife 320
Transposition 94
triu, tril 94
try-catch 323
Typumwandlung 27
Typzuweisung, automatisch 28

U

UIContextMenu 173, 179
uigetfile 214
uint8 204
uiputfile 175
uiresume 159, 160, 162
uisetcolor 175
uiwait 153, 159
Ungedämpfte Schwingungen 251
upper 139
USB 17

V

varargin 152
varargout 153
varargs-Mechanismus 35, 152, 312
Variablen 26
Variablendefinition 26
Variablendeklaration 314
Variablenname 26
Vektoren 29
Vektorfelder 240
Vererbung 130
Vergleiche 54
Vergleichsoperatoren 54
Verknüpfungen 95
Verschachtelte if-else-Strukturen 58
Verschachtelte Schleifen 72
Verzweigung 53, 54
view 109
View Callbacks 161
Visible 302

W

Wertzuweisung 26, 39, 53
which 44
while 74
while-Beispiel: e-Funktion 76
whos 28
Wiederholschleife 74
Wireless-LAN 17
WLAN 17
Workspace Editor 32

X

xlabel 104
xlsread 134
xlswrite 134
XTick 112
XTickLabel 112

Z

Zahlen, imaginäre 25
Zahlendarstellung 24
Zählschleife 65
Zeile 81
Zeilenfortsetzung 61
Zeilentrenner 29, 82
Zeilenvektor 29, 82
Zeilenvorschub 47
Zelle 29, 81
zeros 83, 86
Zugriffsart 131
Zustandsbeobachtung 286
Zustandsvektor 281
2D-CAD-System 122, 180
2D-Grafik 99