

Wolfgang Schweizer
MATLAB® kompakt
De Gruyter Studium

Weitere empfehlenswerte Titel



Simulation physikalischer Systeme

Wolfgang Schweizer, 2017 (in Planung)

ISBN 978-3-11-046106-0, e-ISBN (PDF) 978-3-11-046186-2,

e-ISBN (EPUB) 978-3-11-046193-0



MATLAB-Simulink – Stateflow

A. Angermann, M. Beuschel, M. Rau, U. Wohlfarth, 2014

ISBN 978-3-486-77845-8, e-ISBN (PDF) 978-3-486-85910-2,

e-ISBN (EPUB) 978-3-486-98977-9



Simulation technischer linearer und nichtlinearer Systeme mit MATLAB/Simulink

Josef Hoffmann, Franz Quint, 2014

ISBN 978-3-11-034382-3, e-ISBN (PDF) 978-3-11-034383-0,

e-ISBN (EPUB) 978-3-11-034383-0



Systeme der Regelungstechnik mit MATLAB und Simulink, 2. Auflage

Helmut Bode, 2013

ISBN 978-3-486-73297-9, e-ISBN 978-3-486-76970-8

Wolfgang Schweizer

MATLAB[®] kompakt

6., aktualisierte und erweiterte Auflage

DE GRUYTER
OLDENBOURG

Autor

Prof. Dr. Wolfgang Schweizer
Tübingen
Wolfgang.Schweizer@mathworks.de

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. The MathWorks Publisher Logo identifies books that contain MATLAB and Simulink content. Used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB and Simulink software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular use of the MATLAB and Simulink software or related products. For MATLAB® and Simulink® product information, or information on other related products, please contact:

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7001
E-mail: info@mathworks.com
Web: www.mathworks.com

ISBN 978-3-11-046585-3
e-ISBN (PDF) 978-3-11-046586-0
e-ISBN (EPUB) 978-3-11-046588-4

Library of Congress Cataloging-in-Publication Data

A CIP catalog record for this book has been applied for at the Library of Congress.

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.dnb.de> abrufbar.

© 2016 Walter de Gruyter GmbH, Berlin/Boston
Druck und Bindung: CPI books GmbH, Leck
☼ Gedruckt auf säurefreiem Papier
Printed in Germany

www.degruyter.com

Vorwort zur sechsten Auflage

Die sechste Auflage wurde der aktuellen MATLAB-Version 8.6 (R2015b) angepasst, beschreibt aber auch Abweichungen der Vorgängerversionen 7.x. Seit 2006 erscheinen pro Jahr zwei Releases gekennzeichnet durch Jahr und die Buchstaben a und b. MATLAB-Version 7.2 erschien mit dem Rel. 2006a und 7.3 mit 2006b und so fort.

Wie bei den bisherigen Auflagen ist auch diese für alle diejenigen geschrieben, die nach einer kompakten und vollständigen Übersicht zu MATLAB¹ suchen.

Alle Beispiele des Buches für ein bequemes Testen sowie die Buch-Abbildungen in Farbe sind über das Internet unter <http://www.degruyter.com> zugänglich. Zusätzlich wurden noch einige weitere, ergänzende Beispiele beigefügt. Ein grafisches User-Interface – natürlich in MATLAB geschrieben – unterstützt die einfache Zuordnung der Beispiele zu den Kapiteln bzw. MATLAB-Befehlen.

Für Verbesserungsvorschläge und Hinweise auf noch vorhandene Fehler bin ich dankbar. Ich habe eine große Zahl von Zuschriften bei den vorangegangenen Auflagen erhalten, für die ich mich an dieser Stelle bedanke. Dem De Gruyter Verlag danke ich für die Bereitschaft, dieses Buch zu verlegen, Herrn Pappert für die Übernahme des Lektorats und Herrn Milla für seine Betreuung als Projekt Editor.

Tübingen

Wolfgang Schweizer

¹MATLAB® ist ein eingetragenes Warenzeichen von The MathWorks, Inc.

Vorwort zur ersten Auflage

Dieses Buch wurde für alle diejenigen geschrieben, die nach einer kompakten und vollständigen Übersicht zu MATLAB suchen.

In den vergangenen Jahren habe ich mehreren hundert Ingenieure(inne)n, Wissenschaftler(inne)n und Techniker(inne)n in Einführungs- und Fortgeschrittenenkursen die Grundlagen von MATLAB vermittelt. Dabei wurde immer wieder der Wunsch nach einer deutschsprachigen Übersicht zu MATLAB geäußert. Diese Lücke soll durch dieses Buch geschlossen werden.

Die üblichen Bücher zu MATLAB beschreiben in aller Regel entweder Anwendungen in bestimmten Bereichen oder dienen als Einführung in MATLAB. Das Ziel des vorliegenden Buches ist, neben einer knappen Einführung als vollständige Übersicht mit Beispielen zu dienen und richtet sich an alle, die MATLAB nutzen, unabhängig von ihrem jeweiligen Arbeitsgebiet, gleichgültig ob beispielsweise Finanzanwender, Wissenschaftler in Industrie oder Hochschule, Ingenieur oder Psychologe an einem Max-Planck-Institut. Die MATLAB-Dokumentation kann und will es nicht ersetzen. Die Beschreibungen basieren auf der Version 7.0.1, enthalten aber ergänzende Informationen zu den Vorgängerversionen ab Rel. 6.1. Die Beispiele und beschriebene Syntax wurden unter den Betriebssystemen Windows 2000, XP, Linux unter 32Bit und teilweise 64Bit getestet.

Da dieses Buch in meiner Freizeit entstand, möchte ich am Schluss noch meiner Frau Ursula für ihre Geduld und ihr Verständnis für manch durchschriebenes Wochenende danken.

Inhaltsverzeichnis

1	Einführung	1
1.1	Erläuterungen zum vorliegenden Text	1
1.2	Erste Schritte mit MATLAB	3
1.2.1	1. Projekt: Erzeugen von Variablen	3
1.2.2	2. Projekt: Grafiken erstellen	7
1.2.3	3. Projekt: MATLAB-Funktionen am Beispiel „Lösung eines dynamischen Systems“	11
1.2.4	4. Projekt: Polynome und Interpolationen	15
1.2.5	5. Projekt: Datenanalyse, Laden und Speichern	16
1.3	Tipps zur Effizienzsteigerung	19
1.4	Tabellarische Übersicht ausgewählter MATLAB-Kommandos	23
2	Grafische Utilities	27
2.1	Übersicht	27
2.1.1	Der MATLAB Desktop	28
2.1.2	Hilfe und Dokumentation	34
2.1.3	Eigene Applikationen und Toolboxen erstellen	36
2.1.4	Der Variable Editor	40
2.2	Der MATLAB Editor und Debugger	41
2.2.1	Der Editor	42
2.2.2	Der grafische Debugger	43
2.2.3	Berichte erstellen	44
2.3	MATLAB Code testen	48
2.4	Die Plot Tools	48
2.5	Interaktiver Daten Import	53
3	Allgemein nützliche Kommandos	55
3.1	MATLAB-Hilfe und allgemeine Informationen	55
3.1.1	Befehlsübersicht	55
3.1.2	Demos	55
3.1.3	Hilfe suchen und erstellen	55
3.1.4	M-Files nach Schlüsselbegriffen durchsuchen	56
3.1.5	Versionen	57
3.1.6	Informationen zum aktuellen Release	57
3.1.7	Variablen auflisten	57
3.2	Voreinstellungen und Verfügbarkeit von Toolboxen	58
3.3	Laden, beenden und sichern	59
3.3.1	Befehlsübersicht	59
3.3.2	Variablen löschen	59
3.3.3	MATLAB beenden: exit und quit	60
3.3.4	Variablen speichern, laden und verändern	60
3.3.5	Laden und Speichern von Objekten: loadobj und saveobj	63
3.4	Allgemeine Kommandos und Funktionen	63

3.4.1	Befehlsübersicht	63
3.4.2	Lokalisieren und auflisten	63
3.4.3	Dateien in Abhängigkeit der File-Kennung öffnen	64
3.4.4	Pseudo-Code erzeugen	64
3.4.5	Abhängigkeiten aufdecken	65
3.5	Setzen und Löschen der Suchpfade	65
3.5.1	Befehlsübersicht	65
3.5.2	Verzeichnis hinzufügen	66
3.5.3	Suchpfade	66
3.6	Kontrolle des Command Windows	67
3.6.1	Befehlsübersicht	67
3.6.2	Töne erzeugen: beep	67
3.6.3	MATLAB-Ablauf verfolgen und protokollieren	67
3.6.4	Zahlenformat setzen	68
3.7	Kommandos zum Betriebssystem	69
3.7.1	Befehlsübersicht	69
3.7.2	Informationen zum Computer	69
3.7.3	File-Handling: copyfile, movefile und delete	70
3.7.4	Verzeichnisse	71
3.7.5	Betriebssystemebene	71
3.7.6	Perl	72
3.8	Debuggen von M-Files	72
3.8.1	Befehlsübersicht	73
3.8.2	Debugger: db-Kommandos	73
3.9	Beurteilen von M-Files	76
3.9.1	Befehlsübersicht	76
3.9.2	Abhängigkeiten prüfen: depdir und depfun	76
3.9.3	Effizienz testen: Der Profiler	76
3.9.4	Test auf Probleme: checkcode, mlint und mlintprt	78
4	Operatoren und Sonderzeichen	79
4.1	Arithmetische Operatoren	79
4.1.1	Befehlsübersicht	79
4.1.2	Grundrechenarten	79
4.1.3	Berechnung der Inversen	80
4.1.4	Das Kroneckerprodukt	80
4.2	Vergleichsoperatoren	81
4.3	Logische Operatoren	81
4.4	Die bitweisen Operatoren	83
4.4.1	Befehlsübersicht	83
4.4.2	Die logischen bitweisen Operatoren	83
4.4.3	Bit-Operatoren	83
4.5	Mengen-Operatoren	84
4.5.1	Befehlsübersicht	84
4.5.2	Schnitt- und Vereinigungsmenge	84
4.5.3	Teilmengen	84
4.6	Sonderzeichen	85
4.7	Ausgewählte Variablen und Konstanten	86
5	Matrizen und numerische Arrays	89
5.1	Arrayindizes und Matrixumformungen	89
5.1.1	Befehlsübersicht	89

5.1.2	Arrayindizes	89
5.1.3	Logische Arrays und logische Indizierung	90
5.1.4	Darstellungsänderungen	91
5.1.5	Subarrays	95
5.2	Elementare Matrizen	96
5.2.1	Befehlsübersicht	96
5.2.2	Basismatrizen	96
5.2.3	Verteilungsvektoren	97
5.2.4	Vervielfachung	98
5.2.5	Frequenzvektoren	100
5.2.6	Zufallsmatrizen	100
5.3	Elementare Eigenschaften von Arrays	105
5.3.1	Befehlsübersicht	105
5.3.2	Arraygröße	105
5.3.3	Logische Arrayfunktionen	105
5.3.4	Prüfen von Arrays	106
6	Stringfunktionen	108
6.1	Zeichenketten-Funktionen	108
6.1.1	Befehlsübersicht	108
6.1.2	Zeichenketten erzeugen	109
6.1.3	Leerstellen optimieren	109
6.1.4	Konvertieren	110
6.1.5	Typen-Tests	110
6.1.6	Ausdrücke finden	110
6.1.7	Ausdrücke vergleichen	116
6.1.8	Strings zusammenfügen	116
6.1.9	Zeichen ersetzen	117
6.2	Umwandlung von Zeichenketten	117
7	Polynome, Interpolationen und Computational Geometry	119
7.1	Polynome	119
7.1.1	Befehlsübersicht	119
7.1.2	Darstellung und Auswertung von Polynomen	119
7.1.3	Polynommultiplikation und -division	121
7.1.4	Symbolische Ableitung und Integration eines Polynoms	121
7.1.5	Residuen und Polynomfit	121
7.2	Interpolation	122
7.2.1	Befehlsübersicht	123
7.2.2	Polynominterpolationen	123
7.2.3	Hermite-Interpolationspolynome	124
7.2.4	FFT-Interpolation	125
7.2.5	Spline-Interpolation	126
7.2.6	Padé-Approximation von Zeitverzögerungen	127
7.2.7	Uni- und multivariate Interpolation	128
7.2.8	Oberflächeninterpolation	131
7.3	Geometrische Analyse	134
7.3.1	Befehlsübersicht	134
7.3.2	Triangulationen	134
7.3.3	Konvexe Hülle und Voronoi-Darstellungen	139
7.3.4	Polygone	140
7.3.5	Visualisierungen und Tetraeder-Darstellungen	140

7.4	Graphen und Netzwerke	141
7.4.1	Befehlsübersicht	141
7.4.2	Konstruktion eines Graphen	141
7.4.3	Knoten und Kanten verändern	144
7.4.4	Algorithmische Graphentheorie	146
7.4.5	Matrix Darstellungen	149
7.4.6	Informationen zu Knoten	149
7.4.7	Grafische Darstellung	150
8	Datenanalyse	153
8.1	Grundlegende Datenanalyse	153
8.1.1	Befehlsübersicht	153
8.1.2	Statistische Maßzahlen	153
8.1.3	Standardabweichung	154
8.1.4	Histogramme	155
8.1.5	Sortier-Routinen	159
8.1.6	Summen und Produkte von Array-Elementen	160
8.1.7	Numerische Integration	161
8.2	Korrelation und Kovarianz	162
8.3	Finite Differenzen – numerische Ableitung	163
8.4	Winkel zwischen Unterräumen	163
8.5	Filter	163
8.5.1	Befehlsübersicht	163
8.5.2	Filterfunktionen	164
8.5.3	Faltung	165
8.5.4	Lineare Trends	166
8.6	Fourier-Transformationen	166
8.7	Zeitreihen	171
8.7.1	Befehlsübersicht	172
8.7.2	Grundlegende Eigenschaften von Zeitreihen	172
8.7.3	Daten und Zeiten manipulieren	175
8.7.4	Bearbeiten von Zeitreihen	176
8.7.5	Ereignisse festlegen	180
8.7.6	Statistische Untersuchungen	181
8.7.7	Zeitreihengruppen erzeugen und verwalten	181
8.7.8	Zeitreihengruppen bearbeiten	182
9	MATLAB als Programmiersprache	184
9.1	Entscheidungen und Schleifen	184
9.1.1	Befehlsübersicht	184
9.1.2	Schleifen: for und while	184
9.1.3	Entscheidung: if	185
9.1.4	Fallunterscheidung: switch	187
9.1.5	Ausnahmen: try und catch	187
9.1.6	Break und return	188
9.2	Ausführen von Zeichenketten und MATLAB-Ausdrücken	189
9.2.1	Befehlsübersicht	189
9.2.2	Variablenzuordnung: assignin	189
9.2.3	String-Evaluation	189
9.2.4	Funktionsausführung	190
9.3	Skripte, Funktionen und Variablen	191
9.3.1	Die MATLAB Execution Engine	191

9.3.2	Skripte	191
9.3.3	Funktionen	192
9.3.4	Globale Variablen	198
9.3.5	Persistente Variablen	198
9.3.6	Schutz von M-Files	199
9.3.7	Namenstest	199
9.4	Argumente	200
9.4.1	Befehlsübersicht	200
9.4.2	Anzahl der Funktionsargumente.....	201
9.4.3	Variable Anzahl der Funktionsargumente.....	201
9.4.4	Namen der Funktionsargumente.....	202
9.4.5	Prüfen der Eingabeargumente.....	202
9.5	Meldungen und Ausgaben	205
9.5.1	Befehlsübersicht	205
9.5.2	Ausgabe	205
9.5.3	Fehlermeldung und Warnung.....	205
9.5.4	Ausnahmen und Fehler	206
9.6	Interaktiver Input	208
10	Lineare Algebra	209
10.1	Vektoren und Matrizen	209
10.1.1	Befehlsübersicht	209
10.1.2	Die Norm	209
10.1.3	Von Spur bis Determinante	210
10.1.4	Null- und orthogonale Räume	211
10.2	Matrizen und lineare Gleichungen	213
10.2.1	Befehlsübersicht	213
10.2.2	Kondition	213
10.2.3	Matrix-Faktorisierung	214
10.2.4	Inverse, Pseudoinverse und Backslash-Operator.....	218
10.2.5	Least Square Fit	221
10.3	Modifikation von Matrix-Faktorisierungen	222
10.3.1	Befehlsübersicht	222
10.3.2	Choleski-Modifikationen: cholupdate	223
10.3.3	QR-Modifikationen	223
10.3.4	Ebene Givens-Rotationen	223
10.3.5	Diagonale und Blockdiagonale	223
10.4	Eigenwertprobleme	225
10.4.1	Befehlsübersicht	225
10.4.2	Eigenwerte	225
10.4.3	Singulärwertzerlegung	232
10.4.4	Hessenberg- und Schur-Form	232
10.5	Matrix-Funktionen.....	234
10.6	Spezielle Matrizen	236
10.6.1	Befehlsübersicht	236
10.6.2	Das charakteristische Polynom	237
10.6.3	Testmatrizen und die „Toolbox Gallery“	238
10.6.4	Hilbert-Matrizen	246
10.6.5	Ausgewählte Matrizen	246
10.6.6	Magische Quadrate	247
10.6.7	Binomialkoeffizienten	247
10.7	Logische Abfragen und Bandbreite	248

11	Optimierung, Differentialgleichungen	249
11.1	Optimierung	249
11.1.1	Befehlsübersicht	249
11.1.2	Lokale Minima	249
11.1.3	Nullstellensuche	250
11.1.4	Wahlmöglichkeiten: <code>optimset</code> und <code>optimget</code>	251
11.1.5	Parameter- und Variablensuche	252
11.2	Numerische Integration	252
11.2.1	Befehlsübersicht	252
11.2.2	Eindimensionale Integration	253
11.2.3	Mehrdimensionale Integration	255
11.3	Anfangswertprobleme	256
11.3.1	Befehlsübersicht	256
11.3.2	Allgemeine Syntax der ode-Solver	257
11.3.3	Allgemeine Solver: <code>ode45</code> , <code>ode23</code> , <code>ode113</code>	261
11.3.4	DAE und steife Probleme: <code>ode23t</code> , <code>ode15s</code>	262
11.3.5	Steife Probleme: <code>ode23tb</code> , <code>ode23s</code>	263
11.3.6	Implizite Differentialgleichungen: <code>ode15i</code>	264
11.3.7	Verzögerte Differentialgleichungssysteme	266
11.4	Randwertprobleme	269
11.5	Differentialgleichungen: Ergänzungsfunktionen	274
11.5.1	Funktionsübersicht	274
11.5.2	Differentialgleichungen: Erweiterung der Lösungen	274
11.5.3	Output Functions	275
11.6	Partielle Differentialgleichungen	275
11.6.1	Interpolation von Lösungen: <code>pdeval</code>	279
12	Dünn besetzte Matrizen	281
12.1	Elementare Matrizenoperationen	281
12.1.1	Befehlsübersicht	281
12.1.2	Erzeugen und Wandeln	282
12.1.3	Bearbeiten der Matricelemente	282
12.1.4	Speicherplatz, Funktionen und Visualisierung	283
12.1.5	Faktorisierung und Least-Square-Analyse	284
12.1.6	Parameter zu Matrix-Routinen für dünn besetzte Matrizen	285
12.2	Elementare dünn besetzte Matrizen	286
12.2.1	Befehlsübersicht	286
12.2.2	Einheitsmatrizen, diagonale dünn besetzte Matrizen	286
12.2.3	Zufallsmatrizen	287
12.3	Umordnungsalgorithmen	288
12.3.1	Befehlsübersicht	288
12.3.2	Ausgewählte Umordnungen	288
12.3.3	Optimierung von Matrix-Zerlegungen	290
12.3.4	Spalten-, Zufalls- und Farbpermutation	291
12.4	Lineare Gleichungen	292
12.4.1	Befehlsübersicht	292
12.4.2	Konjugierte Gradientenmethode	292
12.4.3	Methode der Residuen	294
12.4.4	Symmetrisches LQ-Verfahren	295
12.5	Grafische Darstellungen	295

13	Mathematische Funktionen	297
13.1	Trigonometrische Funktionen	297
13.2	Hyperbolische Funktionen	298
13.3	Exponential- und logarithmische Funktionen	298
13.4	Potenzfunktionen	299
13.4.1	Potenzen	299
13.4.2	Wurzeln	299
13.5	Rechnen mit komplexen Werten	299
13.5.1	Befehlsübersicht	299
13.5.2	Polardarstellung einer komplexen Zahl	300
13.5.3	Real- und Imaginärteil einer komplexen Zahl	300
13.5.4	Komplexbijugation	300
13.6	Rund um Zahlen	301
13.6.1	Befehlsübersicht	301
13.6.2	Runden von Zahlen	301
13.6.3	Modulus	302
13.6.4	Vorzeichen	303
13.7	Spezielle mathematische Funktionen	303
13.7.1	Befehlsübersicht	303
13.7.2	Airy- und Bessel-Funktionen	303
13.7.3	Die Gamma- und die Betafunktion	305
13.7.4	Elliptische Integrale	307
13.7.5	Fehlerintegral	308
13.7.6	Das Exponentialintegral	308
13.7.7	Legendre-Polynom	308
13.7.8	Produkte mit Vektoren	310
13.8	Zahlentheoretische Funktionen	310
13.8.1	Funktionen zur Kombinatorik	310
13.8.2	Primzahlen	311
13.8.3	Rationale Approximationen	311
13.9	Koordinaten-Transformationen	312
13.10	Farbtransformationen	312
14	2-D-Grafik	313
14.1	Elementare 2-D-Grafik	313
14.1.1	Lineare 2-D-Plots: plot	313
14.1.2	Plot mit zwei y-Achsen: plotyy	316
14.1.3	Polardarstellung: polar	316
14.1.4	Logarithmische Plots	316
14.1.5	Linieninformationen	317
14.2	Achsen und Beschriftungen	317
14.2.1	Befehlsübersicht	317
14.2.2	Achsen und ihre Eigenschaften	317
14.2.3	Mehrere Plots vereinigen: subplot	317
14.2.4	Achsen bearbeiten: axis und box	320
14.2.5	Hold	320
14.2.6	Gitter hinzufügen	321
14.2.7	Zoomen und Scrollen	322
14.2.8	Achsen beschriften	323
14.2.9	Legende und Titel	323
14.2.10	Text verarbeiten	324
14.3	Ausdruck	327

14.4	Figure Plot Tools	330
15	3-D-Grafik	331
15.1	Befehlsübersicht	331
15.1.1	Lineare 3-D-Plots: plot3	331
15.1.2	3-D-Polygone: fill3	331
15.1.3	Gitter- und Flächengrafiken	332
15.2	Achsen und Beschriftungen	332
15.2.1	Befehlsübersicht	333
15.2.2	Achsendaten und -verhältnisse	334
15.2.3	Farbbalken: colorbar	334
15.3	Farbe	337
15.3.1	Befehlsübersicht	337
15.3.2	Die Farbmatrix	337
15.3.3	Farbschattierung	338
15.3.4	Schwarz-Weiß-Monitor	339
15.4	Beleuchtung und Transparenz	339
15.4.1	Befehlsübersicht	339
15.4.2	Beleuchtung	339
15.4.3	Reflexionen	340
15.4.4	Flächennormale	340
15.4.5	Transparenz	340
15.5	Veränderung des Blickwinkels	342
15.6	Kamerakontrolle	342
15.6.1	Befehlsübersicht	342
15.6.2	Kameraposition und -steuerung	343
15.6.3	Beleuchtungskontrolle	344
16	Fortgeschrittene Grafikaufgaben	347
16.1	Funktionsplotter	347
16.1.1	Befehlsübersicht	347
16.1.2	2-D-Liniengrafiken	347
16.1.3	Konturplots	349
16.1.4	3-D-Linienplot	349
16.1.5	3-D-Grafik	349
16.2	2-D-Grafik	351
16.2.1	Befehlsübersicht	351
16.2.2	Balkendiagramme	351
16.2.3	Diskrete Daten	352
16.2.4	Polardiagramme	354
16.2.5	Streuplots	356
16.2.6	Kometenplot	357
16.2.7	Fehlerbalken	357
16.2.8	2-D-Gebiete und -Polygone	358
16.3	Höhenlinienplot	359
16.3.1	Befehlsübersicht	359
16.3.2	2-D-Konturplots	359
16.3.3	Pseudo-Farbdiagramm	360
16.3.4	3-D-Höhenlinien	361
16.4	3-D-Grafik	361
16.4.1	Befehlsübersicht	361
16.4.2	Diskrete 3-D-Daten	361

16.4.3	Kometenplots	362
16.4.4	Wasserfall-Diagramme	362
16.4.5	Gebänderte Plots	362
16.5	Visualisierung	363
16.5.1	Befehlsübersicht	363
16.5.2	Datenaufbereitung	364
16.5.3	Geschwindigkeitsabbildungen	365
16.5.4	Schnitte	366
16.5.5	Iso-Oberflächen	369
16.5.6	Strömungsdarstellung	371
16.5.7	Kegelabbildungen	374
16.5.8	Volumenfunktionen	375
16.6	Images und Farbfunktionen	375
16.6.1	Befehlsübersicht	375
16.6.2	Images	375
16.6.3	Umwandlung der Farbmatrix	377
16.6.4	Farbdithering	378
16.6.5	Farbapproximation	378
16.7	Animation	378
16.7.1	Befehlsübersicht	379
16.7.2	Erstellen einer Animation	379
16.7.3	Image-Konvertierung	380
16.8	Grafische Flächen	380
16.8.1	Befehlsübersicht	380
16.8.2	Patches	380
16.8.3	Patch-Optimierung	381
16.8.4	Geometrische Körper	382
16.9	Grafische Daten einblenden	383
17	Eigenschaften grafischer Objekte	384
17.1	Die Root- und das Figure-Objekt	385
17.1.1	Befehlsübersicht	385
17.1.2	Das Root-Objekt	386
17.1.3	Das Figure-Objekt	387
17.1.4	Grundlegende Operationen	392
17.1.5	Der OpenGL-Renderer	393
17.2	Das Achsen-Objekt	393
17.3	Ausgewählte Grafische Objekte	399
17.3.1	Befehlsübersicht	399
17.3.2	Textobjekte	400
17.3.3	Linienobjekte	401
17.3.4	Animated Line Objekte	403
17.3.5	Rechteckobjekte	404
17.3.6	Patch- und Flächenobjekte	404
17.3.7	Bildobjekte	411
17.3.8	Illustration Objekte	411
17.3.9	Annotation Objekte	413
17.3.10	Beleuchtungsobjekte	414
17.3.11	Linkeigenschaften	415
17.4	Grafische Operationen	416
17.4.1	Befehlsübersicht	416
17.4.2	Setzen und Lesen von Eigenschaften grafischer Objekte	416

17.4.3	Finden von Objekten	417
17.4.4	Informationshilfen	417
17.4.5	Handles nutzen	418
17.4.6	Auf grafische Objekte zugreifen	418
17.4.7	Anwendungsdaten	419
17.4.8	Ergänzende Grafikfunktionen	420
17.5	Hierarchische Grafik-Handles verwalten	420
17.5.1	Befehlsübersicht	420
17.5.2	Gruppen Objekte	420
17.5.3	Laden, Speichern, Exportieren	422
17.6	ActiveX-Client-Funktionen	422
18	Das Grafische User Interface	424
18.1	GUI-Funktionen	424
18.1.1	Befehlsübersicht	424
18.1.2	GUI-Objekte erzeugen	424
18.1.3	Toolbars erzeugen	434
18.1.4	Warten und Fortfahren	435
18.1.5	Interaktiver Status und Ausführungsreihenfolge	435
18.1.6	Mauseingabe	435
18.1.7	Textanpassung	436
18.1.8	Interaktive Objektwahl	436
18.1.9	Rechtecke reskalieren	436
18.2	Dialog-Boxen	437
18.2.1	Befehlsübersicht	437
18.2.2	File-Handling	437
18.2.3	Daten-Handling	438
18.2.4	Variablen-Handling	438
18.2.5	Font-Dialog	439
18.2.6	Print- und Export-Dialog	440
18.2.7	Töne, Farben, Bilder	440
18.2.8	Hilfe, Warnungen, Fehler	441
18.2.9	Dialoge	442
18.3	GUI Utilities	444
18.3.1	Befehlsübersicht	444
18.3.2	Button-Gruppen	444
18.3.3	Zwischenablage nutzen	444
18.3.4	GUI-Hilfsfunktionen	444
18.3.5	Figure-Hilfsfunktionen	445
18.4	Präferenzen	446
18.4.1	Befehlsübersicht	446
18.4.2	Präferenzen hinzufügen und entfernen	446
18.4.3	Präferenzen erhalten und setzen	447
18.4.4	Präferenz-GUIs	447
19	GUIDE	448
19.1	GUI Design Tools	448
19.1.1	GUI Option Tool	451
19.1.2	Objekte erzeugen und Eigenschaften festlegen	451
19.2	GUI M-File	452
19.2.1	Die Initialisierung	453
19.2.2	Die Opening-Funktion	454

19.2.3	Die Output-Funktion	455
19.2.4	Die Callback-Funktionen	456
19.3	UI-Menüs und Toolbars mit dem GUIDE erzeugen	458
20	File-Handling und Datenverwaltung	460
20.1	Daten- und Textdateien	460
20.1.1	Befehlsübersicht	460
20.1.2	Öffnen und Schließen von Files	460
20.1.3	Aus- und Eingabefunktionen	462
20.1.4	Lesen und Schreiben formatierter Files	468
20.1.5	Stringfunktionen	471
20.1.6	Lesen und Schreiben binärer Files	471
20.2	Big Data Analysen	474
20.2.1	Funktionsübersicht	475
20.2.2	Daten-Abbildung	475
20.2.3	Datenverwaltung	476
20.2.4	Big Data Analyse	478
20.3	Bilddateien verwalten	480
20.3.1	Funktionsübersicht	480
20.3.2	Bilddateien lesen und schreiben	480
20.3.3	Bildinformationen	482
20.3.4	Konversion zu Java	483
20.3.5	Die LibTiff Bibliothek	483
20.4	Internet-Unterstützung	486
20.4.1	Internetzugriff	486
20.4.2	E-Mail aus MATLAB schreiben	487
20.4.3	WSDL-Klassen erzeugen	488
20.5	FTP-Zugriff	488
20.6	File-Handling	489
20.6.1	Befehlsübersicht	489
20.6.2	File-Positionierung	489
20.6.3	File-Status	490
20.6.4	Temporäre Dateien und Voreinstellungen	490
20.6.5	Aufgliedern von Datei- und Funktionsnamen	490
20.6.6	Komprimierte Dateien	491
20.6.7	CDF-File-Handling	491
20.6.8	Network Common Data Format	492
20.6.9	FITS-File-Handling	494
20.6.10	XML-File-Handling	494
20.7	HDF-Bibliothek	495
20.7.1	HDF4- und HDF-EOS-Dateien	495
20.7.2	HDF5-Dateien	496
20.8	Der serielle Port	497
21	Audio- und Videoanwendungen	500
21.1	Audio Input/Output-Objekte und Hardware-Treiber	500
21.1.1	Befehlsübersicht	500
21.1.2	In- und Output-Objekte	500
21.1.3	Die VideoReader und -Writer Klasse	503
21.1.4	Tonausgabe	507
21.2	Audio- und Videodateien	507
21.2.1	Befehlsübersicht	507

21.2.2	Audio-Files bearbeiten	508
21.2.3	Hilfsfunktionen	508
21.2.4	Tonbeispiele	508
22	Datenklassen und Objekte	509
22.1	Datentypen	509
22.1.1	Fließkommazahlen	510
22.1.2	Ganzzahlige Werte	510
22.1.3	Datentypen wandeln	512
22.1.4	Strings	512
22.1.5	Java	512
22.2	Wandeln von Datentypen	513
22.2.1	Befehlsübersicht	513
22.2.2	Hexadezimaldarstellung	513
22.2.3	Binärdarstellung	514
22.2.4	Zahlendarstellung zu einer beliebigen Basis	514
22.3	Zellvariablen, Tabellen und Strukturen	514
22.3.1	Befehlsübersicht	514
22.3.2	Zellvariablen	515
22.3.3	Tabellen	516
22.3.4	Tabellen: Ein- und Ausgabe	520
22.3.5	Strukturen	522
22.4	Kategoriale Variablen	524
22.4.1	Befehlsübersicht	524
22.4.2	Kategoriale Variable erstellen	524
22.4.3	Kategoriale Variablen bearbeiten	525
22.4.4	Logische Abfragen und Informationen	526
22.5	Ergänzende Array-Funktionen	527
22.5.1	Befehlsübersicht	527
22.5.2	Anwendungsfunktionen	527
22.5.3	Zelle und Array konvertieren	529
22.5.4	Tabellen konvertieren	530
22.5.5	Zell- und Strukturvariablen	531
22.6	Objektorientierte Programmierung	532
22.6.1	Kurze Übersicht gültig vor MATLAB Rel. 7.6	532
22.6.2	Aktuelle Version: Befehlsübersicht	532
22.6.3	Die objektorientierte Programmierung	533
22.6.4	Überladene Operatoren	543
22.7	Map-Container	544
23	Zeitfunktionen	547
23.1	Basisfunktionen	547
23.1.1	Befehlsübersicht	547
23.1.2	Aktuelle Zeit	547
23.1.3	Darstellung: Datum	547
23.1.4	Datum verschieben	549
23.2	Datums- und Zeitfunktionen	549
23.2.1	Befehlsübersicht	549
23.2.2	Kalenderfunktionen	549
23.2.3	Datumsachsen plotten	550
23.2.4	Zeitdifferenz	550
23.2.5	Zeit stoppen	550

23.2.6	Pausefunktion	551
23.3	Kalendarische Operationen	551
23.3.1	Befehlsübersicht	551
23.3.2	Zeitdauer	551
23.3.3	Zeitvariablen	552
23.4	Timer Support	555
24	Verwalten und Testen	558
24.1	Versionskontrolle	558
24.1.1	Kommandos zur Versionskontrolle.....	558
24.2	Modultest	560
24.2.1	Skript- und Funktions-basiertes Testen	560
24.2.2	Testtypen	562
24.2.3	Pre- und Post-Aufgaben	564
24.2.4	Klassen-basiertes Testen	564
25	FORTRAN und C in MATLAB einbinden	568
25.1	Aufbau einer MEX-Datei	569
25.1.1	Der MEX-Befehl	570
25.2	Das mxArray	571
25.2.1	mx-Routinen zum Erstellen einfacher Variablen	572
25.2.2	mx-Routinen zum Zugriff auf einfache Variablen.....	574
25.2.3	Strukturen	575
25.2.4	Zellvariablen.....	575
25.2.5	Abfragen	575
25.2.6	Allgemeine Aufgaben	576
25.2.7	Speicherverwaltung	576
25.3	Die MEX-Funktionen	577
25.4	Die MAT-Funktionen	579
25.5	Die Engine	580
25.6	Das Generic DLL-Interface.....	581
26	Java und MATLAB	584
26.1	Vorbemerkungen zu Java	584
26.2	Java-Klassen und -Objekte.....	585
26.2.1	Java-Klassen	585
26.2.2	Java-Objekte	585
26.2.3	Java-Methoden	585
26.2.4	Objekt-Eigenschaften	586
26.3	Daten	587
26.3.1	Austausch von Daten	587
26.3.2	Java Arrays	588
26.3.3	Java-Internetanbindung	589
26.4	Java-Interface-Funktionen.....	590
27	MS-Windows-Integration	592
27.1	Die COM-Schnittstelle	592
27.1.1	MATLAB als Client.....	592
27.1.2	MATLAB als Server	596
27.2	Die Notebook-Funktionalität.....	598
28	Literaturhinweise und Internetlinks	601
	Index	605

1 Einführung

Dieses Buch richtet sich an alle MATLAB-Anwender – unabhängig von ihrem Kenntnisstand oder dem verwendeten Betriebssystem. Der Text basiert auf der MATLAB-Version 8.6, berücksichtigt aber auch die Vorgängerversionen.

1.1 Erläuterungen zum vorliegenden Text

Dieses erste Kapitel dient neben den Erläuterungen zum Aufbau des Buches einer kurzen, an konkreten Beispielen orientierten Einführung in das Arbeiten mit MATLAB. Im zweiten Kapitel werden die grundlegenden grafischen Oberflächen von MATLAB vorgestellt und im dritten allgemeine Kommandos, um beispielsweise aufzuzeigen wie die Hilfe genutzt wird. Das vierte Kapitel ist den „Operatoren und Sonderzeichen“ gewidmet, wie beispielsweise den unterstützten arithmetischen und logischen Operatoren.

Matrizen oder allgemeiner Arrays sind ein zentraler Begriff für MATLAB. Matrizen und grundlegende Operationen mit Matrizen werden im fünften und Stringfunktionen sowie Zeichenketten im sechsten Kapitel angesprochen. Polynome, Interpolationen und Fragen zur Computational Geometry sind Themen des siebten Kapitels. Hier gehen wir auf die neuen, seit dem Rel. 2015b zur Verfügung stehenden Funktionalitäten aus der Graphentheorie ein. Im achten Kapitel geht es um Datenanalyse, Filterung, Fourieranalyse und Zeitreihen.

MATLAB ist auch eine Programmiersprache mit typischen Elementen wie Schleifenkonstrukten oder bedingten Entscheidungen. MATLAB bietet sowohl die Möglichkeit Skripte als auch Funktionen selbst zu programmieren. Wichtiger Unterschied zwischen Skripten und Funktionen ist der Ort, wo die jeweiligen Variablen abgespeichert werden. Diesem Themenkomplex wendet sich das neunte Kapitel zu.

Fragen der Linearen Algebra werden im zehnten Kapitel angesprochen. Befehle zur Optimierung, wie beispielsweise Minimumsuche oder numerische Integration sowie das Lösen von Differentialgleichungen unter MATLAB werden im elften Kapitel diskutiert. Viele Matrizen haben nur wenige Elemente ungleich null. In diesen Fällen wird bei der Abspeicherung aller Elemente unnötig hoher Speicherplatz verbraucht, Berechnungen werden häufig ineffizient. Als Lösung bietet MATLAB dünn besetzte Matrizen (sparse), die mit den dazugehörigen Routinen in Kapitel zwölf behandelt werden. Mathematische Funktionen sind Gegenstand des 13. Kapitels.

Ein Bild sagt bekanntlich mehr als tausend Worte. Kapitel 14–18 ist grafisch orientierten Fragestellungen gewidmet. Zunächst wird das Erstellen elementarer zwei- und dreidimensionaler Grafiken besprochen, bevor mit fortgeschrittenen Themen wie Animationen oder Volumenvisualisierungen fortgefahren wird. Grafische Objekte sind in MATLAB

hierarchisch organisiert. Dieser Aufbau wird detailliert im Kapitel 17 besprochen und zum Abschluss der Entwurf und die Realisierung eigener grafischer Benutzeroberflächen. Dieser Themenkreis wird noch einmal im Kapitel 19 mit dem Guide aufgeworfen. Der Guide ist eine grafische Entwurfshilfe zum Erstellen grafischer Benutzeroberflächen.

Daten einlesen und schreiben, aber auch der Datenzugriff über das Internet oder einen FTP-Server werden in Kapitel 20 behandelt. Den Tönen, Audio- und Videoanwendungen ist Kapitel 21 gewidmet.

MATLAB bietet die Möglichkeit, mit Zellvariablen zu agieren, Strukturvariablen zu erzeugen, mit 8-Byte-Genauigkeit zu arbeiten, oder aber auch mit vorzeichenlosen ganzen Zahlen (unsigned integer). Dieser für viele Anwendungen wichtige Themenkomplex wird in Kapitel 22 behandelt und ergänzt durch einen knappen Überblick zur objektorientierten Programmierung. Die objektorientierte Programmierung im Detail zu besprechen, würde den Rahmen dieses Buches sprengen und soll Gegenstand eines ergänzenden zukünftigen Buches sein. Zeitfunktionen insbesondere auch die mit dem Rel. 2014b neu eingeführten kalendarischen Variablen und Operationen finden sich in Kapitel 23. Sichere und robuste Programme zu schreiben ist wichtig, gleichgültig ob man prozedural oder objektorientiert programmiert. Dieser Themenkreis finden sich im Kapitel 24 wieder.

Externe FORTRAN- oder C-Programme lassen sich in MATLAB über die MEX-Funktionalität einbinden, die in Kapitel 25 vorgestellt wird. Eine weitere Möglichkeit bietet das direkte Ansprechen dynamischer Bibliotheken; dlls unter Windows-Betriebssystemen, Shared Objects unter Unix und Linux. Diese Möglichkeit wird ebenfalls in Kapitel 25 diskutiert. Kapitel 26 zeigt als weitere externe Kommunikationsmöglichkeit die Verknüpfung mit Java auf. In Kapitel 27 wenden wir uns MS-Windows-Funktionalitäten wie ActiveX zu. MATLAB kann hier mit externen Windows-Anwendungen im Rahmen eines Server-Client-Konzepts kommunizieren und dabei sowohl die Rolle des Clients als auch des Servers übernehmen. Beschlossen wird das Buch mit einigen Literatur- und Internethinweisen.

Seit einigen Jahren können sogenannte Hardware Support Packages, beispielsweise zur Anbindung eines Arduino-Boards, herunter geladen werden. Per Download kann ebenso eine Python-Schnittstelle in MATLAB eingebunden werden. Die Vielzahl dieser Möglichkeiten würden den Rahmen dieses Buches bei Weitem sprengen.

Bei der Fülle der Themen war es nicht möglich, Teilthemen immer eindeutig einem Themenblock zuzuordnen. Teilweise mag die Zuordnung willkürlich erscheinen, teilweise hat dies zu gewollten Überlappungen und Mehrfachnennungen geführt. Innerhalb eines Themenkomplexes sind die einzelnen Begriffe teilweise nach Anwendungen, teilweise alphabetisch strukturiert. Beim Umfang dieses Buches werden sicherlich auch einige Tippfehler meine Kontrolle überlebt haben. Bei Fehlern bitte ich den Leser um Nachsicht. Allen Verbesserungsvorschlägen und Korrekturen für zukünftige Auflagen stehe ich aufgeschlossen gegenüber.

1.2 Erste Schritte mit MATLAB

MATLAB wurde Ende der siebziger Jahre für Matrix-Berechnungen entwickelt, die Bezeichnung rührt von **Matrix Laboratory** her. MATLAB bietet eine breite Palette unterschiedlicher Funktionalitäten und ist in vielen Bereichen der Industrie, Forschung und Lehre zum numerischen Standardwerkzeug geworden. Je nach gewähltem Betriebssystem öffnen Sie MATLAB durch Doppelklick auf ein Icon oder durch Aufruf aus einer Shell. MATLAB meldet sich mit der in Abb. (2.1) dargestellten Benutzeroberfläche, die als integrierte Entwicklungsumgebung dient. Bestandteil dieser Benutzeroberfläche ist das MATLAB Command Window (s. Abb. (1.1)), in dem am MATLAB-Doppelprompt die Befehlseingaben erfolgen.

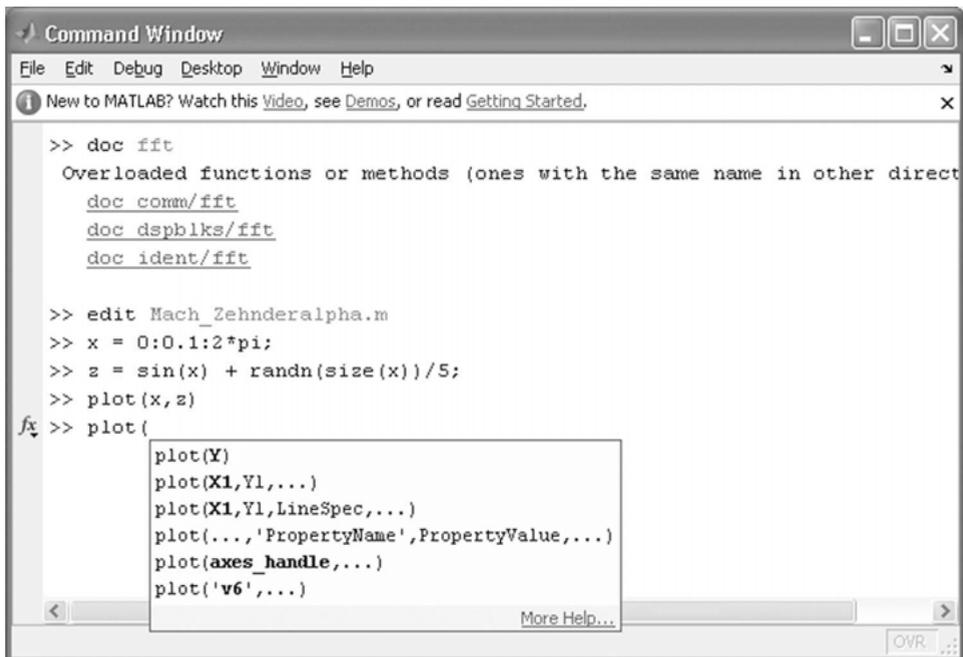


Abbildung 1.1: Das MATLAB Command Window.

1.2.1 1. Projekt: Erzeugen von Variablen

Variablen werden am MATLAB-Doppelprompt `>>` erzeugt und müssen mit einem Buchstaben beginnen. Dabei unterscheidet MATLAB zwischen Groß- und Kleinschreibung. `>> x=5;` ordnet der Variablen „x“ den Wert 5 zu. Das Semikolon unterdrückt die Ausgabe im Command Window. Per Voreinstellung ist „x“ vom Typ `double` und belegt 8 Byte Speicherplatz. String-Variablen werden mittels Hochkommas erzeugt (`>> xs='Ich bin ein String'`) und benötigen pro Element 2 Byte Speicherplatz;

```
>> x = 5;
```

```
>> xs = 'Ich bin ein String';
>> whos
  Name      Size      Bytes  Class  Attributes

  x         1x1         8  double
  xs        1x18        36  char
```

Entsprechend ergeben sich im obigen Beispiel 18 Elemente und folglich 36 Bytes. Mit `whos` können die bereits existierenden Variablen aufgelistet werden.

Matrizenmanipulation. Matrizen werden mit eckigen Klammern erzeugt, die Verwendung des Doppelpunkts erlaubt die Ausgabe von Matrixbereichen:

```
>> A=[8 1 6; 3 5 7; 4 9 2]
A =
     8     1     6
     3     5     7
     4     9     2
>> A(1,2) % 1 Element
ans =
     1
>> A(1,:) % eine Zeile
ans =
     8     1     6
>> A(:,2) % eine Spalte
ans =
     1
     5
     9
>> A(1:2,2:3) % ein Bereich
ans =
     1     6
     5     7
```

Spalten werden durch Leerzeichen oder Kommas voneinander getrennt, Zeilen durch Semikolon oder ein Return. Der erste Index bestimmt die Zeile, der zweite die Spalte. Alternativ ist auch eine lineare Indizierung möglich:

```
>> A(3)
ans =
     4
>> A([1 4 5 9])
ans =
     8     1     5     2
```

Hier wird die Matrix wie ein Spaltenvektor betrachtet (s. Kap. 5.1).

Matrizen können nicht nur vom Typ „full“ sein. Insbesondere für große dünn besetzte Matrizen ist `sparse` eine Speicherplatz sparende Alternative. Für Character Arrays muss wie für numerische Matrizen die Zeilendimension jeder Spalte gleich sein und umgekehrt. Bei Texten ist dies häufig nicht gegeben. `char` (bei älteren Releases `strvcat`) ist in solchen Fällen hilfreich.

```
>> Ac = ['abcd';'efgh';'ijkl']
Ac =
abcd
efgh
ijkl
```

```
>> Afehler=['ich bin';'ungleich';'lang']
```

```
Error using vertcat
```

```
Dimensions of matrices being concatenated are not consistent.
```

```
>> Asogehts=char('ich bin','ungleich','lang')
```

```
Asogehts =
ich bin
ungleich
lang
```

Zusätzlich zur ganzzahligen Indizierung erlaubt MATLAB auch logische Indizes. Hier steht die „0“ für falsch, alle wahren Elemente werden zurückgegeben.

```
>> A
A =
      8      1      6
      3      5      7
      4      9      2
      0.7621
      0.4565
      0.0185

>> %Indexsuche:
>> ind=find((A>5)&(A<8))
ind =
      7
      8

>> A>=5
ans =
      1      0      1
      0      1      1
      0      1      0

>> B=rand(3)
B =
      0.9501      0.4860      0.4565
      0.2311      0.8913      0.0185
      0.6068      0.7621      0.8214

>> [zeile,spalte]=find((A>5)&(A<8))
zeile =
      1
      2
spalte =
      3
      3

>> Baus=B(A>=5) % Bedingung
Baus =
      0.9501
      0.8913
```

In Kapitel 5.2 findet sich eine umfangreiche Liste elementarer Matrizen wie die Einheitsmatrix `eye` oder die Nullmatrix `zeros`. Normalverteilte Zufallsmatrizen lassen sich mit `randn` und gleichverteilte mit `rand` berechnen. `fliplr` und `flipud` führen links-rechts- bzw. oben-unten-Vertauschungen aus; transponieren lassen sich Matrizen mittels `Aht = A'` (hermitesch adjungierte Matrizen) und `At = A.'` (transponierte Matrizen).

MATLAB unterscheidet bei Matrixoperationen zwischen der elementweisen (Punkt-)

Operation und der Matrixoperation. Beispielsweise ist die Matrixmultiplikation

$$C_{i,j} = \sum_k A_{i,k} B_{k,j} \quad (1.1)$$

über $C = A * B$ definiert und die elementweise oder Arraymultiplikation

$$C_{i,j} = A_{i,j} B_{i,j} \quad (1.2)$$

über $C = A .* B$. Analoges gilt für das Potenzieren und das Teilen.

Zell-, Struktur- und Tabellenvariablen. Neben Matrizen und höher dimensionalen Arrays (mehr als 2 Indizes) unterstützt MATLAB unter anderem Zell-, Struktur- und Tabellenvariablen (cell, structure, table), die unterschiedliche Datentypen gleichzeitig verwalten können.

```

                Zellen
>> Zell={1:5,'Ich bin eine Zellvariable';A,B}
Zell =
    [1x5 double]    [1x25 char  ]
    [3x3 double]    [3x3  double]
                Strukturen
>> Str.name='Strukturvariable';
>> Str.vek=1:5;
>> Str.A = A;
>> Str.nochwas = B;

>> Str
Str =
    name: 'Strukturvariable'
    vek: [1 2 3 4 5]
    A: [3x3 double]
    nochwas: [3x3 double]
                Tabellen
>> Wer = {'Otto';'Gabi';'Udo'};
>> Alter = [27;43;25];
>> WieAlt = table(Wer,Alter)

WieAlt =
    Wer    Alter
    -----
    'Otto'    27
    'Gabi'    43
    'Udo'    25

>> WieAlt(2,:)

```

```
ans =
      Wer      Alter
-----
'Gabi'      43
```

Zellvariablen werden mit geschweiften Klammern erzeugt und Strukturvariablen durch Variablen- und Feldnamen, die durch einen Punkt voneinander getrennt sind sowie Tabellen durch das Schlüsselwort `table`.

1.2.2 2. Projekt: Grafiken erstellen

Dieser Abschnitt soll das prinzipielle Arbeiten mit einfachen grafischen Aufgaben zeigen. Tiefer gehende Details und Beschreibungen finden sich in den Kapiteln 14 bis 18.

`plot(x)` plottet den Vektor „x“ gegen seinen Index. Halblogarithmische Darstellungen lassen sich mit `semilogx` und `semilogy`, doppeltlogarithmische mit `loglog` erzeugen.

```
>> x=[0:8 7:-1:2 3:6 5 4 5 5 5];
>> plot(x)
>> plot(x, '*')
```

plottet einzelne Datenpunkte, Abb (1.2). Tabelle (14.1) listet die unterstützten Symbole für die Datenpunkte und Linientypen auf.

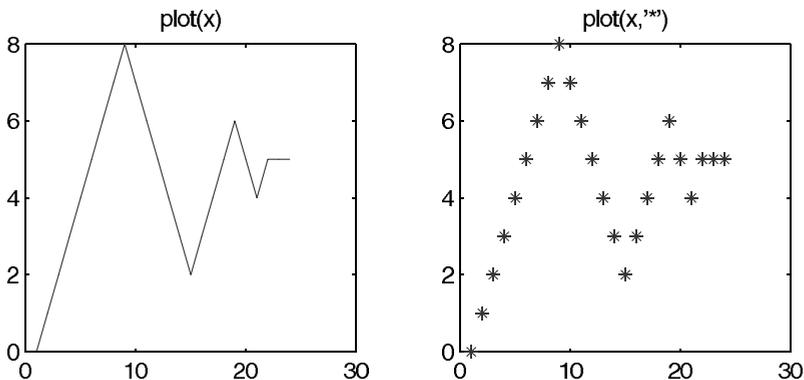


Abbildung 1.2: `>> x=[0:8 7:-1:2 3:6 5 4 5 5 5];`

Größere Punktwolken lassen sich besser mit `scatter` darstellen. Sollen mehrere Plots in einer Abbildung ausgeführt werden, so können entweder mehrere Wertepaare in einem Plotkommando übergeben werden `plot(x1,y1, x2,y2, ...)` oder das Figure Window kann mit `hold on` vor dem Löschen geschützt werden. `hold off` schaltet diese Eigenschaft wieder ab. Eine weitere Möglichkeit ist, mit `subplot` das Figure Window in einzelne Teilfenster zu unterteilen.

Betrachten wir als Beispiel eine gedämpfte Schwingung

$$y(t) = \exp(-\kappa t) \sin(\omega t + \phi) \quad \text{mit} \quad \omega = \sqrt{\omega_0^2 - \kappa^2} \quad . \quad (1.3)$$

κ ist die Dämpfung und ω_0 die dämpfungsfreie Eigenfrequenz, ϕ eine Phasenverschiebung. Dies führt zu drei unterschiedlichen Bereichen: gedämpfte Schwingung, aperiodischer Grenzfall und Kriechfall. Diese drei Fälle sollen mit MATLAB dargestellt werden.

```
omega0=1;
k=[0.2 1 3];
n=0;
for kappa=k
    n=n+1;
    omega=sqrt(omega0^2-kappa^2);
    t=0:0.01:8*pi;
    if (omega==0)
        y=exp(-kappa.*t).*t;
    else
        y=exp(-kappa.*t).*sin(omega*t)/omega;
    end
    if n==1
        subplot(2,2,1:2)
    else
        subplot(2,2,n+1)
    end
    plot(t,y)
    axis tight
    legend(['\kappa = ',num2str(kappa)])
end
```

Zuerst werden entsprechend Gl. (1.3) die Konstanten definiert. Die For-Schleife läuft über die drei unterschiedlichen Fälle, die durch die entsprechenden Gleichungen ausgewertet werden. Für den Schwingungsfall soll das Fenster doppelt so breit sein wie für die anderen beiden, Abb. (1.3). Dies wird durch `subplot(2,2,1:2)` erreicht. `subplot(n,m,q)` teilt das Figure Window in zwei Zeilen und zwei Spalten, der dritte Wert zählt die einzelnen aktiven Teilfenster von links nach rechts und von oben nach unten durch.

Dreidimensionale Liniengrafiken lassen sich mit `plot3` erzeugen. Während Liniengrafiken über Vektoren errichtet werden, werden Flächengrafiken über Arrays erzeugt. Betrachten wir das folgende Beispiel (Abb. (1.4)):

```
>> % Daten fuer 3d-Linienplot      >> % Daten fuer Flaechenplot
>> x=-1:0.05:1;                   >> [X,Y] = meshgrid(x,y);
>> y=x;                             >> Z=X.^2 + Y.^2;
>> z=x.^2 + y.^2;
```

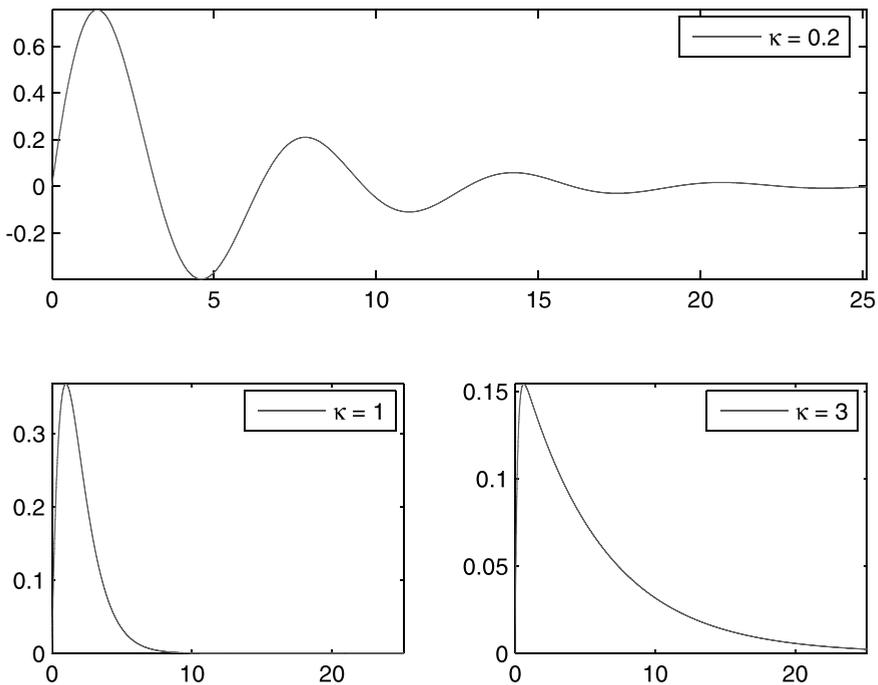


Abbildung 1.3: Die obere Abbildung stellt den Schwingungsfall dar, geplottet in den durch `subplot(2,2,1:2)` erzeugten Bereich; links unten der aperiodische Grenzfall (`subplot(2,2,3)`) und rechts unten der Kriechfall (`subplot(2,2,4)`).

```
>> whos
  Name      Size          Bytes  Class
  X         41x41          13448  double array
  Y         41x41          13448  double array
  Z         41x41          13448  double array
  x         1x41             328    double array
  y         1x41             328    double array
  z         1x41             328    double array
```

Grand total is 5166 elements using 41328 bytes

```
>> subplot(1,2,1)          >> subplot(1,2,2)
>> plot3(x,y,z)           >> surf(X,Y,Z)
```

Weitere häufig für Flächengrafiken genutzte Befehle sind `mesh(X,Y,Z)`, das eine Gittergrafik erstellt, `surf(X,Y,Z)`, das zusätzlich Konturlinien hinzufügt und `contour(X,Y,Z)` für einen Kontourplot. Weitere Details finden sich insbesondere in Kap. 15.

Neben einfachen Grafiken bietet MATLAB die Möglichkeit der Animation mit `movie`,

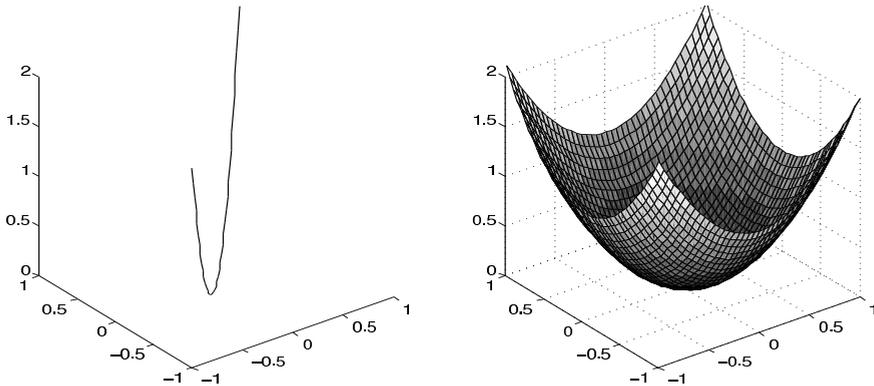


Abbildung 1.4: Links ein 3-D-Linienplot erzeugt mit `plot3(x,y,z)` und rechts die entsprechende Flächendarstellung `surf(X,Y,Z)`.

umfangreiche Volumenvisualisierungsmöglichkeiten und Vieles mehr. Als letztes Beispiel betrachten wir die Ausbreitung eines Gaußpaketes, beispielsweise elektromechanische Wellen oder auch ein quantenmechanisches Wellenpaket. Für ein solches Gaußpaket im 2-Dimensionalen gilt

$$G(x, y, t) = \frac{1}{\sqrt{1+4t^2}} \exp\left(-\frac{(x-vt)^2}{1+4t^2}\right) \exp\left(-\frac{(y-vt)^2}{1+4t^2}\right) \quad (1.4)$$

mit der Zeit t , den Koordinaten x, y und der Ausbreitungsgeschwindigkeit v . Das folgende MATLAB-Beispiel visualisiert mit $v = 3$ eine solche Wellenausbreitung.

```
t=0:0.2:10; % Zeitschritte
x=-10:0.2:40; % Koordinaten
y=x;
k=0; % Laufvariable
% 2-D-Arrays
[X,Y]=meshgrid(x,y);
% Berechnung
for T=t
    xT=X-3*T;
    yT=Y-3*T;
    Z=1./sqrt(1+4*T.^2).* ...
    exp(-2*(xT.^2+yT.^2)/(1+4*T.^2));
    % Grafik
    surf(X,Y,Z)
    shading interp
    xlim([-10 40]),ylim([-10 40])
    zlim([0 0.75])
    k=k+1;
end
```

```

F(k)=getframe; % movieframes
end
      % Abspielen
movie(F,3)

```

For-Schleifen in MATLAB erlauben die Übergabe ganzer Arrays oder wie hier im Beispiel von Vektoren. Die Festlegung der Grenzen, hier insbesondere `zlim`, sorgt dafür, dass der Verlauf erkennbar und nicht durch automatische Umskalierungen der Achsen verschleiert wird. Einige Zeitschnitte sind in Abb. (1.5) dargestellt.

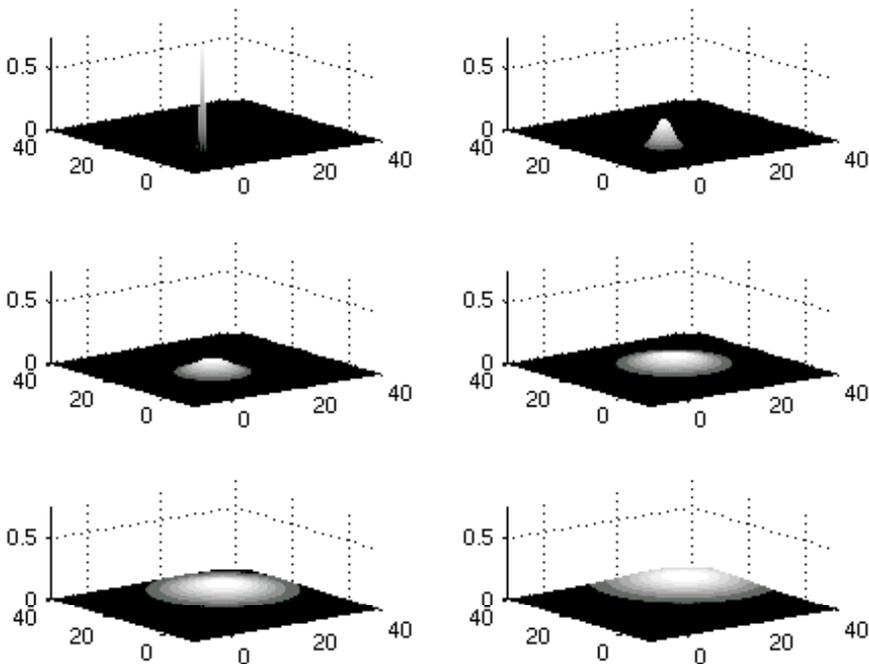


Abbildung 1.5: Einzelne Filmelemente aus dem „Movie“ zur Wellenausbreitung in Zeitschritten von 2, beginnend bei $t=0$.

Das nächste Projekt „Lösung eines dynamischen Systems“ wird ebenfalls auf grafische Darstellungen zurückgreifen.

1.2.3 3. Projekt: MATLAB-Funktionen am Beispiel „Lösung eines dynamischen Systems“

Im Rahmen dieses Beispielprojekts werden wir Skripte, MATLAB-Funktionen und „function functions“ diskutieren. Skripte und MATLAB-Funktionen sind lesbare Files, die ausführbaren Code enthalten. Skripte unterscheiden sich von Funktionen „optisch“ durch das fehlende Schlüsselwort `function`. Einer der wichtigsten Unterschiede ist der Ort, an dem die Variablen abgespeichert werden. Funktionen haben ihren eigenen lokalen

Speicherbereich. Skripte haben keinen eigenen Speicherbereich. Ihre Variablen werden entweder im Base Space, dem Speicherbereich des MATLAB Command Windows oder, falls sie von einer Funktion aufgerufen werden sollten, im Function Space der aufrufenden Funktion abgespeichert. Variablen sind stets lokal, sofern sie nicht explizit als global deklariert wurden. Function functions sind MATLAB-Funktionen, die wiederum Funktionen als Argument enthalten. Ein Beispiel dafür sind die Differentialgleichungslöser `ode...`, ein Anwendungsbeispiel sind dynamische Systeme.

Dynamische Systeme werden durch gewöhnliche Differentialgleichungen beschrieben. MATLAB bietet zur Lösung gewöhnlicher Differentialgleichungen die ode-Familie, die in Kapitel 11.3 ausführlich beschrieben ist. Als einfaches Beispiel betrachten wir das reale ebene Pendel:

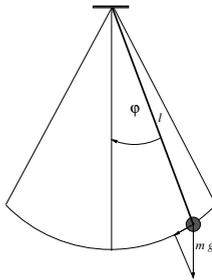


Abbildung 1.6: Das ebene Pendel.

Auf unseren (punktförmigen) Ball der Masse m wirkt die Schwerkraft mg mit g der Erdbeschleunigung. Bei Auslenkung um den Winkel φ wirkt die rücktreibende Kraft $mg \sin \varphi$ und längs der Schnur $mg \cos \varphi$. Damit erhalten wir die Bewegungsgleichung zu

$$\ddot{\varphi} = -\frac{g}{l} \sin \varphi . \quad (1.5)$$

Bei kleiner Auslenkung ist die Periode $T = 2\pi\sqrt{\frac{l}{g}}$ bei größeren Auslenkungen führt die nichtlinearisierte Pendelgleichung auf ein vollständiges elliptisches Integral der ersten Gattung. Die Differentialgleichungen (1.5) untersuchen wir im Folgenden numerisch (MATLAB-Programm `realesPendelDGL.m`). Das Ergebnis der numerischen Rechnung zeigt Abb. (1.7).

Die MATLAB-ode-Familie vermag nur Differentialgleichungssysteme 1. Ordnung zu lösen. Mittels

$$y(1) = \varphi \quad \text{und} \quad y(2) = \dot{\varphi} \quad (1.6)$$

wird Gl. (1.5) auf ein Differentialgleichungssystem 1. Ordnung transformiert und durch folgende MATLAB-Funktion repräsentiert:

```
function dy = realesPendelDGL(t,y,Vorf)
% Differentialgleichung
```

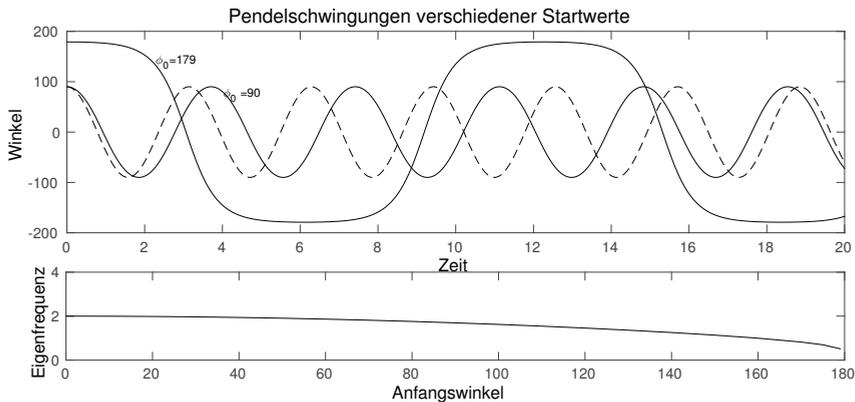


Abbildung 1.7: Schwingungsverläufe für das ebene Pendel für unterschiedliche Startwinkel. Oben ist die Schwingungsamplitude in Abhängigkeit von der Zeit für verschiedene Startwinkel aufgetragen. Die gestrichelte Linie zeigt die Näherung für das mathematische Pendel. Bei einem Startwinkel von 180° würde das Pendel auf dem Kopf stehen und keine Schwingung mehr ausführen. Unten sind die Eigenfrequenzen in Abhängigkeit des Startwinkels φ aufgetragen. Je höher der Startpunkt ist, umso kleiner wird die Eigenfrequenz, d.h. um so langsamer schwingt das Pendel.

```
dy(1) = y(2);           % y1 = phi; y2 = dphi/dt
dy(2) = -Vorf*sin(y(1)); % vorf=g/l
%
dy = dy.';
```

Aufgerufen wird diese Funktion durch das folgende Skript `realesPendel.m`.

```
%% Differentialgleichung loesen
y0 = [0.5*pi;0];           % Anfangswerte
tmax=20;                   % Maximale Simulationsdauer
Vorf = 4;                  % omega0^2 -> g/l
options = odeset('RelTol',1e-8,'AbsTol',1e-8);
[t,y] = ode45(@(t,y) realesPendelDGL(t,y,Vorf),[0,tmax],y0,options);
%% Visualisieren
x1=sin(y(:,1));
x2=-cos(y(:,1));
figure(1),plot(x1,x2),hold on, plot(x1(1),x2(1),'pr'),axis equal, shg
figure(2)
subplot(2,1,1)
plot(t,y(:,1)/pi*180),hold on
plot(t,y0(1)/pi*180*cos(sqrt(Vorf)*t),'m'),shg
```

„options“ ist eine Struktur, die die Defaultoptionen des Lösungsalgorithmus überschreibt. Im Beispiel haben wir die absolute und relativen Toleranzen neu gesetzt. Die anonymen

Funktion `@(t,y) realesPendelDGL(t,y,Vorf)` liefert ein Function Handle zurück, das sicherstellt, dass die Funktion, die das zu lösende Differentialgleichungssystem definiert vom Differentialgleichungssolver `ode...` auch gefunden wird. „(t,y)“ sind die Übergabeparameter mittels denen die Funktionen `ode45` und `realesPendelDGL` miteinander kommunizieren und „Vorf“ ein fester Parameter, der an die Funktion `realesPendelDGL` übergeben wird. Der Parameter „tmax“ legt die Integrationsdauer fest und „y0“ die Anfangswerte. Da es sich um ein Skript handelt, sind die Lösungen im Base Space (MATLAB Command Window) verfügbar.

Die „function-function“ `ode45` verfügt über eine optimierte Schrittweitesteuerung. Soll dagegen ein bestimmter Pendelausschlag berechnet werden benötigen wir zusätzlich eine Eventfunktion, die es erlaubt, numerisch exakt einen bestimmten Punkt anzusteuern. Die Eventfunktion wird über `odeset` festgelegt. Hier nutzen wir sie, um die Periode des realen Pendels zu bestimmen. Zur numerischen Bestimmung der Periode muss genau eine vollständige Schwingung erfasst werden. Hier die Fortsetzung des Skripts:

```
%% Bestimmen der Eigenfrequenzen
% die Eventfunktion sorgt fuer einen Abbruch der Berechnung nach genau
% einer Periode
winkel = [0.01 0.1 1, 10:10:170, 175, 179]; % Startwinkel
n=0;
for k=winkel
    n=n+1;
    y0 = [k/180*pi;0];
    options = odeset('RelTol',1e-8,'AbsTol',1e-8,...
        'event',@(t,y) realesPendeEvent(t,y,y0(1)));
    [t2,y2,te,ye,ie] = ode45(@(t,y) realesPendelDGL(t,y,Vorf),...
        [0,tmax],y0,options);
    omega(n) = 2*pi/(te(2)-te(1));
end
figure(2),
subplot(2,1,2)
plot(winkel,omega),shg
figure(2),subplot(2,1,1)
plot(t2,y2(:,1)/pi*180,'g')
```

Für das numerisch exakte Erreichen des Nulldurchgangs sorgt die Event-Funktion `realesPendeEvent.m`:

```
function [eventwert, isterminal, richt] = realesPendeEvent(t,y,awert)

eventwert = y(1);
isterminal = 0;
richt = -1;
```

„eventwert“ legt den Punkt (= 0) fest, der erreicht werden soll; „isterminal“ ob die Berechnung beendet werden soll (1) oder fortgesetzt (0) und „richt“ ob die Richtung

des Durchlaufens des Eventwertes beliebig ist (0), nur in positive (zunehmend +1) oder nur in negative (abnehmend -1) Richtung registriert werden soll.

Der allgemeine Aufruf einer Funktion wird durch das Schlüsselwort `function` eingeleitet. Die Rückgabewerte `[r1,...] = funktionsname(x1,x2,...)` stehen auf der linken Seite in eckiger `[r1,...]`, die Eingabewerte `(x1,...)` auf der rechten Seite in runder Klammer. Zusätzlich wird eine variable Anzahl von Variablen via `varargin` und `varargout` unterstützt. Die übergebene Anzahl wird in der vordefinierten Variablen `nargin` abgespeichert und in `nargout` die Anzahl der Rückgabewariablen, mit der die Funktion aufgerufen wird. Innerhalb einer Funktion sind `varargin` und `varargout` Zellvariablen.

1.2.4 4. Projekt: Polynome und Interpolationen

MATLAB ist eine numerische Programmiersprache – mit einer Ausnahme: den Polynomen, die auch symbolisch ausgewertet werden können. MATLAB beherbergt mehrere Funktionen, die als Argument ein Polynom erwarten. Dazu werden die Polynomkoeffizienten entsprechend der folgenden Zuordnung

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0 x^0 \leftrightarrow [a_n \ a_{n-1} \ \dots \ a_1 \ a_0]$$

einem Array zugewiesen. Betrachten wir als Beispiel die Funktion $p(x) = x^4 - 3x^3 + 7x$ und berechnen deren Nullstellen und Extrema.

```
>> % Das Polynom wird durch          >> % bestimmt:
>> px=[1 -3 0 7 0];                 >> dpxnull=roots(dpx)
>> % repraesentiert.                 dpxnull =
>> % Nullstellen                       1.5061 + 0.1661i
>> nullst=roots(px)                  1.5061 - 0.1661i
nullst =                               -0.7622

      0
      2.1395 + 0.9463i
      2.1395 - 0.9463i
     -1.2790

>> % Ableitung                       >> % zweite Ableitung
>> dpx=polyder(px)                   >> ddpx=polyder(dpx)
dpx =                                  ddpx =
      4      -9      0      7           12      -18      0

>> % Extrema sind durch die           >> % Minima oder Maxima ?
>> % Nullstellen der Ableitung        >> polyval(ddpx,dpxnull(3))
ans =
      20.6914
```

Die Funktion `polyfit` erlaubt einen Polynomfit an einen bestehenden Datensatz. Dabei ist die wahre Kunst, ein Polynom möglichst niedriger Ordnung zu finden, das den

Fit zufrieden stellend bewerkstelligt. Polynome hoher Ordnung führen häufig zu unerwünschten Oszillationen. Der Aufruf lautet `pfit = polyfit(x,y,n)`. Dabei bezeichnen „x“ und „y“ die zu fittenden Daten, „n“ die Polynomordnung und „pfit“ das Fit-Polynom.

Die Funktionen `interp1`, `interp2`, `interp3` und `interpN` erlauben Interpolationen an einen gegebenen Datensatz. Der Datensatz muss dabei der gewählten Dimension entsprechen. Ein Beispiel ist:

```
[X,Y,Z] = peaks(10);
subplot(1,2,1)
surf(X,Y,Z)
title('original')
% Interpolation
xi=-3:0.2:3;
yi=xi;
[Xi,Yi]=meshgrid(xi,yi);
Zi=interp2(X,Y,Z,Xi,Yi);
% Alternativ
%zi=interp2(X,Y,Z,xi,yi');
%figure,surf(xi,yi',zi)
subplot(1,2,2)
surf(Xi,Yi,Zi)
title('Interpolation')
```

Das Ergebnis zeigt Abb. (1.8).

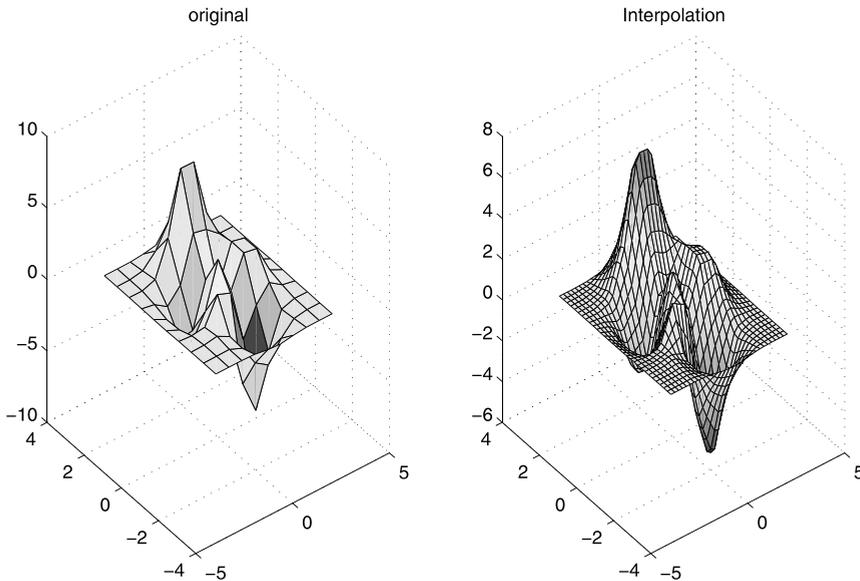


Abbildung 1.8: Beispiel zur 2-D-Interpolation. Beim Aufruf von `interp2` muss für die Interpolationspunkte entweder eine Matrix übergeben werden oder alternativ ein Zeilen- und ein Spaltenvektor.

1.2.5 5. Projekt: Datenanalyse, Laden und Speichern

In diesem Abschnitt sollen an einem Beispiel eine Fourieranalyse, Datenspeichern und -laden aufgezeigt werden.

Der Datenfile „panbsp.mat“ enthält die Orts- und Impulsvariablen eines nichtlinearen zweidimensionalen Systems - wie in der Praxis häufig üblich - in nicht-äquidistanten Zeitschritten. Die Daten können auch selbst mittels der ergänzenden Beispiele (zusätzliches Projekt 3) berechnet werden. Abb. (1.9) zeigt das Ergebnis der Fourieranalyse.

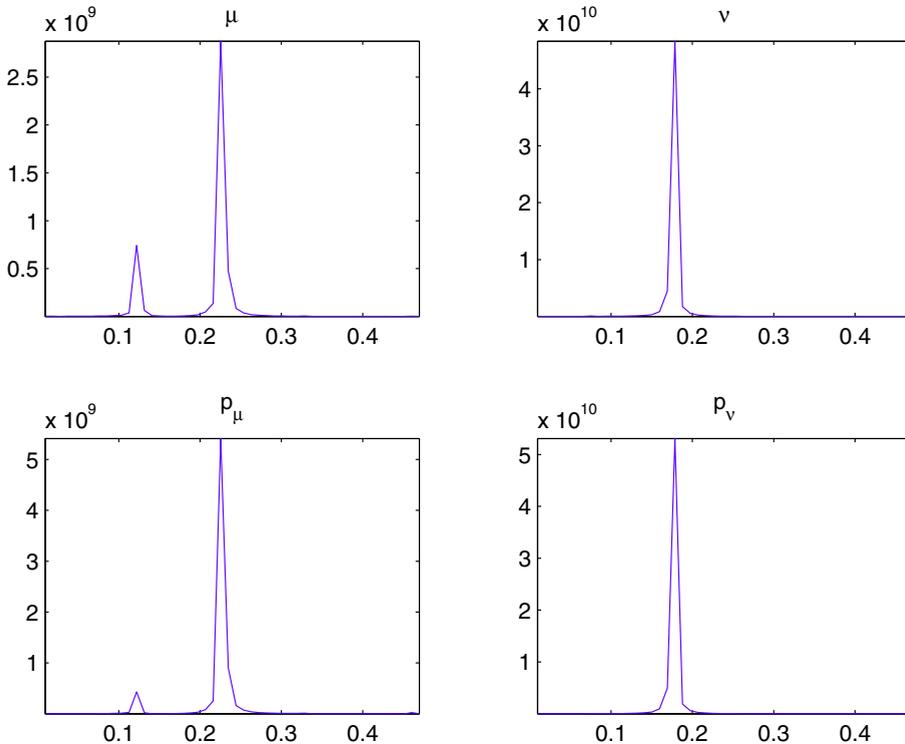


Abbildung 1.9: Fourieranalyse der Bewegung.

Mittels `load panbspdat.mat` werden die Daten geladen. $y(1) = \mu$ und $y(2) = \nu$ sind die Orts- und $y(3), y(4)$ die zugehörigen Impulskoordinaten. Die Auswertung liefert das folgende Skript:

```
ti=linspace(t(1),t(end),length(t));
% Da die urspruenglichen Daten nicht aequidistant in der Zeit sind
% wird in einem ersten Schritt ein gleich grosser aequidistanter
% Zeitvektor erzeugt.
for k=1:4

y1=interp1(t,y(:,k),ti);
% mittels Interpolation werden die zugehoerigen Daten berechnet.

nfour=2^floor(log2(length(t)));
% fft Berechnung ist optimal bei einer Laenge 2^n
```

```

yf1=fft(yi1,nfour);
% fft Berechnung und Powerspektrum
Pyf1=yf1.*conj(yf1);
% Berechnung des zugehoerigen Frequenzvektors
f=(1:nfour/2)/nfour/(ti(2)-ti(1));

% Plotten und Beschriften
% Nur die ersten 50 Werte sind interessant
subplot(2,2,k)
plot(f(1:50),Pyf1(1:50))
axis tight
switch k
    case 1
        title('\mu')
    case 2
        title('\nu')
    case 3
        title('p_\mu')
    case 4
        title('p_\nu')
end
% Bestimmen der beiden maximalen Fourierkomponenten
m1(k)=max(Pyf1(1:20))
m2(k)=max(Pyf1(1:100))
index1=find(m1(k)==Pyf1(1:100));
index2=find(m2(k)==Pyf1(1:100));
fm1(k)=f(index1)
fm2(k)=f(index2)
end

```

MAT-Files sind binäre, vom Betriebssystem unabhängige MATLAB-Daten-Files, die mit dem Kommando `save datname` erzeugt werden können und alle MATLAB-Datentypen sowie deren Variablenamen abspeichern. Mit `save fourierana m1 fm1` werden die Variablen „m1“, „f1“ in den File `fourierana.mat` abgespeichert. Mit `load fourierana` lassen sich alle Variablen wieder laden, mit `load fname v1 v2` nur die Variablen „v1“ und „v2“ aus dem MAT-File `fname`. In analoger Weise kann auch eine Auswahl von Variablen abgespeichert werden, `save fname v1 v2`. Mit `whos -file filename` kann der Inhalt des MAT-Files `filename` überprüft werden, Beispiel:

```

>> whos -file panbspdat.mat

```

Name	Size	Bytes	Class
epsilon	1x1	8	double array
h	1x4	32	double array
options	1x1	2752	struct array
t	492422x1	3939376	double array
tmax	1x1	8	double array

y	492422x4	15757504	double array
y0	1x4	32	double array
z	1x2	16	double array
zwi	1x1	8	double array

Grand total is 2462148 elements using 19699736 bytes

Die Operatorform `altvar=load('mylogfile')` bildet den Inhalt von „mylogfile“ auf die Struktur „altvar“ ab. Als Feldnamen dienen dabei die ursprünglichen Variablenamen. Dies ist insbesondere dann nützlich, wenn Variablen geladen werden sollen, deren Namen bereits vergeben sind.

1.3 Tipps zur Effizienzsteigerung

Geschwindigkeits- und Speicheroptimierung. Das größte Optimierungspotential liegt nicht im Computer oder in der Programmiersprache, sondern in der Art wie programmiert wird. In MATLAB ist es meist vorteilhaft zu vektorisieren, also nicht per Schleifenkonstrukt zu programmieren, sondern Matrizen, Vektoren und so fort in einem Programmierschritt zu erzeugen. Ausnahmen können Operationen mit extrem großen Variablen sein. Sollte Vektorisierung nicht gelingen bzw. ineffizient sein, dann sollte stets der notwendige Speicherbereich prealloziert werden, also beispielsweise durch eine Nullmatrix der erwarteten Größe. Einige Funktionen, die die Vektorisierung unterstützen, sind in Tabelle (1.1) aufgelistet.

Für numerische Daten ist die Standardeinstellung 8-Byte-Genauigkeit (double). Wird diese Genauigkeit nicht benötigt, kann mittels Typkonvertierung auch 4-Byte-Genauigkeit (single) gewählt werden oder sogar auf einen der ganzzahligen Datentypen `int(.)` und `uint(.)` ausgewichen werden. Dies vermindert nicht nur den benötigten Speicherbedarf deutlich, sondern kann auch die Abarbeitung beschleunigen. Allerdings ist bei Integer-Datentypen zu beachten, dass nur die elementaren Rechenoperationen unterstützt werden. Ähnliche Überlegungen gelten auch für Zellvariablen und Strukturen. Hier sollte der Speicherplatz mit dem Befehl `A = cell(m,n)` bzw. `A = struct(...)` vorbelegt werden.

MATLAB bietet unterschiedliche Formen der Indizierung von Arrays. Die lineare Indizierung ist effizienter als die Zeilen-Spalten Indizierung; Logische Indizierung erlaubt, Elemente nach spezifischen Kriterien auszusieben. Liegt ein hoher Anteil an verschwindenden Matrixelementen vor, dann sollte der Typ `sparse` verwendet werden. Dabei ist allerdings zu beachten, dass Matrixoperationen, beispielsweise mehrfachen Multiplikationen, unter Umständen zu vollen Matrizen und damit zu einem erhöhten Speicherverbrauch führen.

MATLAB ist eine Interpretersprache. Skripte werden zeilenweise Funktionen in ihrer Gesamtheit interpretiert. MATLAB-Funktionen sind effizienter als Skripte. Zudem bieten sie eine höhere Sicherheit, da Variablen in MATLAB lokal sind und Funktionsvariablen im eigenen Function Space abgespeichert werden.

Zum Speichern und Laden von Variablen eignen sich besonders die MATLAB-Routinen

Tabelle 1.1: Einige MATLAB-Funktionen zur Optimierung.

Funktion	Kurzbeschreibung
all, any	Test auf verschwindende Matrixelemente
cumsum	kumulative Summe
cumprod	kumulatives Produkt
diff	Differenzen und approximative Ableitungen
find	Indeksuche
ind2sub	Lineare Indizes sind effizienter
sub2ind	ind2sub und sub2ind: Indexkonvertierung
ipermute	inverse Permutation
permute	gewöhnliche Permutation
logical	logische Variablen erzeugen
meshgrid	vgl. Grafik, aber auch statt Doppelsummen
ndgrid	Arrayerzeugung
prod	Produktbildung
repmat	Matrizenreproduktion
reshape	Matrizenumbildung
shiftdim	Dimensionsverschiebung
sort	Sortieren in aufsteigender Reihe
sparse	Dünn besetzte Matrizen (Umkehrung full)
squeeze	Dimensionsreduktion
sum	Summenbildung

`load` und `save`, da sie rascher auf Variablen zugreifen als die anderen Input/Output-Routinen. Darüber hinaus haben sie den Vorteil, dass MAT-Files bei allen von MATLAB unterstützten Betriebssystemen unter MATLAB eingelesen werden können.

Die vereinfachte `switch`-Anweisung wird rascher abgearbeitet als `if -- elseif -- else`-Abfragen. `try catch`-Umgebungen sind ebenfalls zeiteffizienter als Variablenprüfungen vom Typ `nargchk`, `nargoutchk` und `isa` bzw. `class`. Allerdings sind solche Prüfungen im Regelfall nicht sehr umfangreich.

Daten werden bei PCs in einem virtuellen Adressraum abgespeichert. Für eine Variable muss dabei ein zusammenhängender Speicherbereich zur Verfügung stehen. Out-of-Memory-Fehler lassen sich durch die Verwendung kleiner Variablen statt großer verhindern. Insbesondere Struktur- und Zellvariablen benötigen mehr Speicherplatz als ihr „Nettoinhalt“. Auf den Vorteil der Preallozierung wurde bereits oben hingewiesen. Zusätzlich kann die Verwendung von `pack` die Speichernutzung optimieren. Gegebenenfalls kann auch das Starten von MATLAB mit der Option `-nojvm` (no java virtual machine) zu einem größeren nutzbaren Speicherbereich führen. Dies gilt insbesondere für UNIX-Rechner. Die Größe des zur Verfügung stehenden Speicherbereichs wird auch durch den „Java Heap Space“ beeinflusst. Dessen Größe lässt sich mittels der Präferenzen unter `Preferences` ⇒ `General` ⇒ `Java Heap Space` einstellen.

MATLAB erlaubt die Verwendung von Multistatement-Lines, also mehrere Befehle durch Komma getrennt in einer Zeile. Mehrfachbefehle sollten im Regelfall vermieden wer-

den. Zum Einen wird dadurch die Lesbarkeit eines Programms erschwert, zum Anderen gibt es einige Funktionalitäten, die zeilenorientiert sind, beispielsweise der Debugger. Breakpoints können nur zu Beginn einer Zeile gesetzt werden. Folglich schränken Multistatement-Lines die Debug-Möglichkeiten ein. Ähnliche Einschränkungen gelten auch für das Code Coverage Tool, mit dem der Anteil der ausgeführten Zeilen (nicht Befehle!) getestet wird.

Ausgabe professionell gestalten. MATLAB bietet mit `disp` eine einfache Möglichkeit, Ergebnisse formatiert auszugeben. Ein professionelleres Bild gewinnt man mit `fprintf` und `sprintf` (statt `num2str` oder `int2str`). Eine detaillierte Diskussion findet sich in den Kapiteln 9.5 und 20.

Variablenübergabe. Variablen in MATLAB-Funktionen sind stets lokal, sofern sie nicht global definiert sind. Dies hat zur Folge, dass unter Umständen viele Variablen übergeben werden müssen. Nested Functions sind in den Speicherbereich der übergeordneten Funktion eingebettet und kennen deren Variablen ebenfalls. Sollen umfangreiche Daten in mehreren Teilprogrammen genutzt werden, dann bieten Nested Functions eine bequeme Möglichkeit, auf diese Variablen direkt und ohne die Notwendigkeit der globalen Deklaration zuzugreifen. Nested Functions kennen alle Variablen der übergeordneten Funktion, die zum Zeitpunkt der Deklaration der Nested Function existieren (vgl. Kap. 9.4). Die Verwendung globaler Variablen sollte prinzipiell vermieden werden.

Indexgymnastik. In vielen Fällen erwartet eine Funktion einen Spaltenvektor. Die Zuordnung `x = x(:)` stellt sicher, dass ein Spaltenvektor vorliegt. MATLAB-Funktionen, die Matrizen spaltenweise auswerten, können mittels „`:`“ in einem Schritt die gesamte Matrix auswerten. Beispiel:

```
>> A=magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> max(A)           % berechnet spaltenweise die Maxima
ans =
    16    14    15    13

>> max(max(A))     % Das absolute Maximum
ans =
    16

>> max(A(:))       % Einfacher mit dem :-Operator
ans =
    16
```

Der „`:`“-Operator erlaubt auch die Form einer Matrix beizubehalten. Beispiel:

```
>> A=rand(3,4)
```

```
A =
    0.9501    0.4860    0.4565    0.4447
    0.2311    0.8913    0.0185    0.6154
    0.6068    0.7621    0.8214    0.7919
```

```
>> b=A; % wie A, aber fortlaufende Werte
>> b(:)=1:numel(b)
b =
     1     4     7    10
     2     5     8    11
     3     6     9    12
```

Damit erspart man sich die ineffizientere Verwendung von `reshape`.

Sollen Zeilen oder Spalten repliziert werden, so bietet sich dafür `repmat` an, dessen Effizienz deutlich optimiert wurde. Beispiel:

```
>> b
b =
     1     4     7    10
     2     5     8    11
     3     6     9    12
>> B=repmat(b,8,3);
```

```
>> whos
  Name      Size      Bytes  Class

  b         3x4         96  double array
  B        24x12       2304  double array
```

Endliche Genauigkeit beachten. Die maximale Genauigkeit unter MATLAB ist durch die 8-Byte-Darstellung der Zahlen gegeben. Dies bedeutet, dass nur solche Zahlen exakt wiedergegeben werden können, die eine exakte Bit-Darstellung erlauben wie etwa ganze Zahlen oder durch $1/2, 1/4, \dots$ darstellbare rationale Zahlen. Beispielsweise ergibt wegen der endlichen Auflösung $(0.7 + 0.6 - 0.3)$ nicht exakt 1:

```
>> 1 == (0.7 + 0.6 - 0.3)      ans =
                                0
```

Statt identischer Abfragen sollen daher stets Ungleichungen gewählt werden, also im Beispiel

```
>> abs(1 - (0.7 + 0.6 - 0.3)) < eps(2)
ans =
     1
```

wobei die Schranke durch eine sinnvolle Grenze gegeben sein muss. `eps(x)` liefert die 8-Byte-Genauigkeit der Zahl x . Da bei Vergleichen mehrere Zahlen im Spiel sind, darf diese Grenze nicht zu eng gesteckt werden. Bit-bedingte Abweichungen können sich ja auch aufaddieren.

Strings. In Kapitel 6 werden die Stringfunktionen diskutiert. Mehrere Funktionen können vergleichbare Aufgaben übernehmen. So ist beispielsweise für das Zusammenpacken mehrerer String-Variablen `[\dots]` effizienter als `strcat`. Je nach Anwendung sind kategoriale Variablen, `categorical`, effizienter als String-basierte Ausdrücke (vgl. Kap. 22.4). Dies betrifft insbesondere Untersuchungen diskreter Merkmale bei textbasierten Daten.

Für jede Art von Pattern Matching sollte `regexp` benutzt werden, das eine Fülle sehr effizienter Möglichkeiten bietet. `strcmp` vergleicht Strings effizienter als `isequal`. Der Code wird rascher abgearbeitet und ist auch besser verständlich. Falls zwischen Groß- und Kleinschreibung beim Vergleich nicht unterschieden wird, ist `strcmpi` günstiger als `strcmp`.

Mit seinen vielen Funktionalitäten bietet MATLAB häufig mehrere Möglichkeiten, ein Problem zu lösen. Bei den Alternativen sollte neben der Geschwindigkeit auch die Lesbarkeit des Programms im Vordergrund stehen. Im Zweifelsfall lässt sich die Effizienz durch ein kleines Testprogramm, in dem die Alternativen mehrfach ausgeführt werden, und durch Messen der Abarbeitungszeit mit den Befehlen `tic` und `toc` testen. Um gleiche Bedingungen zu gewährleisten, sollte bei Arrays stets Speicher prealloziert und andere Anwendungen abgeschaltet werden, sonst kann das Ergebnis leicht verfälscht werden und zu falschen Schlüssen führen.

1.4 Tabellarische Übersicht ausgewählter MATLAB-Kommandos

In den folgenden Tabellen sind einige häufig genutzte MATLAB-Befehle aufgelistet. Die Übersicht ist unvollständig und soll nur einen raschen Überblick leisten.

Tabelle 1.2: *Matrix- und Arrayoperationen.*

Matrix	Array .-Operator	Kurzbeschreibung	Erläuterung an einem Beispiel
<code>+, -</code>	<code>+, -</code>	Addition und Subtraktion	
<code>*</code>	<code>.*</code>	Multiplikation	$(A * B)_{m,n} = \sum_k A_{m,k} * B_{k,n}$
<code>^</code>	<code>.^</code>	Potenzieren	
<code>\</code>	<code>.\</code>	Linksinverse	aber
<code>/</code>	<code>./</code>	Rechtsinverse	
<code>'</code>	<code>.'</code>	Transponieren	$(A .* B)_{m,n} = A_{m,n} * B_{m,n}$

Tabelle 1.3: Allgemeine MATLAB-Befehle.

Funktion	Kurzbeschreibung
clear	Löschen von Variablen
clc	Löschen des Command Windows, Variablen bleiben unberührt
dir	Auflisten des Verzeichnisses
doc	Aufruf der Dokumentation, Beispiel: doc plotyy
edit	Aufruf des Editors
help	Hilfeaufruf, Beispiel: help plotyy
load	Laden von Variablen
save	Speichern der Variablen
which	Lokalisierung von Variablen und Files
whos	Übersicht der Variablen
;	Unterdrücken der Bildschirmausgabe
[]	Erzeugen von Arrays
{ }	Erzeugen von Zellvariablen

Tabelle 1.4: Polynome.

Funktion	Kurzbeschreibung	Beispiel
conv	Polynommultiplikation	$p(x) = 3x^2 - 5x + 7$
deconv	Polynomdivision, Dekonvolution	in MATLAB
poly	Charakteristisches Polynom	$\text{px} = [3 \ -5 \ 7]$
polyfit	Polynomfit	$\text{x} = -2:0.1:2$
polyval	Polynomauswertung	$\text{y} = \text{polyval}(\text{px}, \text{x})$
roots	Nullstellen eines Polynoms	$\text{null} = \text{roots}(\text{px})$

Tabelle 1.5: Datenanalyse.

Funktion	Kurzbeschreibung	Funktion	Kurzbeschreibung
max	Maximum bestimmen	min	Minimum bestimmen
mean	Mittelwert	median	Median berechnen
sort	Daten aufsteigend sortieren	std	Standardabweichung
sum	Summe bilden	prod	Produkt berechnen
diff	Differenz, numerische Ableitung	gradient	Numerischer Gradient
corrcoef	Korrelationskoeffizienten	cov	Kovarianzmatrix
abs	Absolutwert	angle	Phasenwinkel
fft	1-D-Fast-Fouriertransformation	ifft	inverse FFT
fft2	2-D-Fast-Fouriertransformation	ifft2	inverse 2-D-FFT
fftn	n-D-Fast-Fouriertransformation	ifftn	inverse n-D-FFT
timeseries	Zeitreihenobjekt erzeugen	tstool	Zeitreihen-Tool öffnen

Tabelle 1.6: Matrixfunktionen.

Funktion	Kurzbeschreibung	Aufruf: Matrix A
chol	Choleski-Zerlegung	<code>R = chol(A)</code>
det	Determinante einer Matrix	<code>Adet = det(A)</code>
diag	Auslesen des Diagonalteils oder Erzeugen einer Diagonalmatrix	<code>Adiag = diag(A)</code> <code>neu = diag(Adiag)</code>
eig	Eigenwerte berechnen	<code>[a,b] = eig(A)</code>
eigs	Eigenwerte dünn besetzter Matrizen	
full	Matrix aus dünn besetzter Matrix	<code>A = full(As)</code>
inv	Inverse berechnen	<code>Ainv = inv(A)</code>
lu	LU-Zerlegung	statt <code>inv</code> besser \
norm	Vektor- und Matrixnorm	<code>an = norm(A,p)</code>
qr	QR-Zerlegung	
rank	Rang einer Matrix	
reshape	Umordnen einer Matrix	<code>B = reshape(m,n,A)</code>
schur	Schurzerlegung	
sparse	dünn besetzte Matrix erzeugen	<code>As = sparse(A)</code>
svd	Singulärwertzerlegung	

Tabelle 1.7: Differentialgleichungen.

Funktion	Kurzbeschreibung
ode45, ode23, ode113	Anfangswertprobleme: Systeme gewöhnlicher Differentialgleichungen
ode15s, ode23s	steife Differentialgleichungen
ode23t, ode23tb	moderat steife Differentialgleichungen
ode15s, ode23t	Index 1 differential-algebraische Gleichungen
dde23	gewöhnliche Differentialgleichungen mit Verzögerung
bvp4c, bvp5c	Randwertprobleme gewöhnlicher Differentialgleichungen
pdepe	partielle Differentialgleichungen

Tabelle 1.8: Integration und Interpolation.

Funktion	Kurzbeschreibung	Beispiel
quad	Integration (Simpson-Verfahren)	<code>w = quad(@fun,a,b)</code>
quadl	Integration (Lobatto-Verfahren)	Integrationsgrenzen: a, b
quadv	vektorierte Integration	@fun: Function Handle
integral	adaptive Integration	<code>w = integral(@fun,a,b)</code>
dblquad, integral2	2-D-Integration	
quad2d	planare 2-D-Integration	
triplequad, integral3	3-D-Integration	
interp1	1-D-Interpolation	<code>yi = interp1(x,y,xi)</code>
interp2	2-D-Interpolation	x,y Datensatz
interp3	3-D-Interpolation	yi Interpolationswerte an den
interp	n-dimensionale Interpolation	Stützstellen xi
griddedInterpolant	n-dimensionale Interpolation	
interpft	Fourier-basierte Interpolation	
pchip	kubisch-hermitesche Interpolation	<code>yi = pchip(x,y,xi)</code>
spline	Spline-Interpolation	

Tabelle 1.9: Daten Ein- und Ausgabe.

Funktion	Kurzbeschreibung	Funktion	Kurzbeschreibung
dlmread	ASCII-Files lesen	dlmwrite	ASCII-Files schreiben
fopen	Daten-File öffnen	fprintf	Textdatei schreiben
fread	Binäre Datei lesen	fscanf	Textdatei lesen
fwrite	Binäre Datei schreiben	importdata	Daten einladen
imread	Bilddateien lesen	imwrite	Bilddateien schreiben
load	MAT-File laden	save	MAT-File schreiben
textread	Formatierte Daten lesen	xlsread	Exceldateien lesen

Tabelle 1.10: Plot-Routinen.

Funktion	Kurzbeschreibung	Beispiel
plot	2-D-Linienplot	<code>plot(x,y,'mp:')</code>
plotyy	2-D-Linienplot, 2 y-Achsen	x,y Datensatz
polar	Polarkoordinatenplot	'mp:', Farbe, Datenpunkte, Linie
semilogx	halblogarithmischer Plot (x-Achse)	
semilogy	halblogarithmischer Plot (y-Achse)	
loglog	doppeltlogarithmischer Plot	
ezplot	Funktionsplot	<code>ezplot(@fn,-2,2)</code>
ezpolar	Funktions-Polarplot	plottet $f_n = 0$ von $-2 \dots 2$
hist	Histogramm	<code>y=rand(10000,1), hist(y,25)</code>
bar	Balkenplot	
stairs	Treppenplot	
pie	Kuchenplot	
plot3	3-D-Linienplot	<code>plot3(x,y,z,'mp:')</code>
mesh	3-D-Gitterlinien-Flächenplot	$x=-2:0.1:2, y=x$
surf	3-D-Flächenplot	<code>[X,Y]=meshgrid(x,y)</code>
surfc	3-D-Flächen-Konturplot	$Z = X.^2 Y.^2$
meshgrid	Erzeugen der 2-D-Arrays	<code>mesh(X,Y,Z), surf(X,Y,Z)</code>

Tabelle 1.11: Plot-Hilfsfunktionen.

Funktion	Kurzbeschreibung	Funktion	Kurzbeschreibung
title	Titel erstellen	legend	Legende hinzufügen
grid	Gitterlinien	text	Text einfügen
gtext	mauspositionierter Text	hold	Abbildung halten
x-,y-,zlabel	Achsen beschriften	subplot	Mehrfachabbildungen

2 Grafische Utilities

2.1 Übersicht

Dieser Abschnitt dient einer Übersicht über die MATLAB-Oberfläche (Desktop) und ihrer Grundelemente Command Window, Command History, Workspace Browser und Current Folder. Gegenstand sind außerdem wichtige grafische Benutzeroberflächen wie Help Browser, Variable Editor, Editor und Debugger, Code Analyzer und den Profiler, das interaktive Einlesen von Daten sowie die Plot- und Ausgabe-Tools beispielsweise für HTML-, \LaTeX - oder Worddokumente.

Eine rasche Einführung in die verschiedenen Funktionalitäten bieten die Video Tutorials. Über „Help“, s. Abb. (2.7), „Examples“ springen Sie direkt in die in der MATLAB-Dokumentation aufgelisteten Beispiele. Unter “Getting Started“ finden sich mehrere nützliche Videoclips. In den älteren Versionen geht der Weg über den Help Browser und Demos. Hier eine knappe Übersicht der Eigenschaften des MATLAB-Desktops und der interaktiven Möglichkeiten, beispielsweise Daten einzulesen, Programme zu debuggen und Diagramme zu erstellen.

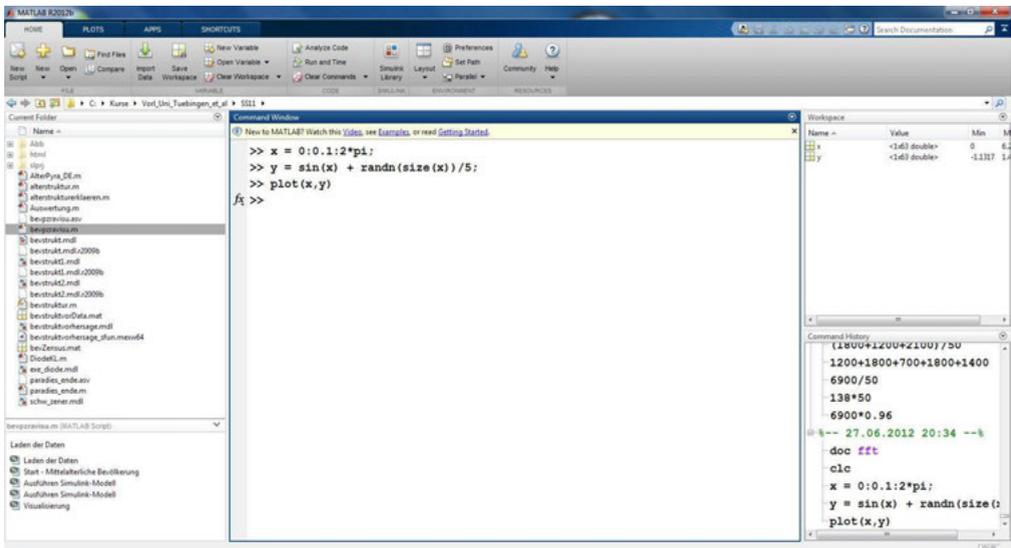


Abbildung 2.1: Die MATLAB-Oberfläche ab dem Rel. 8.0

2.1.1 Der MATLAB Desktop

Nach einem Doppelklick auf das MATLAB Icon oder dem Aufruf aus einer Shell unter UNIX/Linux öffnet sich die MATLAB-Oberfläche, auch als MATLAB Desktop bezeichnet, die eine integrierte Entwicklungsumgebung darstellt. Mit dem Rel. 2012b hat sich MATLAB ein neues und moderneres Kleid gegeben, s. Abb. (2.1), mit der Werkzeugleiste (Toolstrip), Abb. (2.2), am oberen Rand, die mittels Registerkarten (Tabs) nach verschiedenen Anwendungen strukturiert ist. Jede Registerkarte ist in einzelne Sektionen (Section) unterschiedlicher Funktionalitätsgruppen eingeteilt, beispielsweise „PLOTS“ in die Sektionen „SELECTION“, „PLOTS“ und „OPTIONS“. Die Anzahl der Sektionen, Abb. (2.2), hängt von den installierten Produkten ab und welche Registerkarte ausgewählt wurde.

Beginnen wir mit der Registerkarte „HOME“, s. Abb. (2.1). Von links nach rechts ist an erster Stelle der File-Bereich, der es erlaubt Dateien zu öffnen, zu erstellen und miteinander zu vergleichen. Unter „New“ lassen sich per Knopfdruck verschiedene Templates im Editor erzeugen sowie der Shortcut-Editor, die Figure-Umgebung und der Guide öffnen. Die Sektion „VARIABLE“ stellt Funktionalitäten zum Importieren (s. Abschnitt 2.5 Interaktiver Daten Import), Erstellen, Speichern, Löschen und Öffnen (vgl. 2.1.4 Va-

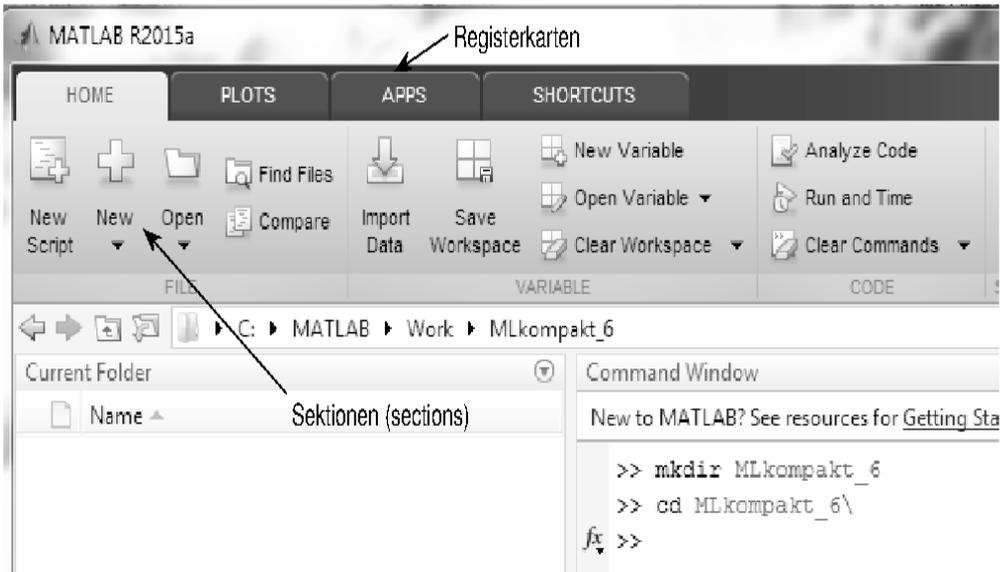


Abbildung 2.2: Die Werkzeugleiste der MATLAB-Oberfläche ist u.a. mittels Registerkarten nach verschiedenen Anwendungen strukturiert, die wiederum in einzelne Sektionen unterschiedlicher Funktionsgruppen unterteilt sind.

riable Editor) von Daten zur Verfügung. Der Bereich „CODE“ dient dem Analysieren von Programmen mit dem Code Analyzer „Analyze Code“, dem Testen auf Effizienz mit dem Profiler „Run and Time“ sowie dem Löschen einzelner Befehle des Command Windows oder der Command History. Die Möglichkeiten MATLAB-Code zu analysieren und die Effizienz zu testen, werden im Abschnitt 2.3 vorgestellt. Den Simulink-Bereich überspringen wir. Die Sektion „ENVIRONMENT“ erlaubt das aktuelle Layout der MATLAB-Oberfläche abzuspeichern, bereits existierende Layouts zu öffnen und die einzelne Grundelemente der MATLAB-Entwicklungsumgebung ab- bzw. auszuwählen. Sollten Sie die Werkzeugleiste abgewählt haben, so lässt sie sich mit dem Pfeilsymbol rechts oben wieder dazu packen. Die grafischen Benutzeroberflächen zur Festlegung der Präferenzen, beispielsweise Schriftgröße, numerisches Format, usf. und dem Aufbau des MATLAB-Pfads können ebenfalls per Mausklick in der Sektion „ENVIRONMENT“ geöffnet werden. „Add-Ons“ unterstützt beispielsweise das Erstellen von Apps, Toolboxes und das Herunterladen von Hardware-Support-Packages. Hardware-Support-Packages dienen dem Einbinden von Arduino Boards, Raspberry Pis und anderen Hardwareprodukten in MATLAB und Simulink. Der letzte Bereich „RESOURCES“, erlaubt per Mausklick die Hilfe aufzurufen, s. Abb. (2.7) und auf Internetseiten zuzugreifen.

Wo und wie kann man das Erscheinungsbild des Desktop anpassen?

Die einzelnen Elemente des Desktops lassen sich beliebig anordnen. Soll beispielsweise die Command History neben das Command Window verschoben werden, so wird die Kopfleiste der Command History einfach mit der linken Maustaste festgehalten und an den entsprechende Ort verschoben. Unter dem Menüpunkt „Layout“ lässt sich dieses neue Erscheinungsbild des Desktops unter einem geeigneten Namen abspeichern und später wieder laden, Abb. (2.4). Außerdem lassen sich hier auch die einzelnen Elemente an- und abwählen. Eine weitere Gestaltungsmöglichkeit bieten die Pfeiltaste der einzelnen Desktop-Elemente. Durch anklicken öffnet sich ein Dialogfenster, das es beispielsweise erlaubt das jeweilige Teilfenster ab- bzw. anzudocken. Diese Dock-Möglichkeit besteht

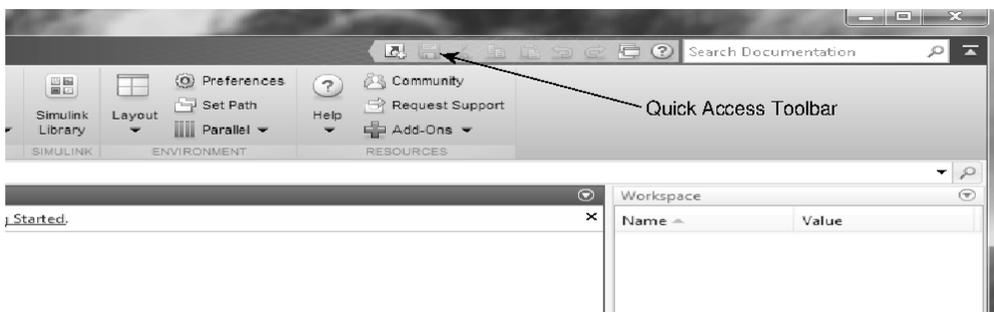


Abbildung 2.3: Durch Rechtsklick auf das Symbol einer Anwendung, beispielsweise einer App kann diese der „Quick Access Toolbar“ hinzugefügt werden. Rechts oben befindet sich „Search Documentation“, das erlaubt die Dokumentation nach Schlagwörtern zu durchsuchen und durch Klicken auf den Treffer, direkt an die entsprechende Stelle in der Dokumentation zu springen.

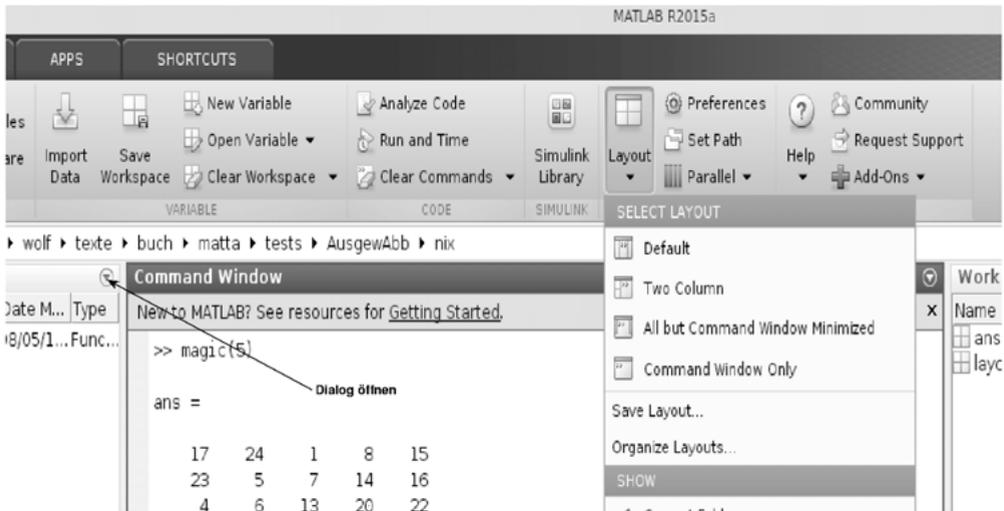


Abbildung 2.4: Unter „HOME“ befindet sich das Menü Layout mit dem sich selbsterstellte Desktop-Layouts speichern und laden lassen. Mit der Pfeiltaste lässt sich in jeder Desktop-Komponente ein Dialogfenster öffnen mit dem sich beispielsweise diese Komponente ausdocken lässt.

auch für eigene Abbildungen (Figure), Editor und Variable Editor. Beim Schließen von MATLAB wird das letzte Layout abgespeichert und beim nächsten Starten automatisch verwandt.

Seit dem MATLAB-Rel. 7.9 werden eigene Tastaturkürzel (keyboard shortcuts) unterstützt. Unter Preferences → Keyboard → Shortcuts können unter „Action Name“ verschiedene Aktionen ausgewählt werden und über das +-Symbol mit einem Tastaturkürzel verknüpft werden. Im darüber liegenden Fenster wird über das Tastatursymbol die Zahl der notwendigen Tastendrücke ausgewählt und die entsprechende Wunschkombination eingegeben. Konflikte werden gegebenenfalls im darunter befindlichen Fenster „All possible conflicts“ angezeigt.

Mit dem MATLAB-Rel. 7.9 wurde unter dem Menüpunkt „Desktop“ der neue Unterpunkt „File Exchange“ eingeführt. „File Exchange“ öffnet ein Dialogfenster mit dem sich direkt auf die File Exchange-Internetseite zugreifen und nach MATLAB-Funktionen suchen lässt. Unter dem Schlagwort „File Exchange“ verbirgt sich eine Internetseite auf der jeder MATLAB-Nutzer eigene Funktionalitäten für die MATLAB-Gemeinschaft zur Verfügung stellen kann. Das Rel. 8.0 führt nun in der Toolbar über „Community“ auf eine MathWorks-Internetseite, auf der sich zusätzlich Links zu weiteren kommunikativen Internetseiten befinden.

Die Plotmöglichkeiten der Registerkarte „PLOTS“ werden über den Workspace Browser und Variable Editor genutzt.

Shortcuts. Die vierte Registerkarte, Abb. (2.2), ist den Shortcuts gewidmet. Shortcuts sind knappe MATLAB-Anwendungen, die häufig ausgeführt werden und daher bequem mittels Mausklick angestoßen werden sollen. Ein typisches Beispiel ist die Befehlsabfolge `clear`, `close all`, `clc`, um alle Variablen zu löschen (`clear`), alle Figure-Windows mit eigenen Abbildungen zu schließen (`close all`) und alle Einträge im Command Window zu löschen (`clc`).

Mit „New Shortcut“ öffnet sich der Shortcut-Editor. Unter „Label“ wird ein geeigneter Namen gewählt unter dem das Shortcut verwaltet wird und unter Icon ein entsprechendes Symbol. Shortcuts sind kleine Skripte. Sie haben wie MATLAB-Skripte keinen eigenen Speicherbereich und speichern ihre Variablen im Basespace ab. Häufig werden Shortcuts über die Command History erzeugt.

Das Command Window. Das Command Window dient dem Aufruf von MATLAB-Befehlen, Erzeugen von Variablen und dem Anzeigen von Ergebnissen. Die Eingabe erfolgt hinter dem MATLAB-Doppelprompt `>>`, s. Abb. (2.5). Befehle müssen nicht vollständig ausgeschrieben werden, die Tab-Taste vervollständigt bei Eindeutigkeit einen Befehl oder gibt eine Liste in einem kleinen separaten Fenster aus. Während dem Schreiben von Befehlen geht ein kleines Hilfenfenster mit Hinweisen zu den möglichen Funktionsargumenten auf. Eine nützliche Neuerung sind die Korrekturvorschläge bei Tippfehlern. Zu früheren Eingaben kann mit den Cursortasten zurückgeblättert werden. Ist bereits ein Teil eines Kommandos eingegeben, springt MATLAB mit der Cursortaste direkt zu dem letzten Befehl, der mit derselben Zeichenfolge startete. Erzeugte Variablen werden im Workspace Window aufgelistet, Befehle in der Command History. Der Befehl `commandwindow` öffnet das MATLAB Command Window bzw. bringt es in den Vordergrund. Mit `get(0,'CommandWindowSize')` wird die aktuelle Zahl der Spalten und Zeilen ausgegeben, die von der momentanen Größe abhängt.

Klickt man auf das vor dem Doppelprompt stehende Symbol „*f*“ öffnet sich der Function Browser, mit einem Suchfeld zur Eingabe von Suchbegriffen und einem Inhaltsverzeichnis zum Durchblättern. Der Function Browser kann auch mit der Tastenkombination „Umschalttaste, F1“ geöffnet werden und dient der raschen Hilfe. Per Mausklick auf den Suchbegriff öffnet sich ein Hilfenfenster mit einer kurzen Beschreibung, die vollständige Dokumentation der ausgewählten Funktion kann aus dem Hilfenfenster durch Klick auf den Link „More Help“ geöffnet werden.

Command History. Hier werden die im Command Window eingegebenen Befehle zur Wiederverwendung nach dem Datum geordnet abgespeichert. Durch Doppelklick können diese Befehle direkt ausgeführt werden und mit der linken Maustaste in das Command Window zur erneuten Bearbeitung verschoben werden. Die linke Maustaste gemeinsam mit der Shift- oder der Steuerungstaste erlaubt die Auswahl von Gruppen der im History Window gespeicherten Kommandos. Mit der rechten Maustaste öffnet sich ein Fenster, das das Kopieren der ausgewählten Befehle, das direkte Erzeugen von M-Files und Shortcuts oder das Löschen aus der Command History erlaubt. Eigene Shortcuts lassen sich auch durch einfaches Verschieben ausgewählter Befehlsfolgen auf die Shortcut-Leiste erzeugen. `>> commandhistory` öffnet die Command History bzw. bringt sie in den Vordergrund.

Der Workspace Browser. Der Workspace Browser zeigt die aktuellen Variablen des Base-Speicherbereichs (Command Window) oder beim Debuggen die des zugehörigen Funktionsspeicherraums an. Die Variablen sind alphabetisch geordnet, können aber auch durch Anklicken der Spaltenüberschriften nach den damit verknüpften Eigenschaften umgeordnet werden. Anklicken der Pfeiltaste öffnet ein selbsterklärendes Fenster mit dem sich beispielsweise alle Variablen abspeichern lassen.

Durch Markieren einer oder mehrerer Variablen mit Hilfe der Maus- und Umschalt- oder Steuertaste werden unter der Registerkarte „PLOTS“ für das Plotten geeignete Graphen aktiv, die per Mausklick ausgeführt werden können. In der Sektion „Selection“

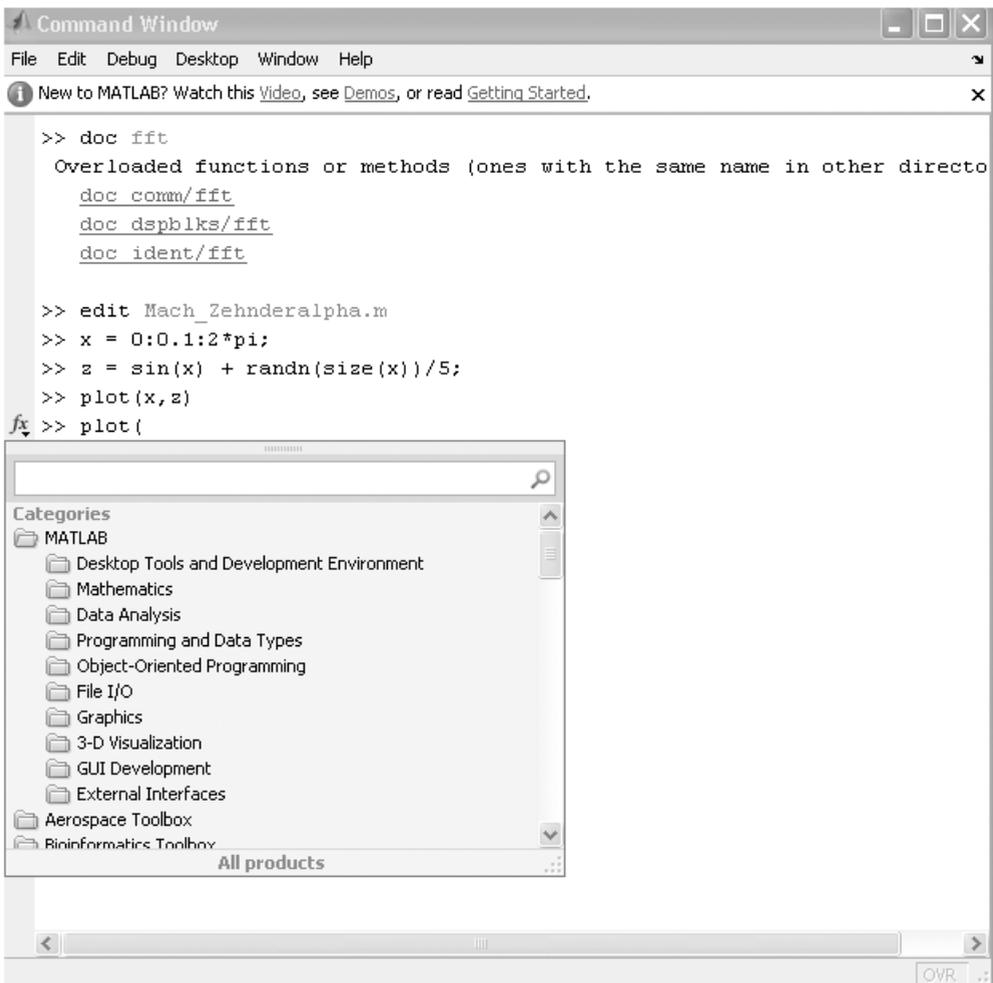


Abbildung 2.5: Das Command-Window (hier das Rel. 7.9) hat sich in den letzten Jahren nur unwesentlich verändert.

sind die ausgewählten Variablen aufgelistet. MATLAB erkennt dabei automatisch, ob der Variablentyp zum Plotten geeignet ist oder nicht. Alternativ kann ein Dialogfenster durch Anklicken der Variablen im Workspace Browser mit der rechten Maustaste geöffnet werden. Mit der linken Maustaste können Variablen in das Command Window beispielsweise als Argument einer Funktion verschoben werden. Durch Doppelklick auf eine Variable öffnet sich der Variable Editor.

Current Folder. (Veraltet Current Directory Browser.) Verzeichnisse werden in einer Baumstruktur aufgelistet und Funktionen, Skripte und Klassen durch unterschiedliche Icons grafisch hervorgehoben. Mittels Doppelklick kann direkt in Unterverzeichnisse gesprungen werden. Die erste Hilfezeile ausführbarer MATLAB-Dateien wird als Informationszeile im darunterliegenden Fenster ausgegeben. Dort finden sich auch Informationen zu anderen Dateien wie Inhaltsangaben zu Mat-Files. Durch Doppelklick auf eine Variable wird diese Variable aus der Mat-Datei geladen.

Klicken auf die Pfeiltaste öffnet ein Dialogfenster, s. Abb. (2.6), das erlaubt Dateien zu vergleichen (Compare) und verschiedene Reports zu erstellen. Der Code Analyzer Report (s. Abschnitt 2.3) führt einen Effizienztests des Codes durch. Der TODO/FIXME Report durchforstet die Files des Verzeichnisses und gibt zu allen Files, in denen das Schlüsselwort TODO oder FIXME verwandt wurde, die relevanten Zeilenpositionen aus. Der Help Report erstellt eine Liste aller Files ergänzt durch den Help Block (der erste Kommentarzeilenblock in MATLAB-Files). Der Contents Report erstellt eine Übersichtsliste der Files; die erste Kommentarzeile wird bei MATLAB-Dateien mit ausgegeben. Der Dependency Report zeigt in einem eigenen Fenster die wechselseitigen Abhängigkeiten

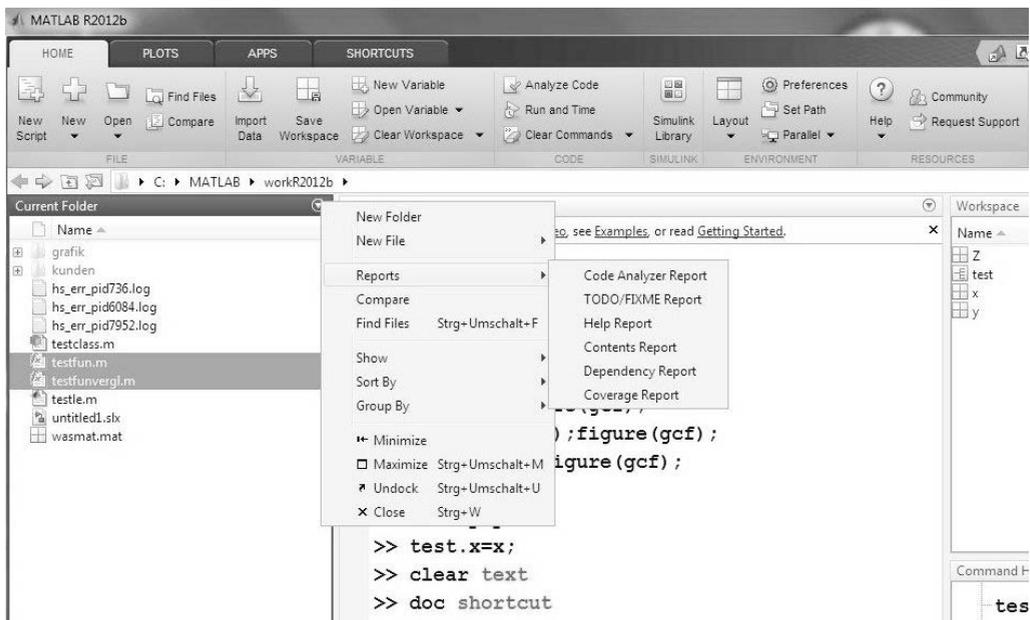


Abbildung 2.6: Current Folder Browser mit dem über die Pfeiltaste geöffneten Dialogfenster.

der MATLAB-Dateien auf. Coverage Report testet, welcher Programmanteil in MATLAB-Dateien tatsächlich ausgeführt worden ist. „Find Files“ öffnet eine grafische Umgebung zum Suchen nach Files oder Verzeichnissen.

MATLAB kann nur diejenigen Dateien ausführen, die entweder über den MATLAB-Suchpfad bekannt sind oder im aktuellen Verzeichnis liegen. Um einige der obigen Reports ausführen zu können, müssen gegebenenfalls Verzeichnisse über „Set Path“ unter der Registerkarte „HOME“ hinzugefügt werden.

File Comparison Tool. Über „Compare“ öffnet sich ein Dialogfenster zur Auswahl der zu vergleichenden Dateien. Alternativ kann auch eine Datei mit der rechten Maustaste ausgewählt werden und im sich öffnenden Dialogfenster unter „Compare Against“ die zu vergleichende Datei, oder beide Dateien werden markiert und via rechten Maustaste über „compare selected files“ Zeile für Zeile miteinander verglichen. Im sich öffnenden Vergleichsfenster wird das Ergebnis farbig dargestellt und man kann durch Mausclick auf den Dateinamen direkt in den Editor springen bzw. durch Klicken auf die Zeilennummer in die zugehörige Codezeile, die im MATLAB-Editor farbig gekennzeichnet wird.

Mit `>> filebrowser` wird der Current Folder Browser aus dem Command Window heraus aktiviert.

2.1.2 Hilfe und Dokumentation

Neben dem `help`-Kommando verfügt MATLAB über eine sehr umfangreiche on- und offline Dokumentation. In den Präferenzen läßt sich die jeweilige Priorität auswählen. Im folgenden beschränke ich mich im Wesentlichen auf die lokal installierte Dokumentation. Die Web-basierte Alternative, eröffnet zusätzlich die Möglichkeit die Dokumentation nicht installierter Produkte zu nutzen.

Unter der Registerkarte „HOME“ in der Sektion „RESOURCES“ befindet sich das ?-Symbol mit dem man direkt in Dokumentation springt sowie das Menü „Help“, s. Abb. (2.7), mit den Auswahlpunkten Dokumentation, Examples (für die dokumentierten Beispiele), aber auch ergänzende Punkte wie Request Support, Training oder Check for Updates mit denen man auf die entsprechenden Internetseiten geführt wird.

Mit dem Sprung in die Dokumentation kann man über die Produktliste interessierende Themen aufrufen. Über die gelbe Stern Taste können eigene Favoriten dazu gefügt werden. Das Haussymbol führt stets zur Produktübersicht zurück und das Listensymbol zur aktuellen Inhaltsangabe. Im Suchfeld können geeignete Suchbegriffe eingegeben werden. Alternativ können Suchbegriffe im MATLAB-Desktop direkt unter „Search Documentation“ übergeben werden. Abb. (2.8) zeigt ein Beispiel zum Suchbegriff „fft“. Im Hauptfenster befindet sich eine kurze Beschreibung der Treffer. Durch Anklicken springt man in das entsprechende Fenster. Je nach Release wird der Suchbegriff farblich hervorgehoben. Auf der linken Seite besteht die Möglichkeit eine Verfeinerung des Suchergebnisses nach Produkten, Kategorien und Typen durchzuführen. Vor MATLAB-Rel. 8.0 war eine Verfeinerung der Suche nur über die ausgewählten Produkte unter „Preferences“ möglich.

Zur Dokumentation gehört eine vollständige Funktionsübersicht der einzelnen MAT-

LAB-Kommandos, die stets aus einer schlagwortartigen Funktionsbeschreibung, einer Übersicht der erlaubten Syntax, aus einer ausführlichen Beschreibung, gegebenenfalls ergänzenden Hinweisen, einem Beispielteil und einem Verweis auf verwandte MATLAB-Kommandos sowie in einigen Fällen aus ergänzenden Literaturangaben besteht. Der Beispielteil erlaubt das Austesten des Befehls. Hier einfach mit der linken Maustaste markieren, mit der rechten Maustaste ergibt sich dann die Möglichkeit, den markierten Teil zu kopieren oder im MATLAB Command Window auszuführen und dort zum weiteren Experimentieren geeignet zu verändern. An die entsprechende Stelle in der Dokumentation kann direkt mit dem Befehl `doc befehl` gesprungen werden, wobei „befehl“ für den Namen des Kommandos steht. Mit der rechten Maustaste öffnet sich ein Dialogfenster beispielsweise zum Drucken und Speichern der Information. In dem mit der Lupe gekennzeichneten Feld kann wieder eine Suche über die gesamte Dokumentation durchgeführt werden.

Die lokal verfügbare Dokumentation wird durch pdf-Dokumentationen im Internet ergänzt. Nach der Auswahl des interessierenden Produkts auf der Startseite der Dokumentation, beispielsweise MATLAB, befindet sich in der untersten Zeile die Auswahl PDF Documentation. In der untersten Zeile kann eine Übersicht aller MATLAB-Funktionen ausgerufen werden.

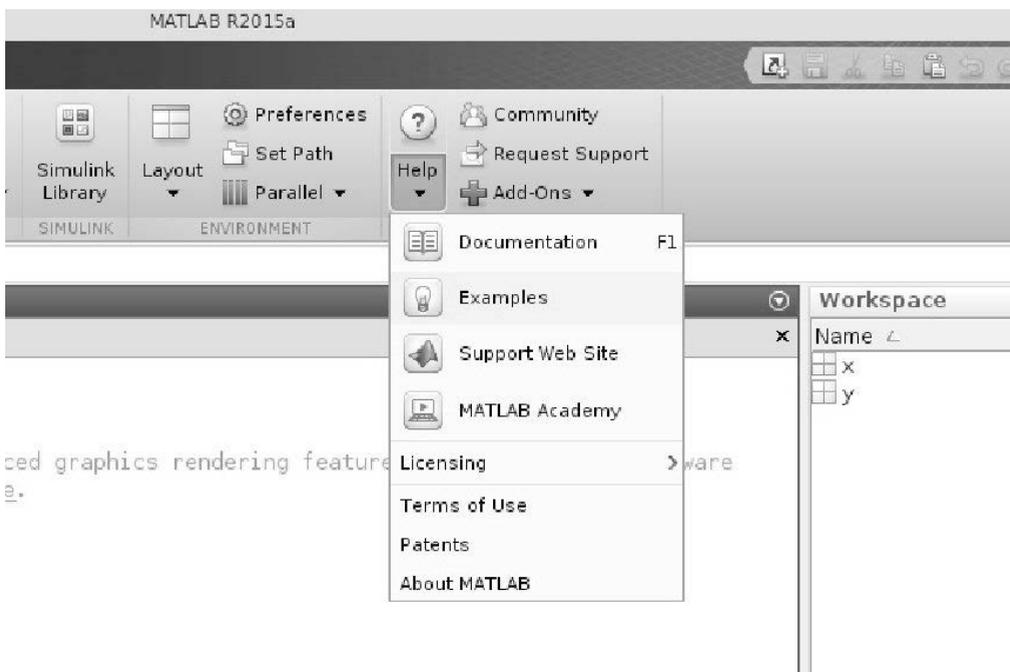


Abbildung 2.7: Öffnen der Hilfe aus der Werkzeugleiste.

2.1.3 Eigene Applikationen und Toolboxen erstellen

Applikationen (Apps) verwalten und erstellen. Die dritte Registerkarte „APPS“ ist den Applikationen (Apps) vorbehalten, aufgeteilt in die Bereiche „FILE“ und „APPS“, Abb. (2.9). Applikationen (Apps) sind auf grafische Benutzeroberflächen basierende Anwendungen. Der Begriff „Apps“ wird also dreifach verwendet. Bei alten MATLAB-Releases finden sich ein Teil der Anwendungen in der linken unteren Ecke unter „Start“.

In der Sektion „APPS“, Abb. (2.9), können eigene Anwendungen mit eingebunden werden. Klicken auf ein Symbol öffnet die zugehörige Apps und Klicken auf die rechte Pfeiltaste öffnet alle verfügbaren Anwendungen. Ganze Bereiche lassen sich über das Top-Symbol nach oben bewegen. Durch Klicken mit der rechten Maustaste können ausgewählte Applikationen den Favoriten oder der „Quick Access Toolbar“ (Abb. (2.3) hinzu gefügt werden.

Was sind und wie lassen sich eigene Applikationen erstellen?

Applikationen (Apps) basieren auf grafischen Benutzeroberflächen. Eine direkte Datenübergabe wie bei einem Funktionsaufruf ist dabei nicht erlaubt. Um ein eigenes Apps zu erstellen, wird im

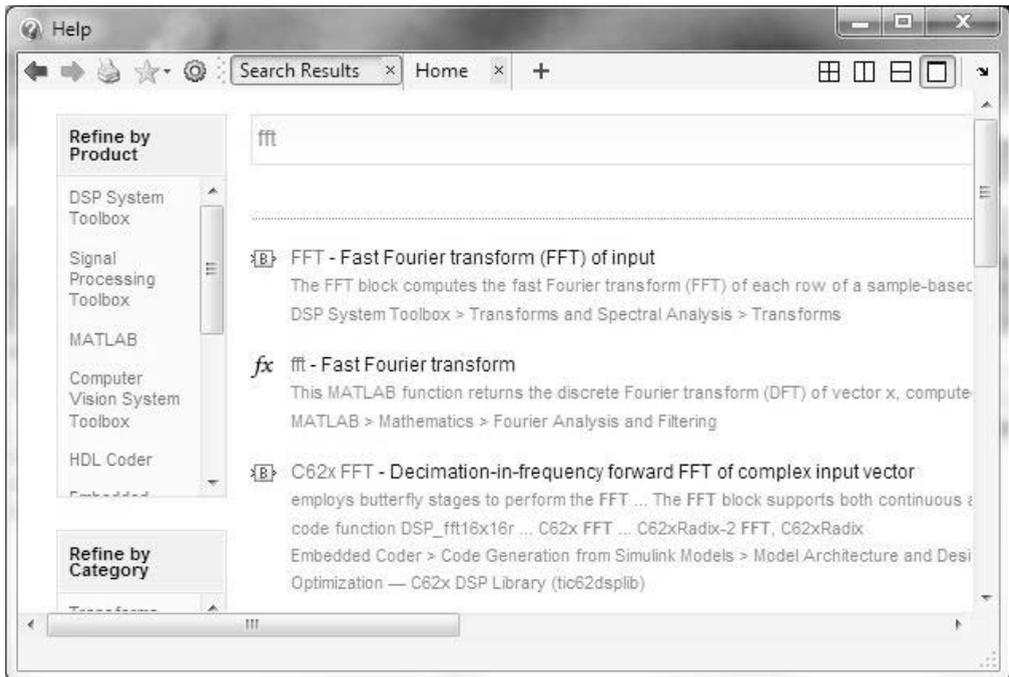


Abbildung 2.8: Die MATLAB Dokumentation. Über das +-Symbol lassen sich gleichzeitig mehrere Hilfeseiten öffnen.



Abbildung 2.9: Unter der Registerkarte „Apps“ befinden sich auf grafischen Benutzeroberflächen basierende Anwendungen aller installierten Produkte sowie selbst erstellte Applikationen.

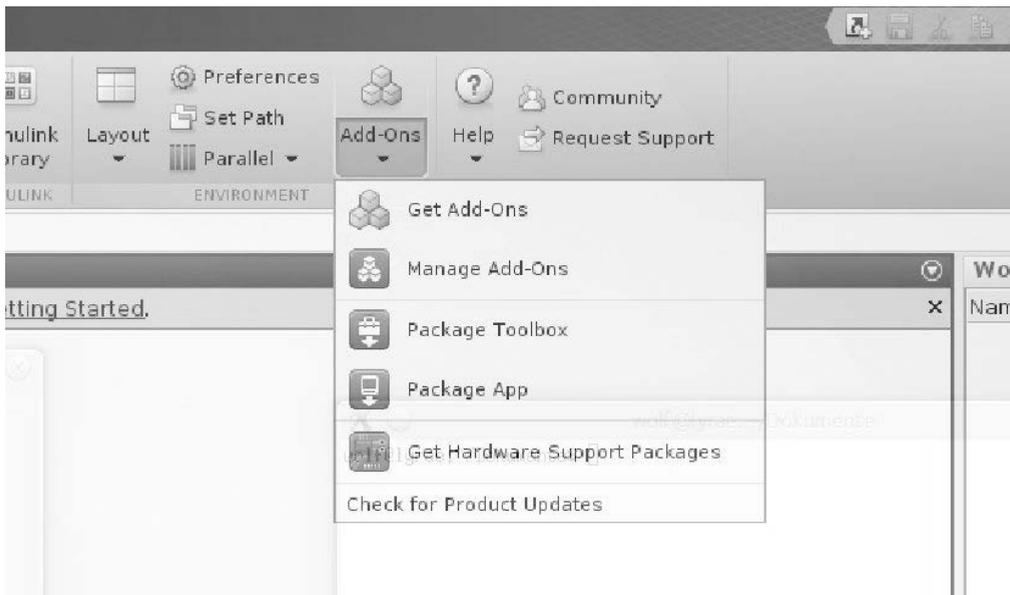


Abbildung 2.10: Unter „Add-Ons“ befindet das Menü Package Toolbox aber auch weiter Funktionalitäten wie beispielsweise der Download-Button zum Herunterladen der Hardware-Support-Packages.

- ersten Schritt das Apps-Installationsverzeichnis über „HOME“ → „Preferences“ → „Apps“ ausgewählt. Dieses Verzeichnis dient als Stammverzeichnis für alle zukünftig erstellten Applikationen. Hier werden beim Installieren eines Apps automatisch Unterverzeichnisse mit dessen Namen angelegt.
- Im nächsten Schritt wird ein beliebiges Verzeichnis ausgewählt in dem die Applikation selbst erstellt wird. Die Hauptfunktion der Applikation muss auf einer grafischen Benutzeroberfläche ohne direkte Datenübergabe basieren.
- Klicken auf „Package App“ öffnet ein Dialogfenster auf dem links die Hauptfunktion (Add main file) eingegeben wird. Dieser Main-File wird automatisch analysiert und erkannte externe Unterfunktionen dazugepackt. Nicht erkannte Abhängigkeiten wie Unterfunktionen, die sich nicht im MATLAB-Pfad befinden müssen von Hand eingegeben werden. In der Mitte wird insbesondere der Name des Apps festgelegt und eine ergänzende Beschreibung übergeben.
- Durch Klicken auf „Package“ werden in diesem Verzeichnis zwei Files mit dem Namen der Applikation und der Dateierweiterung .mlappinstall und .prj erzeugt.
- Letzter Schritt ist durch Klicken auf „Install App“, das erstellte App zu installieren. Dabei wird im Apps-Installationsverzeichnis ein Unterverzeichnis mit dem Namen dieser Applikation erstellt und die entsprechenden Dateien hineinkopiert bzw. erstellt.

Deinstallieren lassen sich diese Apps per Dialog durch Klicken mit der rechten Maustaste. Verbleibt noch der Menüpunkt „Get More Apps“. Damit lassen sich per Mausklick über das MathWorks File Exchange Portal im Internet zur Verfügung gestellte Apps einbinden.

Eigene Toolboxes erstellen. Unter dem Menü „Add-Ons“, Abb. (2.10), findet sich das Symbol „Package Toolbox“. Mit dessen Hilfe lassen sich selbst erstellte Toolboxes in einem Installationsfile einbinden und verteilen. Eine Toolbox besteht aus Funktionalitäten und einer Hilfe eingebunden in die MATLAB-Dokumentation. Ein Beispiel (FraktaleTB) findet sich in den beigefügten Dateien. Wie ist die prinzipielle Vorgehensweise?

- Anlegen eines Toolbox-Ordners mit einem geeigneten Namen. In diesem Verzeichnis wird die Datei „info.xml“ erstellt. Außerdem werden die zugehörigen m-Files (gegebenenfalls in Unterordnern strukturiert) in dieses Verzeichnis kopiert.
- Anlegen eines Dokumentationsverzeichnisses im Toolbox-Ordner. Im Dokumentationsverzeichnis wird die Datei „helptoc.xml“ erstellt sowie die html-Files der Hilfe.
- Unter „Add-Ons“ auf „Package Toolbox“ klicken. Es öffnet sich eine selbsterklärendes Fenster „Package a Toolbox“ in dem die Toolbox Informationen eingegeben werden. Klicken auf „Package“ erstellt einen Installationsfile `NameTB.mltbx`, der an andere Nutzer weitergegeben werden kann und alle Funktionalitäten und Dokumentationen der Toolbox umfasst. Installiert wird durch Doppelklick.

info.xml Die im Toolbox-Ordner residierende Datei „info.xml“ hat eine streng vorgeschriebene Form und enthält Informationen über Release, Toolbox Name und Ort der Hilfe. Hier ein Beispiel:

```
<productinfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="optional">
  <?xml-stylesheet type="text/xsl"href="optional"?>
  <matlabrelease>2015b</matlabrelease>
  <name>FraktaleTB</name>
  <type>toolbox</type>
  <icon></icon>
  <help_location>dokumentation</help_location>
</productinfo>
```

Ein Template befindet sich in der MATLAB-Dokumentation.

Erstellen der Toolbox-Dokumentation. Die Dokumentation besteht aus html-Dateien, die die einzelnen Funktionen und Toolbox Eigenschaften beschreiben. Diese html-Dateien können direkt entworfen werden oder auch mittels der Publish-Funktionalität (s. 2.2.1) erstellt werden. Beispielsweise kann ein Dummy-Verzeichnis erstellt werden in das die m-Files der Toolbox kopiert werden und mit geeigneten zusätzlichen Hilfetexten versehen werden. Hier können auch weitere nur für die Dokumentation vorgesehene m-Dateien erstellt werden. Mittels „Publish“ werden dann die html-Dateien erzeugt und in das Dokumentationsverzeichnis der Toolbox kopiert. In diesem Verzeichnis wird auch der File „helptoc.xml“ erstellt, der alle Metainformationen zur Dokumentation enthält. Auch dazu ein Beispiel:

```
<?xml version='1.0' encoding="utf-8"?>
<toc version="2.0">
  <tocitem target="FracTB.html">Fraktale Toolbox
    <tocitem target="Anfang.html"
      image="HelpIcon.GETTING_STARTED">Getting Started Guide
    <tocitem target="functionlist.html">Liste der
      MATLAB-Files</tocitem>
  </tocitem>
  <tocitem target="Barnsleys_Farn.html">Barnsleys Farn</tocitem>
    <tocitem target="Koch_Kurve.html">Koch Kurve</tocitem>
    <tocitem target="Sierpinski_Dreieck.html">Sierpinski
      Dreieck</tocitem>
    <tocitem target="logbsp.html">Logistische
      Abbildung</tocitem>
    <tocitem target="Undokumentiertes.html">Wat
      n dat?</tocitem>
  </tocitem>
</toc>
```

Die Dokumentation kann auch noch mittels `builddocsearchdb('Absoluter Pfad')` durch eine Hilfsdatenbank ergänzt werden. Um aus jedem Verzeichnis Zugriff auf die Toolbox zu haben, müssen die Toolboxdirektories in den MATLAB-Suchpfad eingebunden werden. Sowohl beim Erstellen von Applikationen als auch von Toolboxen werden zusätzlich Projektdateien mit der Dateierweiterung „.prj“ erstellt, die alle notwendigen Informationen enthalten und die wieder mit dem Toolbox- bzw. Applikationsfenster geladen werden können.

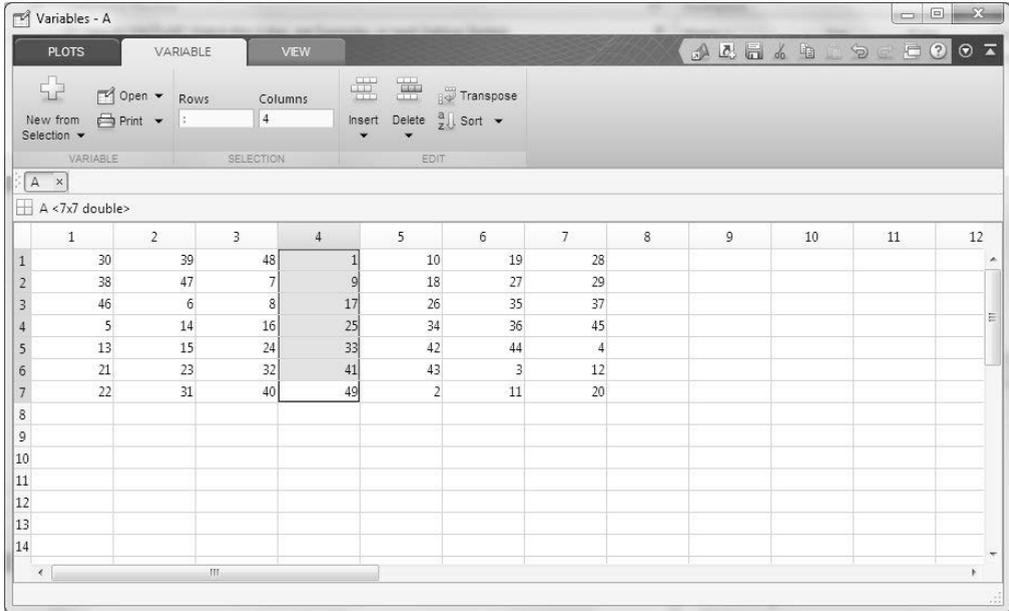


Abbildung 2.11: Der Variable Editor ist ebenfalls mittels geeigneter Registerkarten strukturiert.

2.1.4 Der Variable Editor

Der Variable Editor, Abb. (2.11), (früher Array Editor) dient dem Visualisieren und interaktiven Editieren von Workspace-Variablen und wird durch Klicken auf die Variable im Workspace geöffnet. Alternativ kann der Variable Editor mittels `openvar('name')` aus dem Command Window geöffnet werden. „name“ ist der Name der Variablen. Der Variable Editor kann wie bereits erwähnt am MATLAB Desktop angedockt werden. Spalten im Editor lassen sich kopieren, löschen und teilen. Durch Auswahl einzelner Elemente, ganzer Spalten oder Zeilen durch Klicken auf den nummerierten Spaltenkopf bzw. Zeile können über „New from Selection“ neue Variablen erstellt werden. Exceldaten lassen sich unter Windows-Betriebssystemen mit Copy und Paste in den Variable Editor kopieren. Mittels „Insert“ und „Delete“ lassen sich interaktiv Zeilen und Spalten einfügen bzw. löschen. Mit Hilfe der Maus und der Umschalt- bzw. Steuertaste können beliebige einzelne Elemente, Zeilen oder Spalten zur weiteren Bearbeitung ausgewählt werden.

Mit der rechten Maustaste läßt sich zu ausgewählten Elementen ein Dialogfenster öffnen, das neben dem Kopieren und Ausschneiden auch das Ersetzen der Werte durch Nullen erlaubt. *Achtung, Veränderungen werden stets sofort wirksam und die ursprüngliche Variable überschrieben.*

Nach Auswahl einer Spalte erlaubt das Sort-Menü das Umsortieren in auf- bzw. absteigender Reihenfolge nach der ausgewählten Spalte. Bei gleichen Werten innerhalb der Spalte kann auch nach mehreren Spalten sortiert werden. Um nach Zeilen zu sortieren, muss zunächst das Array transponiert werden (Transpose). Jetzt kann man wieder nach Spalten sortieren und erneutes transponieren vervollständigt das Sortieren nach Zeilen.

Die Registerkarte „PLOTS“, erlaubt per Mausklick geeignete Plots zu erstellen und die Registerkarte „VIEW“, beispielsweise das Zahlenformat zu verändern.

2.2 Der MATLAB Editor und Debugger

Der MATLAB Editor und Debugger, Abb. (2.12), dient beispielsweise dem Schreiben von MATLAB-Skripten und -Funktionen sowie dem grafischen Debuggen.

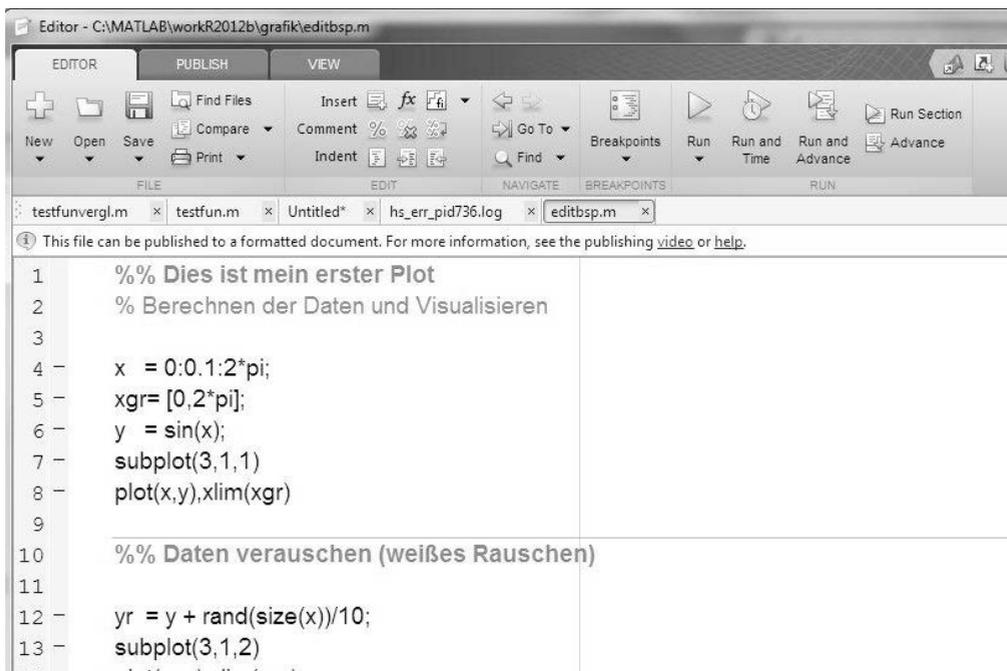


Abbildung 2.12: Der MATLAB Editor im Cell Mode.

2.2.1 Der Editor

Der Editor hebt die MATLAB-Schlüsselworte oder Kommentare in unterschiedlichen Farben hervor, erkennt aber auch andere Formate wie beispielsweise HTML. Unter den Präferenzen lassen sich verschiedene Grundeinstellungen wie Auswahl des Editors, Dateisprache, Zeilenlänge usw. einstellen. Aufgerufen wird der Editor entweder mit

```
>> edit fname
```

zum Editieren des Files „fname“, aus dem Desktop unter HOME → New bzw. Open und dem sich öffnenden Auswahlfenster oder durch Klicken auf einen File im Current Folder Browser. Mit „Find Files“ unter der Registerkarte „HOME“ öffnet sich ein Dialogfenster, das es beispielsweise erlaubt nach Dateien, die einen bestimmten Text enthalten zu suchen und sie mit Edit zu öffnen. Mit der Tastenkombination „Ctrl 0“ springt man direkt aus dem Editor in das MATLAB Command Window und mit „Ctrl-Shift 0“ wieder zurück.

Der Editor ist ebenfalls mittels verschiedener Registerkarten und Sektionen strukturiert und läßt sich an das MATLAB-Desktop andocken. Einige im folgenden beschriebene Eigenschaften des Editors finden sich bei den älteren MATLAB-Versionen im Editor unter den Menüs „Edit“ und „Text“.

Die Registerkarte „EDITOR“ des Editors ist in die Sektionen „FILE“, „EDIT“, „NAVIGATE“, „BREAKPOINTS“ und „RUN“ aufgeteilt. Die Sektion „FILE“ enthält im Wesentlichen dieselben Funktionen wie die gleichnamige Sektion unter „HOME“ ergänzt durch die selbsterklärenden Menüs „Save“ und „Print“. Die Sektion „EDIT“ bietet Hilfestellungen zum Editieren. Kommentarzeilen werden in MATLAB mit einem %-Zeichen und Zellen mit einem Doppel-%-Zeichen eingeleitet, siehe Abb. (2.12). Die Strukturierung in Zellen erlaubt es, Teilbereiche eines MATLAB-Skripts auszuführen und Reports geeignet zu formatieren, s. Abschnitt 2.2.3. „Insert“ fügt zwei %-Zeichen ein, „Comment“ erlaubt das Hinzufügen bzw. Löschen von Kommentarzeilen und -blöcken, dazu muss der entsprechende Textbereich zuvor markiert werden. Das Symbol mit den übereinander liegenden %-Zeichen hängt eine nachfolgende Kommentarzeile an ihren direkten Vorgänger an. Mit „indent“ lassen sich markierte Textbereiche einrücken bzw. rückgängig machen. Einfacher ist das Einrücken markierter Textbereiche über die Tabulatortaste und das Rückgängig machen mit Umschalttaste plus Tabulatortaste.

Die Sektion „NAVIGATE“ erlaubt mit „Go To“ im File zu navigieren und mit „Find“ Text zu suchen und gegebenenfalls zu ersetzen. Die Suche ist dabei nicht auf die aktuelle Datei („Look in:“) beschränkt. Den Abschnitt „BREAKPOINTS“ werden wir im nächsten Abschnitt diskutieren. „RUN“ enthält verschiedene Alternativen zum Ausführen der Datei. Das Menü „Run“ führt die Datei aus und speichert sie gegebenenfalls zuvor ab, „Run and Time“ ruft zusätzlich den Profiler auf und liefert Informationen über die notwendigen Ausführungszeiten einzelner Programmschritte. Mit „Run and Advance“ wird die aktuelle Zelle ausgeführt und in die nachfolgende Zelle gesprungen. „Run Section“ führt die aktuelle Zelle aus und mit „Advance“ springt man in die nachfolgende Zelle.

Die Möglichkeiten, die die Registerkarte „PUBLISH“ bietet werden im Abschnitt „Berichte erstellen“ vorgestellt. Bleibt noch „VIEW“. Im MATLAB-Editor werden einzelne zusammengehörige Programmblöcke wie Unterfunktionen, Schleifen usw. über eine Baumstruktur visualisiert. Diese Bereiche lassen sich in ihre einzelnen Programmschritte

aufflösen (Expand, --Zeichen) oder zu einer Zeilen komprimieren (Collapse, +-Zeichen). Außerdem läßt sich unter „VIEW“ festlegen wie mehrere geöffnete Dateien im Editor angeordnet sind, ob eine einzelne Datei aufgespalten sein soll und ob die Zeilennummern angezeigt werden sollen.

Eine wichtige Eigenschaft des Editors ist die integrierte Code Analyse, s. Abschnitt 2.3. Permanent wird der eingetippte Code auf Korrektheit überprüft und Fehler in der rechten Spalte rot, Warnungen und Verbesserungsvorschläge in orange markiert. Legt man den Mauszeiger über eine Markierung öffnet sich die zugehörige Fehlermeldung bzw. ein Verbesserungsvorschlag wird angezeigt. Seit dem MATLAB-Rel. 7.9 kann durch Mausklick ein Hilfefenster geöffnet werden bzw. der Vorschlag per Mausklick angenommen und automatisch umgesetzt werden. Werden keine Fehler gefunden erscheint im Editor ein grünes Quadrat. Diese Code Analyse kann allerdings keine Laufzeitfehler aufdecken. Dies ist dem Debugger vorbehalten.

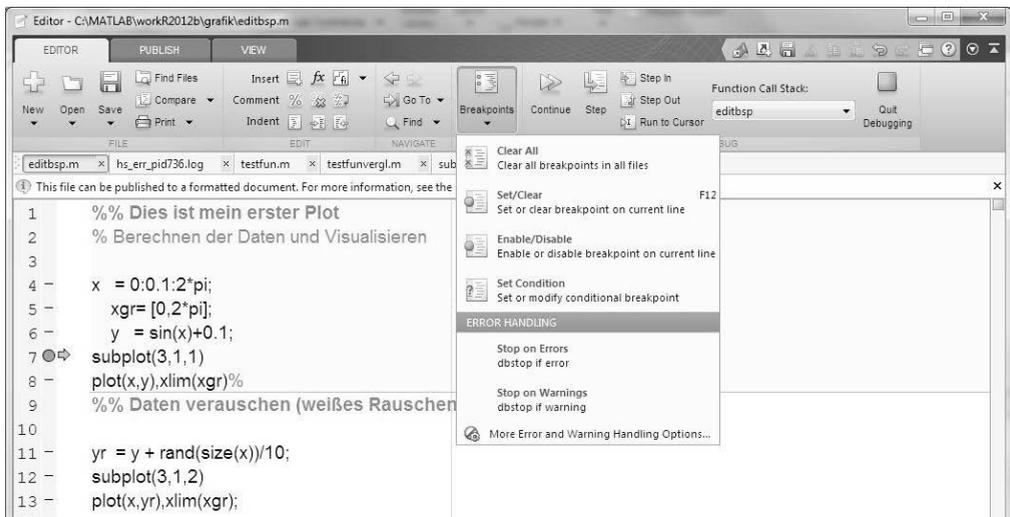


Abbildung 2.13: Setzen von Breakpoints zum Debuggen.

2.2.2 Der grafische Debugger

Der Editor bietet zusätzlich viele Möglichkeiten zum Aufspüren von Laufzeitfehlern und dient als grafischer Debugger, s. Abb. (2.13). Breakpoints zum Unterbrechen des Programmflusses lassen sich entweder durch Klick vor die Zeilennummer oder über das Menü Breakpoints setzen. Starten des Programms öffnet eine zusätzliche Sektion „DEBUG“ in der Werkzeugleiste des Editors. Grafisch gesteuert kann dann eine Funktion schrittweise („Step“) oder von Breakpoint zu Breakpoint springend („Continue“) durchlaufen werden. Mit „Step In“ werden Unterfunktionen bzw. die sich hinter MATLAB-Befehlen verbergenden Funktionen im Editor geöffnet und hinein und mit „Step Out“ wieder herausgesprungen. Mit „Run to Cursor“ wird das Programm bis zur aktuellen Position des Cursors abgearbeitet. Zusätzlich gibt es in der Sektion „DEBUG“

das Auswahlmü „Function Call Stack“ mit dem die Funktion ausgewählt wird deren Variablen im Workspace Browser angezeigt werden sollen. Desweiteren erscheint im Command Window der Debug-Prompt `K>>`; dort können die Variablen der im Function Call Stack angezeigten Funktion aufgerufen und während des Programmablaufs verändert werden (vgl. Kap. 3.8). Breakpoints sind zeilenorientiert und im Regelfall kann pro Zeile nur ein Breakpoint stehen. Eine Ausnahme bilden anonyme Funktionen. In Zeilen mit anonymen Funktionen können mehrere Breakpoints auftreten. Mehrfache Breakpoints werden im Editor blau, einfache rot hervorgehoben.

Die einzelnen Auswahlpunkte im Menü „Breakpoints“ sind selbsterklärend. Während bis Release 7.0 verschiedene Typen von Breakpoints gewählt werden konnten, lassen sich nun über „Set Condition“ Conditional Breakpoints mit beliebige logischen Bedingungen setzen. Zusätzlich können Programmunterbrechungen beim Auftreten von Fehlern, Warnungen, NaNs (not a number), unendlicher Werte (inf) und Try/Catch Werte erfolgen oder unterbunden bzw. auf bestimmte Meldungen eingeschränkt werden. Abb. (2.14) zeigt das zugehörige Dialogfenster, das unter „Breakpoints“ über „More Error and Warning Handling Options ...“ geöffnet wird.



Abbildung 2.14: Dialogfenster für Fehler-Breakpoints.

Nicht gelöschte Breakpoints bleiben beim Speichern eines Programms erhalten. Daher sollten alle Breakpoints nach Beseitigung der Laufzeitfehler eines Programms gelöscht werden. Andernfalls springt man beim erneuten Aufruf dieses Programms wieder in den Debug-Modus. Breakpoints werden gelöscht durch Klicken auf den Breakpoint oder über das Menü „Breakpoints“.

2.2.3 Berichte erstellen

Seit Release 7 wird das Erstellen von Berichten aus dem Editor heraus unterstützt. Unter MATLAB-Rel.7.x muss zunächst der Cell-Mode (im Menü „Cell“) aktiviert werden. Unter „Cell“ → „Insert Text Markup“ können verschiedene Textstrukturen, \LaTeX -basierte Gleichungen etc. eingefügt werden. Das Reportsymbol (Papiersymbol) führt die

Datei aus und erstellt den Bericht per Voreinstellung als HTML-Dokument. Änderungen der Voreinstellungen, wie die Auswahl anderer Formate, erlaubt ein Auswahlfenster, das über das danebenliegende Pfeilsymbol geöffnet wird.

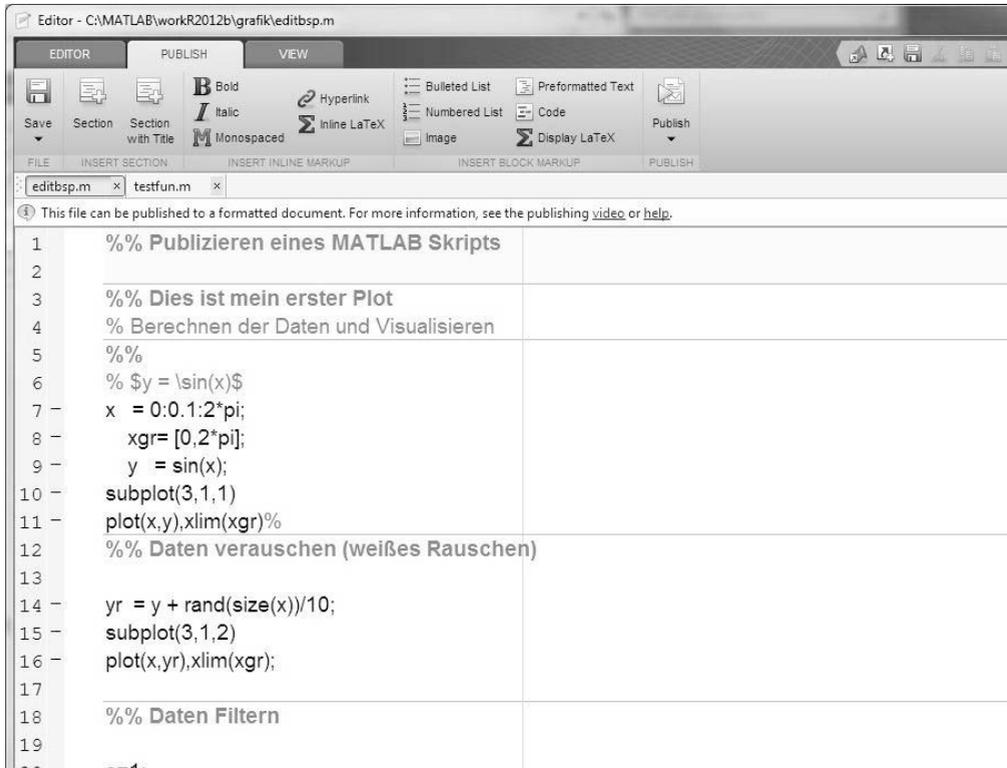


Abbildung 2.15: Registerkarte „PUBLISH“ mit einem Beispielskript.

Seit dem MATLAB-Rel. 8.0 ist das Erstellen von Berichten unter der Registerkarte „PUBLISH“ angesiedelt. Eine wichtige Rolle spielen die %%-Zeichen in der zu publizierenden Datei, Abb. (2.15). Die %%-Zeichen dienen als Titel des Reports bzw. als Kapitelüberschriften. Zu Beginn wird daraus ein Inhaltsverzeichnis erstellt, das bei HTML-Dokumenten mit Hyperlinks versehen ist. Die Kommentare und die Programmzeilen werden als Text aufgenommen, Figures grafisch eingebunden und Ergebnisse, die im MATLAB Command Window ausgegeben werden, ebenfalls in dem Bericht festgehalten. Das Schriftbild verändern, Hyperlinks und \LaTeX -Gleichungen einfügen wird in der Sektion „INSERT INLINE MARKUP“ per Mausklick unterstützt. Ein Beispiel für eine \LaTeX -Gleichung zeigt Abb. (2.15), Zeile 5 und 6. Das Erstellen von Listen, das Einfügen von Bildern (Image) und Code wird unter „INSERT Block MARKUP“ unterstützt. Erlaubte Bildformate sind png, jpeg, bmp und tiff. Die Auswahl erfolgt über „Publish“ → „Edit Publishing Options . . .“ im Fenster „Edit Configurations“, Abb. (2.16). Hier werden die verschiedenen Eigenschaften ausgewählt. Um aus MATLAB-Funktionen Berichte zu erstellen, wird im Feld „MATLAB-Expression“ der Name der Funktion mit