HANSER

Leseprobe

Ulrich Stein

Programmieren mit MATLAB

Programmiersprache, Grafische Benutzeroberflächen, Anwendungen

ISBN (Buch): 978-3-446-44864-3

ISBN (E-Book): 978-3-446-45423-1

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-44864-3

sowie im Buchhandel.

Vorwort zur 6. Auflage

Die erste Auflage dieses Lehrbuchs erschien im Jahr 2007, also vor zehn Jahren. Im Rahmen des Umbaus vom Abschluss Dipl.-Ing. auf Bachelor und Master hatten wir damals am Department „Maschinenbau und Produktion“ der Hochschule für Angewandte Wissenschaften, am Berliner Tor in Hamburg, das Modul „Angewandte Informatik“ von der Programmiersprache C auf MATLAB® umgestellt – mit der Vorgabe, möglichst alles, was Kernighan und Ritchie in der Sprache C darlegen, mit Hilfe von MATLAB zu lehren.

Gut, bei den Zeigern haben wir versagt. Aber den Rest der Lernziele und Lehrinhalte konnten wir ohne größere Probleme übertragen. Nur fehlte ein Lehrbuch zum Programmieren mit MATLAB. Hier half der Fachbuchverlag Leipzig. Und so entstand dieses Buch, das sich in der Stoffauswahl an unserer Informatik-Vorlesung orientiert, gedacht für die Studentinnen und Studenten der Ingenieurwissenschaften in den ersten Semestern – und jetzt hoffentlich auch für Sie.

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.

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. Informationen zu diesen Themen finden Sie in Büchern zur Informatik, wie im „Informatik-Handbuch“ von P. Rechenberg und G. Pomberger.

Leider mussten auch die verketteten Listen wegfallen – dazu fehlen in MATLAB die Zeiger. Wie man dies (und andere Spezialitäten) trotzdem auch in MATLAB realisieren kann, steht in meinem zweiten Lehrbuch „Objektorientierte Programmierung mit MATLAB“.

Frau Dipl.-Ing. Erika Hotho, vom Fachbuchverlag Leipzig, machte mir vor über zehn 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 Herrn Manuel Leppert, 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 hilfreiche Tipps gaben, wie Prof. Dr.-Ing. habil. Jürgen Dankert, Prof. Dr.-Ing. Bernd Kost, Prof. Dr. rer. nat. Bernd Baumann. Nicht zu vergessen die Mitarbeiter des Rechenzentrums Berliner Tor, die mir in Fragen der MATLAB-Installation zur Seite standen. Und einen besonderen Dank an meine Frau, Elfriede Neubauer, die mir bei der stilistischen Überarbeitung eine große Hilfe war. Danke auch für die vielen Zuschriften und für die nahezu einhellig positive Reaktion der Leser.

Für die sechste Auflage wurde das Layout des Buches vollkommen umgestellt. Außerdem habe ich mehrere Abschnitte überarbeitet und erweitert, zum Beispiel die Bereiche objektorientierte Programmierung und Grafik – MATLAB hat in den letzten Jahren im Grafiksystem größere Änderungen vorgenommen. Für die numerischen Verfahren habe ich einen eigenen Abschnitt eingebaut und hierzu weitere Verfahren präsentiert.

Die im Buch beschriebenen und abgebildeten Abläufe beziehen sich auf die Bedieneroberfläche der Version **MATLAB 2017a**. 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 2017a 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 Juni 2017 Ulrich Stein

Inhalt

1 Einführung ... 13

1.1 Hello, world ... 13
1.2 Datenverarbeitung ... 15
1.2.1 Hardware ... 15
1.2.2 Software ... 17
1.2.3 Datentypen ... 18
1.2.4 Editieren ... 19
1.2.5 Programmausführung ... 19
1.3 Erster Kontakt mit MATLAB ... 20
1.3.1 Der MATLAB-Desktop ... 20
1.3.2 MATLAB als Taschenrechner ... 21
1.3.3 Zahlen- und Textdarstellung ... 23
1.3.4 Variablen und Datentypen ... 24
1.3.5 Vektoren und Matrizen ... 28
1.3.6 MATLAB aufräumen ... 31
1.3.7 Zusammenfassung ... 31
1.3.8 Aufgaben ... 31

2 Programmstrukturen ... 33

2.1 Funktionen ... 33
2.1.1 Eine Black Box ... 33
2.1.2 Eingangs- und Rückgabeparameter ... 34
2.1.3 Funktionen in MATLAB ... 35
2.1.4 Funktionsbeispiel: Umfang ... 37
2.1.5 Stack, Funktionsparameter ... 39
2.1.6 Ablaufprotokoll ... 42
2.1.7 MATLAB-Arbeitsverzeichnis ... 42
2.1.8 Zusammenfassung ... 45
2.1.9 Aufgaben ... 45
2.2 Ein- und Ausgabe ... 46
2.2.1 I/O-Kanäle ... 46
2.2.2 Einfache Ausgabe ... 47
2.2.3 Formatierte Ausgabe ... 47
2.2.4 Einfache Eingabe ... 49
2.2.5 Ein-/Ausgabe-Beispiel: UmfangInput ... 51

2.2.6 varargs-Mechanismus ... 52
2.2.7 Zusammenfassung ... 53
2.2.8 Aufgaben ... 53
2.3 Ablaufstrukturen ... 54
2.4 Verzweigungen ... 55
2.4.1 Bedingungen ... 55
2.4.2 Vergleiche ... 55
2.4.3 Logische Verknüpfungen ... 56
2.4.4 Alternative ... 57
2.4.5 *if-else*-Beispiele ... 60
2.4.6 Fallunterscheidung ... 62
2.4.7 Zusammenfassung ... 64
2.4.8 Aufgaben ... 64
2.5 Schleifen ... 65
2.5.1 Schleifenbedingung ... 65
2.5.2 Zählschleife ... 66
2.5.3 Summen- und Produkt-Bildung ... 69
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 Numerische Verfahren ... 78
2.5.9 Schleifen verlassen ... 80
2.5.10 Zusammenfassung ... 81
2.5.11 Aufgaben ... 81
2.6 Felder ... 82
2.6.1 Matrizen ... 82
2.6.2 Matrix-Beispiel: sinPlot ... 85
2.6.3 Matrizen erzeugen ... 87
2.6.4 Der :-Operator und linspace ... 88
2.6.5 Analyse von Feldern ... 90
2.6.6 meshgrid ... 91
2.6.7 Matrix-Operatoren ... 94
2.6.8 Verknüpfungen ... 96
2.6.9 Cell-Arrays ... 96
2.6.10 Zusammenfassung ... 98
2.6.11 Aufgaben ... 99
2.7 Grafik ... 100
2.7.1 Grafiktypen ... 100
2.7.2 2D-Grafik ... 100
2.7.3 3D-Grafik ... 108
2.7.4 Mehrere Plots in einer figure ... 111
2.7.5 3D-Kurven ... 113
2.7.6 Grafik-Handle ... 114
2.7.7 Zusammenfassung ... 117
2.7.8 Aufgaben ... 118

2.8 Strukturen und Klassen ... 119
2.8.1 Strukturierte Daten ... 119
2.8.2 Datenfelder ... 120
2.8.3 struct ... 120
2.8.4 Suchen ... 122
2.8.5 struct ändern ... 124
2.8.6 *struct*-Beispiel: CAD-Drahtmodell ... 125
2.8.7 Objektorientierte Programmierung (OOP) ... 128
2.8.8 Zusammenfassung ... 134
2.8.9 Aufgaben ... 135
2.9 Dateien ... 135
2.9.1 Dateizugriff ... 135
2.9.2 Dateien lesen ... 137
2.9.3 Dateien schreiben ... 137
2.9.4 Excel-Dateien ... 138
2.9.5 MAT-Files ... 140
2.9.6 Zusammenfassung ... 141
2.9.7 Aufgaben ... 141
2.10 Strings ... 142
2.10.1 Character-Arrays ... 142
2.10.2 String-Funktionen ... 144
2.10.3 String-Evaluation ... 147
2.10.4 Zusammenfassung ... 148
2.10.5 Aufgaben ... 149

3 GUI ... 151

3.1 Grafische Benutzeroberfläche ... 151
3.1.1 Das große Warten – Callbacks ... 151
3.1.2 Einführung in GUIDE ... 153
3.1.3 Zusammenfassung ... 156
3.1.4 Aufgaben ... 156
3.2 GUI-Elemente ... 156
3.2.1 Fenster und Maus ... 156
3.2.2 GUIDE-M-File ... 157
3.2.3 Text-Ausgabefeld ... 161
3.2.4 Text-Eingabefeld ... 164
3.2.5 GUI-Rückgabewert ... 166
3.2.6 GUI-Grafikobjekt ... 170
3.2.7 Pop-up-Menü ... 171
3.2.8 Zusammenfassung ... 173
3.2.9 Aufgaben ... 174
3.3 GUI-Menüs ... 174
3.3.1 Menu Bar ... 175
3.3.2 Context Menu ... 178
3.3.3 Zusammenfassung ... 180
3.3.4 Aufgaben ... 181

3.4 Standarddialoge ... 181
3.4.1 Standarddialog-Typen ... 181
3.4.2 Aufgaben ... 185
3.5 Callback-Interaktionen ... 185
3.5.1 Maus-Interaktion ... 185
3.5.2 Tastatur-Interaktion ... 189
3.5.3 Zusammenfassung ... 191
3.5.4 Aufgaben ... 191

4 Anwendungen ... 192

4.1 Akustik: Signalverarbeitung ... 192
4.1.1 Schwingungen ... 192
4.1.2 Fourier-Transformation ... 196
4.1.3 Audio-Funktionen ... 200
4.1.4 Zusammenfassung ... 201
4.1.5 Aufgaben ... 201
4.2 Bildverarbeitung ... 203
4.2.1 RGB-Farbmodell ... 203
4.2.2 Grafikformate ... 203
4.2.3 Bilder einlesen ... 204
4.2.4 Bilder bearbeiten ... 207
4.2.5 Hoch- und Tiefpass ... 211
4.2.6 Zusammenfassung ... 214
4.2.7 Aufgaben ... 215
4.3 Spiel: Projekt Labyrinth ... 215
4.3.1 Projektstruktur ... 215
4.3.2 Datenbasis ... 217
4.3.3 Spiel laden ... 218
4.3.4 Spielfeld zeichnen ... 223
4.3.5 Spielablauf ... 225
4.3.6 Zusammenfassung ... 226
4.3.7 Aufgaben ... 226
4.4 Mathematik: Funktionen ... 227
4.4.1 Polynome ... 227
4.4.2 Kurvendiskussion ... 228
4.4.3 Polynom-Fit, Lineare Regression ... 230
4.4.4 Datenauswertung ... 234
4.4.5 Nullstellen ... 235
4.4.6 Newton-Verfahren ... 239
4.4.7 Numerische Integration ... 242
4.4.8 Vektorfelder ... 244
4.4.9 Zusammenfassung ... 246
4.4.10 Aufgaben ... 247
4.5 Physik: Differentialgleichungen ... 248
4.5.1 Federschwingung ... 248
4.5.2 Differentialgleichungen ... 249

4.5.3 Numerische Lösung ... 251
4.5.4 Ungedämpfte Schwingungen ... 255
4.5.5 Gedämpfte Schwingungen ... 258
4.5.6 Erzwungene Schwingungen ... 261
4.5.7 Randwertproblem ... 265
4.5.8 Zusammenfassung ... 269
4.5.9 Aufgaben ... 270
4.6 Technische Mechanik ... 271
4.6.1 Zentrales Kraftsystem ... 271
4.6.2 Lineare Gleichungssysteme ... 272
4.6.3 Zusatzaufgabe ... 274
4.6.4 Gaußsches Eliminationsverfahren ... 276
4.6.5 Multivariate Regression ... 279
4.6.6 Zusammenfassung ... 281
4.6.7 Aufgaben ... 281
4.7 Regelungstechnik ... 283
4.7.1 Stehpendel ... 283
4.7.2 Stabilität ... 287
4.7.3 Eigenwerte und Eigenvektoren ... 288
4.7.4 Regelung ... 291
4.7.5 Control System Toolbox™ ... 295
4.7.6 Simulink® ... 297
4.7.7 Zusammenfassung ... 303
4.7.8 Aufgaben ... 303
4.8 Prozess-Kommunikation, Internet ... 304
4.8.1 COM, OLE und ActiveX ... 305
4.8.2 Kontakt zum Internet Explorer ... 305
4.8.3 Java Virtual Machine (JVM) ... 310
4.8.4 Zusammenfassung ... 313
4.8.5 Aufgaben ... 314
4.9 MEX – C in MATLAB ... 314
4.9.1 C ... 315
4.9.2 DLL ... 317
4.9.3 C-Beispiel ... 318
4.9.4 Parameterübergabe ... 320
4.9.5 Zusammenfassung ... 323
4.9.6 Aufgaben ... 323

5 Programmierhilfen ... 324

5.1 Das Programm läuft nicht! ... 324
5.2 Der Debugger ... 328
5.3 Weitere MATLAB-Tools ... 330
5.3.1 M-Lint Code Checker ... 330
5.3.2 Profiler ... 330
5.3.3 Dependency Report ... 331

5.3.4 Help Report 331
5.3.5 File Comparison Report 331
5.4 Zusammenfassung 331

6 Befehlsübersicht 333

Literatur 343

Index 347

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, Klasse, String.

2.1 Funktionen

Funktionen dienen dazu, Teilprobleme zu lösen. Dazu übergibt man ihnen eine Anzahl von Daten, aus denen die Funktion die nötigen Ergebnisse berechnet und diese dem Aufrufer zurückgibt.

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.

Bild 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 Wurzelfunktion, 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.

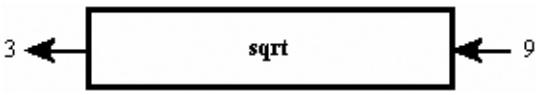

Bild 2.2 Black Box *sqrt*-Funktion

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.

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 zuverläs-

sigen 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 )
  % Verarbeitung der Eingangsparameter
  ... x
  % Zuweisung an den Rückgabeparameter
  y = ...
end   % Ende der Funktions-Definition
```

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 die 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 in der Funktionsdeklaration der vordere Teil `y = ...`, 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.

MATLAB kennt, im Gegensatz zur Programmiersprache C++, weder Zeiger noch Referenzen. Die Übergabe der Eingangsdaten an die Funktion erfolgt daher immer mittels Pass-by-value, das heißt, die in den Variablen gespeicherten Werte werden als Kopie an die Funktion übergeben. Sie können auf diesem Weg keine Daten an den Aufrufer zurückreichen. Auch wenn Sie die Daten innerhalb der Funktion verändern, hat dies keinerlei Einfluss auf die Variablen, mit denen die Funktion aufgerufen wurde. Eine Rückgabe erreichen Sie nur über die Zuweisung der Werte an einen Rückgabeparameter.

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.

Das abschließende *end* am Ende der Funktion kann bei einfachen Funktionen weggelassen werden. Wenn in einem M-File mehrere Funktionen definiert sind, müssen aber entweder alle mit einem *end* abgeschlossen sein oder keine.

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 = 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 nur eine Variable (L-Value) stehen, jedoch 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 = 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 = 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 = 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, die Zusammenarbeit 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 * π * *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 „Home + New Script“ gestartet wird. Tippen Sie dort hinein den Programm-Code unserer Funktion *Umfang*:

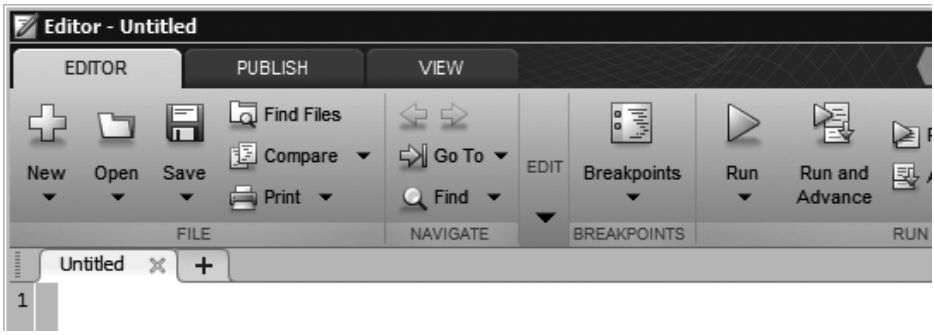

Bild 2.3 Der MATLAB-Editor

Listing 2.1 Funktion *Umfang*

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

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

In den ersten beiden Zeilen steht der Hilfetext, dann folgt der Funktionskopf mit der Deklaration des Funktionsnamens und der Parameter. Die vierte Zeile enthält die erste ausführbare Anweisung des Programms: `erg = 2*pi*r;`. Hier steht rechts der Ausdruck `2*pi*r` und links die Variable *erg*. Diese **Wertzuweisungen** werden von rechts nach links interpretiert. Zuerst wird der Ausdruck `2*pi*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 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 Funktionsdeklaration 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 die Eingangs- und Rückgabevariablen (hier *rad* und *umf*) **andere Namen** verwenden, als im Funktionskopf von *Umfang* angegeben. 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 auch nicht notwendigerweise Variablen für die Eingangsparameter der Funktion anlegen, sondern können die Daten sogar direkt übergeben.

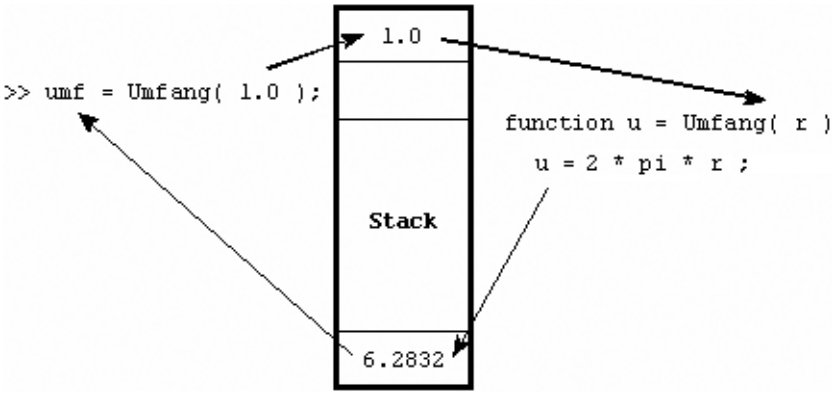

Bild 2.4 Stack zur Wertübergabe

Zur Übergabe der Eingangsparameter an die Funktion werden beim Funktionsaufruf nämlich 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 Funktionsdeklaration nicht einmal vereinbart.

Zur Laufzeit ist den Variablen in MATLAB jedoch ein wohldefinierter Datentyp zugewiesen. Wenn Sie also eine Typüberprüfung der Eingangswerte wünschen, können Sie dies selbst mittels der Funktion *isa* (besitzt den Typ) durchführen, am besten am Anfang des Funktionsrumpfes.

Man kann 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 muss man 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 die 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 weiterverwenden, wenn Sie diese über Rückgabeparameter aus der Funktion „exportieren", in unserem Bespiel durch die Zuweisung des Wertes in *erg* an den Rückgabeparameter *u*, der dann über den Stack an die Variable *umf* im Workspace gelangt. Wenn Sie beispielsweise bei der Berechnung der Daten eines Hohlzylinders zweimal den Umfang benötigen, einmal für den Innenradius 1.0 und dann für den Außenradius 2.0, dann können Sie die Funktion *Umfang* zweimal aufrufen:

```
umfInnen  = Umfang( 1.0 );
umfAussen = Umfang( 2.0 );
```

Eine Funktion ist also nicht nur eine Abfolge von Befehlen, sondern kontrolliert auch ihren eigenen Datenraum. Wenn Sie stattdessen nur eine Befehlsliste abarbeiten wollen, können Sie die Kommandos nacheinander im Command Window eingeben. Oder Sie verwenden ein **MATLAB-Script** zur Automatisierung von Befehlsfolgen. Hierzu erzeugen Sie eine neue Text-Datei, zum Beispiel mit dem Namen „UmfangScript.m", und kopieren in diese Datei die Zeilen der Funktion „Umfang.m", jedoch ohne den Funktionskopf `function u = Umfang(r)`:

Listing 2.2 Script *UmfangScript*

```
% Script UmfangScript:
%   verwendet die Workspace-Variablen r, erg, u
  erg = 2*pi*r;
  u   = erg;
```

Im Command Window können Sie das Script aufrufen:

```
>> UmfangScript;
```

Ein Script hat keine Eingabe- und Rückgabeparameter. Es verwendet die Variablen des MATLAB-Workspaces, hier den (vordefinierten Wert) von *r* und die Variable *u*, in den sie das Ergebnis schreibt. Der Aufrufer muss also genau wissen, wie die Variablen lauten, mit denen das Script arbeitet. Er muss unter anderem sicherstellen, dass er nicht selbst wichtige Daten in einer Variable *u* gespeichert hat, die durch den Aufruf des Scripts überschrieben werden. In längeren Programmen sollten Sie deshalb besser auf die Verwendung von Scripten verzichten.

Zur Übergabe von Daten an eine Funktion ohne die Verwendung von Eingabe- und Rückgabeparametern gibt es eine weitere Möglichkeit, die eine Kontrolle über den Scope ermöglicht: Sie können einzelne Parameter als *global* definieren, wie es beispielsweise im Spielprojekt im Kapitel 4 geschieht. So eine globale Variable kann dann in beliebigem Kontext verwendet werden, verlangsamt aber den Programmablauf.

Index

Symbole

2D-CAD-System 125, 188
2D-Grafik 100
3D-Grafik 108
3D-Kurven 113
@ – Function Handle 235
% – Kommentar 35
\ – Links-Division 95
\n – Neue Zeile 47
:-Operator und linspace 88
/ – Rechts-Division 95
' – Transposition 95

A

Abklingkoeffizient 259
Ablaufprotokoll 42
Ablaufstrukturen 54
abs 77
Abtasttheorem 197
Achsenbeschriftung 116, 171
Ackermann-Funktion 295
ActiveX 305
actxserver 305
Add-Ons 20
Aktualparameter 37
Akustik\Signalverarbeitung 192
Alternative 57
– einfache 59
Amplitude 193
Analyse von Feldern 90
Anfangsbedingungen 249
Anfangswertproblem 250
angle 202
anonyme Funktion 236
Antwort unterdrücken 24
Anwenderprogramme 17
Anwendungen 192
Aperiodischer Grenzfall 259
Arbeitsverzeichnis 42
Arduino 138
Argumente 34
Array 29
ASCII-Tabelle 18
Aspect Ratio 206
Attribute 129
Audio-Funktionen 200
audioinfo 200
audioread 201
audiorecorder 201
audiowrite 200
Ausgabe, einfache 47
Ausgabe, formatierte 47
Ausgleichsgerade 231
Äußere Schleife 73
Automatische Typzuweisung 27
axes 114, 170, 186
axis 112
axis image 206

B

Backslash 47
Bedingte Auswahl 58
Bedingungen 55
Befehlsübersicht 333
Benutzeroberfläche, grafische 151
Bessel-Filter 214
Betriebssystem 18
Bibliotheken 21
Bilder bearbeiten 207
Bilder einlesen 204
Bildverarbeitung 203

Binden 316
BIOS 16
Bit 18
Black Box 33
Bluetooth 16
bmp-Format 203
Bogenmaß 273
Boot-Vorgang 17
break 80
Breakpoint 328
Bus 16
Butterworth-Filter 214
BVP 265
bvp4c 266
bvpinit 266
Byte 18

C

C 13, 315
C++ 128, 315
CAD-Beispiel 125, 188
Callback 151, 160, 165, 177, 186
Callback-Interaktionen 185
cart2pol 246
cart2sph 246
cast 27, 82, 210
C-Beispiel 318
C-Compiler 315
cd 43
CD-Laufwerk 16
ceil 235
cell 96
cell2mat 97
Cell-Array 96, 173
char 27
Character-Array 24, 30, 142
Check Box 173
cla 219
class 131
clc 31
clear all 31
clf 108
CloseRequestFnc 168
colorbar 111
colormap 111
color style marker 105
COM 305
COM-Client 305
Command-Window 157
Compiler 20
COM-Server 305 f.
Concurrent-Engineering 37
Context Menu 175, 178
continue 81
contour3 111
contourf 111
Control System Toolbox 295
conv 230
cos 273
cosd 273
CPU 16
curl 245
Current Character 189
Cursor-Tasten 190

D

date 145
Dateien 135
Dateien lesen 137
Dateien schreiben 137
Dateiname 36, 42
Dateizugriff 135
Datenauswertung 234
Datenbasis 217
Datenfelder 120
Datenkapselung 132
Daten, strukturierte 119
Datentypen 18, 26, 120
Datentyp-Kennzeichner 48
datenum 146
Datenverarbeitung 15
datestr 146
datevec 146
DCOM 305
Debugger 328
delete 169, 178, 310
Dependency Report 331
det 95
deval 258
DGL 249

DGL, numerische Lösung 251
diag 87
Dialoge, modale 166
diff 245
Differentialgleichungen 249
– gewöhnliche 250
– partielle 250
Digitalisierung 193
digraph 107
DirectX 305
disp 47
divergence 245
DLL 317
doc-Funktion 22
Document Object Model 307
DOM 307
double 26
Drahtmodell 125
DVD-Laufwerk 16
Dynamic Link Library 317

E

Editieren 19
Editor 38
Edit Text 164
e-Funktion 76
eig 288
Eigenschaften 129
Eigenvektoren 288
Eigenwerte 288
Ein-/Ausgabe-Beispiel\
– UmfangInput 51
Einführung 13
Eingabe, einfache 49
Eingangsparameter 34
elseif 60
eps 56
error 80
errordlg 181
Erzwungene Schwingungen 261
Euler-Verfahren 251
eval 147
Excel 28
Excel-Dateien 138
Exceptions 327
exe-Version 20
eye 87

F

Fadenkreuz 187
Fallunterscheidung 62
Farben invertieren 209
Fast Fourier Transform 197
fclose 136
Federschwingung 248
Fehlermeldung 23, 325
Felder 82
Fenster 156
ferror 136
Festplatte 16
fft 197
fft2 211
fgetl 137
fgets 137
FIG-File 157
figure 101, 106, 111, 114
File Comparison Report 331
File-Identifier 135
Filter 211
findobj 114
floor 235
Flussdiagramm 54
fopen 135
for 66, 88
Formalparameter 37
format 47
Format-Anweisung 47
Formatierte Ausgabe 47
Fourier-Reihe 196
Fourier-Transformation 196
fplot 100
fprintf 47, 137
fread 137
fscanf 137
function 35
Function Handle 102, 235
Funktion 33, 55
Funktionen in MATLAB 35
Funktionsaufruf 36, 39

Funktionsbeispiel\
– Umfang 37
Funktionskopf 35
Funktionskörper 35
Funktionsname 35, 42
Funktionsparameter 39
Funktionsrumpf 318
fwrite 138
fzero 235

G

Gaußsches Eliminationsverfahren 276
gca 114
gcf 114
gco 114
Gedämpfte Schwingungen 258
get 115, 165, 308
Gewöhnliche Differentialgleichungen 250
ginput 187
Gleichungen, quadratische 228
Gleichungssysteme, lineare 272
global 27, 41, 217
Grad 273
gradient 245
Grafik 100, 105
Grafiken, mehrere 105
Grafik-Export 101
Grafikformate 203
Grafik-Handle 114
Grafikkarte 16
Grafiktypen 100
Grafische Benutzeroberfläche 151
graph 107
Graufilter 209
grid on 86
Grundton 194
GUI 151
guidata 160, 167
GUIDE-Einführung 153
GUIDE-M-File 157
GUI-Elemente 156
GUI-Grafikobjekt 170
GUI-Menüs 174
GUI-Rückgabewert 166

H

Handle-Klassen 130
handles 159, 167
Hard Disc 16
Hardware 15
Header-Datei 318
Hello, world 30, 47, 316
help-Funktion 21
Help Report 331
Heron-Verfahren 241
hidden on/off 110
H-Line 35
hObject 159
Hochpass 211
hold on/off 106

I

if-Abfrage 58
if-else-Abfrage 58
if-else-Beispiele 60
if-else-Strukturen, verschachtelte 59
ifft2 212
imag 24
image 206, 224
Imaginäre Zahlen 24
imfinfo 204
imread 205
imwrite 207
Info 178
Innere Schleife 73
input 49, 148
inputdlg 181
int2str 144
int32 26
integral 243
Interfaces 305
Internet 304
Internet Explorer 304
interp2 93
Interpretierte Programme 19
inv 95
invoke 307
I/O 46
I/O-Kanäle 46

isa 28, 40
ischar 51, 145
isnumeric 51
isreal 229
Iteration 71
Iterationsverfahren 240

J

Java in MATLAB 322
Java Virtual Machine 310
jpg-Format 204
JVM 310

K

KeyPressFcn 189, 225
Klasse 131
Kommentare 35
Kompilieren 316
Kompilierte Programme 19
Komponenten 121
Konstruktor 130
Kontrollstrukturen 80
Kraftsystem, zentrales 271
Kriechfall 259
Kurvendiskussion 228

L

LAN 16
lasterr 327
Layout-Editor 153
legend 107
length 90, 144 f.
Lineare Gleichungssysteme 272
Lineare Regression 230, 279
Linken 316
Links-Division 95, 273, 281
linspace 89
Lint 330
Listbox 173
Literale 23
load 140
logical 27
Logischer Ausdruck 55
Logische Verknüpfungen 56
loglog 107
lower 145
ls 43
L-Value 36

M

MAT-Files 140
Mathematik\
– Funktionen 227
MATLAB 14
MATLAB als Taschenrechner 21
MATLAB-Arbeitsverzeichnis 42
MATLAB aufräumen 31
MATLAB Compiler 20
MATLAB-Desktop 20
MATLAB-Editor 38
MATLAB, Funktionen 35
MATLAB-Hilfe 22
MATLAB-Tools 330
Matrix 29, 82, 84
Matrix-Beispiel\
– sinPlot 85
Matrix-Funktionen 87
Matrix-Operatoren 94
Matrizen 28, 82
Matrizen erzeugen 87
Maus 156
Maus-Interaktion 185
max 234
mean 234
Memory 16
Menu Bar 175
Menu Editor 175
Menu Label 176
Menu Tag 176
mesh 110
meshgrid 91, 109
Message-Loop 151
Methoden 129, 305
methods 129
MEX 314
mexFunction 317
M-File 36
min 234

Mittelwert 234
M-Lint 330
mod 235
Modale Dialoge 166
movie 271
msgbox 181
Multivariate Regression 233, 279

N

Namen 23
nargin 52
Netzwerkkarte 16
New Context Menu 178
New Menu 175
New Menu Item 176
Newton-Verfahren 239
Nichtmodale Dialoge 166
nnz 90
norm 95
Null-Device 136
Nullstellen 235
num2cell 97
num2str 51, 144
numel 90
Numerische Integration 242
Numerische Lösung der DGL 251
Numerische Verfahren 78, 235, 239, 242, 251

O

Obertöne 194
Objekte 128
Objektorientierte Programmierung 128
ODE 250
ode45 254
OLE 305
ones 87
OOP 128
OpeningFcn 159, 163, 166, 170
Optimization Toolbox 235
orderfields 125
Ordnung 250
OutputFcn 159, 167
Output-Vektor 292

P

Parameterübergabe 320
Partielle Differentialgleichungen 250
PDE 250
Peripheriegeräte 15
persistent 27
Phase 193
Physik\
– Differentialgleichungen 248
place 295
Platzhalter 47
plot 86, 102
plot3 113
Plots, mehrere 111
pol2cart 246
Pol-Vorgabe 295
polyder 229
polyfit 231
Polygonzugverfahren 251
polyint 230
Polynome 227
Polynom-Fit 230
polyval 228
Pop-up-Menü 171
Präfix 216
Präprozessor 318
Private Funktionen 36
Profiler 258, 330
Programmausführung 19
Programme, interpretierte 19
Programme, kompilierte 19
Programmierhilfen 324
Programm läuft nicht 324
Programmstart 17
Programmstrukturen 33
Projektstruktur 215
Prompt 21
properties 129
Property Inspector 161, 179, 186
Prozess-Kommunikation 304
Prozessor 16
Punkt-Operator 30, 121, 129
Push-Button 153
pwd 43

Q

Quadratische Gleichungen 228
questdlg 181
quiver 244

R

Radio Button 173
RAM 16
rand 87, 247
randn 87, 247
Random Walk 247
Randwertproblem 265
real 24
Rechteck-Funktion 195
Rechteckregel 242
Rechts-Division 95
Regelung 291
Regelungstechnik 283
regress 281
Reibungskraft 258
Rekursion 71
Resonanz 263
return 80
RGB-Farbe 184
RGB-Farbmodell 203
rmfield 125
ROM 16
root 114
roots 228, 235
round 235
RPC 304
Rückgabeparameter 34
Runge-Kutta-Verfahren 254

S

Sample-Rate 194, 200
save 140
Schleife 55, 65
– äußere 73
– innere 73
Schleifenbedingung 65
Schleifen verlassen 80
Schleifen, verschachtelte 72
Schleifenzähler 77
Schranke eps 77
Schrittweite 251
Schwingfall 259
Schwingungen 192
– erzwungene 261
– gedämpfte 258
– ungedämpfte 255
Script 41
Semikolon ; 24
semilogx 107
semilogy 107
Sequenz 35, 54
Services 305
set 115, 163, 187
short-circuit 57
Show Subfunctions 331
sign 202, 235
Signal Processing Toolbox 214, 235
Signalverarbeitung 192, 214, 235
Simulink 297
sin 273
sind 273
size 90
Skalarfeld 244
Skalarprodukt 95
Slider 173
Software 17
Solver 254
sort 234
Sortieren 234
sound 200
Spaghetti-Code 37
Spalte 82
Spaltentrenner 28, 83
Spaltenvektor 29, 83
Speicherbausteine 16
Speicherklasse 27
sph2cart 246
Spiel\
– Projekt Labyrinth 215
split 145
Sprechende Namen 26
sprintf 144
sqrt 21, 34
sscanf 145, 222
Stabilität 287

Stack 40
Standardabweichung 234
Standarddialoge 181
Standarddialog-Typen 181
Static Text 161
Statistics Toolbox 281
std 234
stderr 136
stdout 136
Stehpendel 283
str2double 144
str2num 50
strcmp 320
Strg c 325
String-Evaluation 147
String-Funktionen 144
string-Klasse 143
Strings 24, 142
strlen 145
strlength 143
strncmp 145
struct 120
struct ändern 124
struct-Beispiel\
– CAD-Drahtmodell 125
Strukturen und Klassen 119
Strukturierte Daten 119
Stützstellen 251
subplot 111
Suchen 122
Summen- und Produkt-Bildung 69
surf 93, 108
switch 63
Syntax 49
Syntax-Highlight 19, 325
Systemtakt 16

T

Table 173
Tastatur-Interaktion 189
Technische Mechanik 271
text 113
Text-Ausgabefeld 161
Textdarstellung 23
Texte 24, 142
Text-Editor 19
Text-Eingabefeld 164
Tiefpass 211, 213
title 107
Toggle Button 173
Toolbox 5, 20, 214, 235, 244, 283, 297, 317
Totschleife 324
Transposition 95
triu, tril 95
try-catch 327
Typumwandlung 27
Typzuweisung, automatisch 27

U

Überladen 133
UIContextMenu 179, 186
uigetfile 219
uint8 210
uiputfile 181
uiresume 166 f., 169
uisetcolor 181
uiwait 160, 166
Ungedämpfte Schwingungen 255
upper 145
USB 16

V

varargin 52, 159
varargout 159
varargs-Mechanismus 34, 52, 159, 317
Variablen 24
Variablendefinition 25
Variablendeklaration 319
Variablenname 25
Vektoren 28
Vektorfelder 244
Vererbung 133
Vergleiche 55
Vergleichsoperatoren 56
Verknüpfungen 96
Verschachtelte if-else-Strukturen 59
Verschachtelte Schleifen 72
Verzweigung 55

view 112
View Callbacks 168
Visible 309

W

wav-Dateien 200
Wertzuweisung 25, 38, 54
which 44
while 74
while-Beispiel\
– e-Funktion 76
whos 27
Wiederholschleife 74
Wireless-LAN 16
WLAN 16

X

xlabel 107
xlsread 139
xlswrite 139
XTick 115
XTickLabel 116

Z

Zahlendarstellung 23
Zahlen, imaginäre 24
Zählschleife 66
Zeile 82
Zeilenfortsetzung 62
Zeilentrenner 28, 83
Zeilenvektor 29, 83
Zeilenvorschub 47
Zelle 28, 82
zeros 85, 87
Zugriffsart 136
Zustandsbeobachtung 292
Zustandsvektor 287