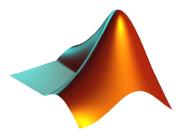
Eine Einführung in

MATIAB

und ein bisschen mehr aus Sicht eines Mathematikers



GÜNTER M. GRAMLICH
Professor für Mathematik an der
Hochschule Ulm
Fakultät Grundlagen
http://www.hs-ulm.de/gramlich
Ulm, 16. Oktober 2007

Vorwort

AT ge

ATLAB¹ ist ein sehr leistungsfähiges Softwaresystem für alle Arten von Berechnungen. Der Numeriker CLEVE MOLER hat die erste Version

von Matlab in FORTRAN Ende der siebziger Jahre geschrieben. Es wurde in Lehre und Forschung beliebt und mündete in ein kommerzielles Softwaresystem. Matlab wird nun in Hochschulen und Industrie eingesetzt.

MATLAB dient im Gegensatz zu Computeralgebrasystemen (CAS) nicht primär der symbolischen, sondern der numerischen (zahlenmäßigen) Lösung von Problemen.

Heutzutage werden viele physikalische, biologische, technische, informationstechnische und ökonomische Produkte am Computer entwickelt. Hierbei ist eine Simulation ein wesentlicher Bestandteil. Mit Hilfe einer Simulation kann ein Funktionsnachweis oft schneller erbracht werden. Außerdem ist sie meist schneller als ein Experiment, kann daher ein Projekt zeitlich verkürzen und so die Kosten reduzieren. Durch Computeranimationen entsteht eine Anschaulichkeit und außerdem ist eine Simulation völlig ungefährlich. Mit dem Softwaresystem Matlab lassen sich realitätsnahe Anwendungen rasch und unaufwendig bereits mit wenigen Codezeilen programmieren bzw. simulieren.

Diese Einführung soll ein Einstieg in Matlab sein. Ich habe nur die wichtigesten Eigenschaften von Matlab behandelt. Dabei zeige ich, wie

und wozu man Funktionen aus Matlab nutzen kann, erkläre aber nicht die Mathematische Theorie und die Algorithmen, die sich dahinter verbergen. Auch gehe ich davon aus, dass Sie grundlegende Kenntnisse im Programmieren und mit dem Umgang wenigstens eines Betriebssystems haben.

Der Umfang von Matlab ist in den letzten Jahren stark angestiegen. An den Dokumentationen können Sie dies gut erkennen, siehe [14, 15, 16, 17]. Hier ein kleiner Abriss über die verschiedenen Versionen:

1978: Klassisches Matlab FORTRAN-Version

1984: Matlab 1 C-Version

1985: Matlab 2

30% mehr Funktionen und Kommandos, Dokumentation

1987: Matlab 3

Schnellere Interpreter, Farbgrafik, hochauflösende Grafik als Hardkopie

1992: Matlab 4

Sparsematrizen, Animation, Visualisierung, User-Interface-Kontrolle, Debugger, Handle-Grafik

1997: Matlab 5

Profiler, objekt-orientierte Programmierung, mehrdimensionale Arrays, Zellenarrays, Strukturen, mehr lineare Algebra für Sparse-Probleme, neue DGL-Löser, Browser-Hilfe

2000: MATLAB 6 (R12)

MATLAB-Desktop mit Browser-Hilfe, Matrizenrechnungen basierend auf LAPACK mit BLAS, Handle-Funktionen, eigs Schnitt-

¹Matlab ® ist eingetragenes Warenzeichen von The MathWork Inc.

stelle zu ARPACK, Randwertproblemlöser, partieller Differenzialgleichungssystem-Löser, JAVA Unterstützung

2002: Matlab 6.5 (R13)

Performance-Beschleunigung, schnellere Geschwindigkeit der Kernfunktionen der Linearen Algebra für den Pentium 4, mehr Fehler- und Warnhinweise

2004: MATLAB 7 (R14)

Mathematik auch auf nicht double-Datentypen (single precision, integer); Anonymous Functions; Nested Functions; m-Files können in HTML, LATEX, usw. publiziert werden; erweiterte und verbesserte Plot-Möglichkeiten

Nicht näher gehe ich auf die Themen: Objektorientiertes Programmieren mit Matlab, JavaSchnittstellen, GUI (Graphical User Interface) Werkzeuge und die Publikationstools wie
HTML, XML, LATEX, usw. ein. Dafür finden Sie
aber einen Zugang zum symbolischen Rechnen
mit Matlab, zu Optimierungs- und Statistikfunktionen, sowie eine Einführung in Simulink
(Symbolic-Toolbox, Optimization Toolbox, Statistics Toolbox, Simulink). Im Anhang finden
Sie ein kleines Glossar, sowie eine Auflistung
wichtiger Matlab-Funktionen.

Den vorliegenden Text habe ich vollständig in LATEX erstellt. Die Literaturhinweise wurden mit BibTEX und der Index mit MakeIndex erzeugt. Alle Bilder habe ich mit Matlab erstellt.

Die Mathematik habe mit den dort üblichen Symbolen und der dort üblichen Schreibweise ausgedrückt. Vektoren sind kleine (a, b, c usw.) und Matrizen sind große lateinische Buchstaben (A, B, C usw.). Funktionen, Kom-

mandos, Codes, usw. aus Matlab habe ich in die Schriftart Typwriter gesetzt. Das Ende von Aufgaben habe ich wie folgt gekenntzeichnet:

Für jede Anregung, nützlichen Hinweis oder Verbesserungsvorschlag bin ich dankbar. Sie erreichen mich am Besten über E-Mail: gramlich@hs-ulm.de. **Dank** an Markus Sommereder (Wien) für den ein oder anderen Fehlerhinweis.

Nun viel Freude und Erfolg mit MATLAB!

Ulm, 16. Oktober 2007 Günter Gramlich

Inhaltsverzeichnis			17.	Das lookfor-Kommando	27
1.	Einführung	9	18.	Alle Funktionen?	27
1.1. 1.2.	Erste Schritte	9 10	19.	Wichtige Funktionen?	28
1.3.	Grafik	15	20.	Der Path Browser	28
2.	Allgemeines	16	21.	Den Datenträger verwalten	28
3.	Was macht den Erfolg von Matlab aus?	19	22.	Wie man weitere Systeminformationen erhält	28
4.	Starten und beenden	19	23.	Neuigkeiten und Versionen	29
5. 5.1.	Die Arbeitsoberfläche Das Command Window	19 20	24.	Voreinstellungen	29
5.2. 5.3.	Command History	20 21	25.	Einfaches Rechnen	29
5.4.	Current Directory	22	26.	Welche arithmetische Operation	
6.	Der Help Browser	22		hat Vorang?	30
7.	Der Array Editor	23	27.	Zahlen und Formate	31
8.	Der Editor/Debugger	23	28.	Variablen und Konstanten	32
9.	Plots	25	29.	Komplexe Zahlen	32
10.	Der Import Wizard	25	30.	IEEE-Arithmetik und double	33
11.	Matlab unterbrechen	25	31.	Nicht double-Datentypen	35
12.	Lange Eingabezeilen	25	32. 32.1.	Merkmale von Matlab Keine Deklaration notwendig	37 37
13.	Eine Sitzung aufzeichnen	25	32.2. 32.3.	Variable Argumentenliste Komplexe Arrays und Arithmetik	37 38
14.	Das help-Kommando	26	33.	Mathematische Funktionen	38
15.	Das doc-Kommando	26	33.1. 33.2.	Rundungsfunktionen	40
16.	Demos	26		Funktionen in Matlab	40

34.	Vektoren	41	40.18.	Flächenstücke (Patches)	77
35.	Vektorenoperationen	43	41.	Vergleichsoperatoren, Vergleichsfunktionen	78
36.	Matrizen	43			
36.1.	Matrizen erzeugen	43	42.	Logische Operatoren und logische	
36.2.	Der Doppelpunkt	46		Funktionen	78
36.3.	Matrizen- und Arrayoperationen	48	42.1.	Logische Operatoren	78
36.4.	Matrizenmanipulationen	52	42.2.	Logische Funktionen	79
36.5.	Datenanalye	53		· ·	
			43.	Steuerstrukturen	7 9
37.	Vektorielle Programmierung	55	43.1.	for-Schleife	79
• •			43.2.	while-Schleife	80
38.	Ein- und Ausgabe	56	43.3.	if-Anweisung	81
38.1.	Benutzereingabe	56	43.4.	switch-Anweisung	81
38.2.	Bildschirmausgabe	57			
38.3.	Dateien lesen und schreiben	58	44.	m-Files	81
39.	Function Functions	59	44.1.	Script-Files	81
39.	runction runctions	39	44.2.	Function-Files	83
40.	Grafik	59	44.3.	Namen von m-Files	84
40.1.	2D-Grafik	60	44.4.	Editieren von m-Files	84
40.2.	3D-Grafik	62	44.5.	Zur Struktur eines m-Files	84
40.3.	Funktionsdarstellungen	64	44.6.	Blockkommentare	85
40.4.	Parametrisierte Kurven	67	44.7.	Übungsaufgaben	85
40.5.	Parametrisierte Flächen	68			
40.6.	Implizite Kurven	70	45.	Globale und lokale Variablen	88
40.7.	Implizite Flächen	70	16	NT.	0.0
40.8.	Koordinatenachsen skalieren	70	46.	Namenstest	88
40.9.	Zwei y-Achsen	72	47.	Wie man effiziente Programme	
	Koordinatentransformationen	72	47.	schreibt	88
	Spezielle Grafikfunktionen	72		schielot	00
	Vektorfelder visualisieren	72	48.	Lineare Algebra (Teil 1)	92
	Grafiken importieren, exportie-		48.1.	Lineare Gleichungssysteme und	
10.15.	ren und drucken	74	10.11	Matrizen	92
40 14	Animationen	76	48.2.	Vektoren in der Ebene und im	
	Handle Graphics	76	.0.2.	Raum	95
	Graphical User Interface (GUI) .	76	48.3.	Analytische Geometrie von Ge-	,,
	Geometrische Körper	76		raden und Ebenen	97
/ •	- · · · · · · · · · · · · · · · · · · ·				- '

48.4.	Reelle Vektorräume und Unter-		51.3.	Subfunctions	128
	räume	97	51.4.	Nested Functions	128
48.5.	Determinanten	101	51.5.	Beispielhafte Funktionen	129
48.6.	Eigenwerte und Eigenvektoren .	102			
48.7.	Lineare Abbildungen und Matrizen	n 105	52.	Lineare Ausgleichsaufgaben	129
48.8.	Matlab-Funktionen für die Li-		53.	Polynome	129
	neare Algebra im Überblick	106	53.1.	Darstellung von Polynomen	130
			53.1.	Nullstellen von Polynomen	130
49.	Lineare Algebra (Teil 2)	107	53.2.	Multiplikation von Polynomen	131
49.1.	Lineare Gleichungssysteme (2) .	107	53.4.	Addition und Subtraktion von	131
	Quadratische Systeme	107	33.4.	Polynomen	131
	Überbestimmte Systeme	108	53.5.	Division von Polynomen	131
	Unterbestimmte Systeme	109	53.6.	Ableiten von Polynomen	132
49.2.	Lineare Gleichungssysteme (3) .	110	53.7.	Integrieren von Polynomen	132
49.3.	Lineare Gleichungssysteme (4) .	113	53.8.	Auswerten von Polynomen	132
49.4.	Inverse	113	53.9.	Zusammenfassung	132
5 0			33.3.	Zusammemassung	133
50.	Lineare Algebra (Teil 3)	114	54.	Polynominterpolation	133
50.1.	Normen	114		To be a second	
50.2.	Konditionszahlen	116	55.	Polynomapproximation	134
50.3.	LU-Faktorisierung	117			
50.4.	Cholesky-Faktorisierung	118	56.	Kubische Splineinterpolation	135
50.5.	QR-Faktorisierung	118			100
50.6.	Singulärwertzerlegung	119	57.	Stückweise lineare Interpolation	136
50.7.	Pseudoinverse	121	58.	Nichtlineare Gleichungen (1)	136
50.8.	Eigensysteme	121	36.	Nichtimeare Gleichungen (1)	130
50.9.	Iterative Methoden	121	59.	Optimierung (Teil 1)	140
50.9.1.	Iterative Methoden für lineare			opumerung (1011 1)	1.0
	Gleichungssysteme	122	60.	FFT	141
50.9.2.	Iterative Methoden für Eigensys-				
	teme	123	61.	Integration	143
50.9.3.	Iterative Methoden für Singulär-		61.1.	Mehrfachintegrale	145
	wertsysteme	124	61.2.	Tabellarische Daten	146
50.10.	Funktionen einer Matrix	124	61.3.	Numerische uneigentliche Inte-	
E 1	Mala as Faul Canas	100		gration	148
51.	Mehr zu Funktionen	126	61.4.	Zusammenfassung	148
51.1.	Function-Handles	126	60	D. C	1.40
51.2.	Anonymous Functions	127	62.	Differenzialgleichungen	148

62.1.	Anfangswertaufgaben	149	67.6.	Endliche und unendliche Summen	181
62.2.	Randwertaufgaben	152	67.7.	Differenziation	184
62.3.	Partielle Differenzialgleichungen	154	67.8.	Partielle Differenziation	184
02.0.	- wave-re more magnetic managem		67.9.	Der Gradient	185
63.	Statistik	156		Die Hesse-Matrix	185
				Die Jacobi-Matrix	186
64.	Kombinatorik	156		Integration	186
64.1.	Fakultäten, Binomial- und Poly-			Polynome	189
	nomialzahlen	156		Taylor-Polynome	189
64.2.	Permutationen ohne Wiederholung	159		Die Funktionen funtool und	
64.3.	Variationen ohne Wiederholung .	159		taylortool	190
64.4.	Kombinationen ohne Wiederho-		67.16.	Mehrdimensionale Taylor-	
	lung	159		Polynome	190
64.5.	Permutationen mit Wiederholung	159	67.17.	Lineare Algebra	191
64.6.	Variationen mit Wiederholung .	159		Differenzengleichungen	191
64.7.	Kombinationen mit Wiederholung	160		Differenzialgleichungen	192
64.8.	Weitere Funktionen	160		Die kontinuierliche Fourier-	
				Transformation	195
65.	Zufallszahlen	160	67.21.	Laplace-Transformation	197
65.1.	Gleichverteilte Zufallszahlen	161		Spezielle mathematische Funk-	
65.2.	Normalverteilte Zufallszahlen .	162		tionen	198
65.3.	Im Vergleich: Gleich- und nor-		67.23.	Variable Rechengenauigkeit	198
	malverteilte Zufallszahlen	162		Überblick über alle symboli-	
65.4.	Andere Verteilungen	164		schen Funktionen	200
			67.25.	Weitere Bemerkungen und Hin-	
66.	Stochastische Simulationen	164		weise	200
66.1.	Näherung für π	166			
66.2.	Zum Ziegenproblem	167	68.	Nichtlineare Gleichungen (2)	201
66.3.	Das Geburtstagsparadox	168			
66.4.	Bestimmte Integrale	169	69.	Optimierung (Teil 2)	201
		4.60	69.1.	Lineare Optimierung	202
67.	Symbolisches Rechnen	169	69.2.	Quadratische Optimierung	203
67.1.	Erste Schritte	170	69.3.	Überblick über alle Funktionen	
67.2.	Wie man Maple-Funktionen ver-	1.70		zur Optimierung	204
65.3	wendet	173			
67.3.	Mathematische Funktionen	173	70.	Nichtlineare Ausgleichsaufgaben	204
67.4.	Algebraische Gleichungen	177			• • •
67.5.	Grenzwerte	178	71.	Simulink	205

Erste Schritte	206	83.	Ähnliche Systeme	224
	206	A.	Glossar	225
_				
		В.	Die Top Matlab-Funktionen	227
-				
•		Literat	tur	229
		G.: 1		221
		Stichw	vortverzeichnis	231
-				
	212			
9	212			
weise	212			
Diinn besetzte Matrizen	212			
-				
The Sparsemanzen recinen	210			
Mehrdimensionale Arrays	216			
Datentynen (Klassen)	217			
The state of the s				
Zenen- und Strukturenarrays	21)			
Audiosignale (Töne, Musik)	221			
WWW-Seiten	222			
Das Mirror Land	222			
Das Matlab-Logo	223			
Studentenversion	223			
Statementersion	223			
Cleve's Corner	223			
Handbücher	223			
Duo commicatino	222			
rioganimerups	223			
Literatur	224			
	Konstruktion eines Blockdiagramms Weitere Arbeitsschritte Ein erstes Beispiel Konstruktion des Blockdiagramms Weitere Arbeitsschritte Simulation Beispiele Vereinfachungen Kommunikation mit MATLAB Umgang mit Kennlinien Weitere Bemerkungen und Hinweise Dünn besetzte Matrizen Sparsematrizen erzeugen Mit Sparsematrizen rechnen Mehrdimensionale Arrays Datentypen (Klassen) Zeichenketten (char) Zellen- und Strukturenarrays Audiosignale (Töne, Musik) WWW-Seiten Das MATLAB-Logo Studentenversion Cleve's Corner Handbücher Progammiertips	Konstruktion eines Blockdiagramms	Konstruktion eines Blockdiagramms	Konstruktion eines Blockdiagramms

1. Einführung

It is probably fair to say that one of the three or four most important developments in numerical computation in the past decade has been the emergence of Matlab as the preferred language of tens of thousands of leading scientists and engineers.

Lloyd N. Trefethen, 1997.

"Der Fortschritt der Menschheit ist eng mit der Verwendung von Werkzeugen verbunden. Werkzeuge wie Hammer, Zange oder Baukran verstärken menschliche Fähigkeiten. Werkzeuge wie Fernglas, Mikroskop oder Flugzeug verleihen sogar neue Fähigkeiten. Auch Computer und Computerprogramme sind Werkzeuge. Sie ermöglichen es dem Menschen Berechnungen schneller durchzuführen, auf Knopfdruck Diagramme zu erzeugen und Daten mit hoher Geschwindigkeit über das Internet zu transportieren. Werkzeuge sind zum einen das Ergebnis menschlichen Erfindungsgeistes, zum anderen sind sie aber auch die Grundlage für neue Erkenntnisse und neu Denk- und Arbeitsweisen. So ermöglichte erst die Erfindung des Rades den einfachen Transport größerer Güter über weitere Entfernungen, mit dem Fernrohr entdeckte Galiei die Jupitermonde und mit dem Computer lassen sich Berechnungen durchführen, die jenseits der Möglichkeiten von Papier und Beistift liegen. Werkzeuge haben aber auch eine didaktische Dimension, da ihr Einsatz geplant und der Umgang mit ihnen gelernt und gelehrt werden muss. Darüber hinaus ziehen neue Werkzeuge auch neue Verfahren, Arbeits- und Denkweisen nach sich". ([27])

1.1. Erste Schritte

Um ihnen ein Gefühl dafür zu geben, wie Mat-LAB arbeitet, starten wir gleich mit ein paar Beispielen. Nähere Erklärungen und weitere Informationen erhalten Sie dann in den folgenden Abschnitten.

Variablen werden nach dem Prompt erzeugt und müssen mit einem Buchstaben beginnen. Matlab unterscheidet zwischen Groß- und Kleinbuchstaben. Die Anweisung

erzeugt die Variable x und ordnet ihr den Wert 3 zu. Das Semikolon am Ende der Anweisung unterdürckt die Ausgabe im Command Window. Nach Voreinstellung ist x vom Datentyp double und belegt acht Byte Speicherplatz. Diese Informationen können mit dem Kommando whos erfragen:

>> whos Name	Size	Bytes Class
х	1x1	8 double array
Grand to	al is 1	element

Außerdem hat die Variable x die Size (Größe, Ordnung) 1x1 was bedeutet, dass sie ein Skalar ist. Ein Skalar ist ein Array mit einer Zeile und einer Spalte.

Zeichenketten (Strings) werden in Hochkommas erzeugt.

```
>> s = 'Ich bin ein String';
```

Ein Zeichen benötigt zwei Byte Speicherplatz. Entsprechend ergeben sich bei 18 Zeichen 36 Bytes.

Matrizen (zweidimensionale Arrays) werden mit eckigen Klammern erzeugt.

```
>> A = [1 2 3; 4 5 6]
A =
1 2 3
4 5 6
```

Spalten werden durch Leerzeichen oder Kommas getrennt, Zeilen durch Semikolons. Gibt man A(1,2) ein, so spricht man das Matrixelement 2 an, denn der erste Index bezieht sich auf die Zeile während der zweite Index sich auf die Spalte bezieht. Dies entspricht ganz der mathematischen Notation. Mit dem Doppelpunkt kann man auch ganze Zeilen oder Spalten einer Matrix ansprechen.

In diesem Beispiel die erste Zeile und die zweite Spalte der Matrix A.

Es gibt in Matlab eingebaute Funktionen, die Ihnen viel Arbeit abnehmen. So gibt es zum Beispiel Funktionen, um Matrizen aufzubauen, ohne sie mühsam eingeben zu müssen. Ein Beispiel ist die Funktion rand. Diese Funktion erzeugt eine Zufallsmatrix mit Elementen zwischen 0 und 1. Die Größe der Matrix bestimmen Sie.

0.4447	0.9218	0.4057	
0.6154	0.7382	0.9355	
0.7919	0.1763	0.9169	

Mit dem Operator * können Sie die Matrizen A und B nun im Sinne der Matrizenalgebra (Lineare Algebra) multiplizieren, wenn Sie wollen, denn die Mulitplikation ist definiert.

Neben Matrizen und mehrdimensionalen Arrays (mehr als zwei Indizes) unterstützt MATLAB Zellenvariablen und Strukturvariablen (Abschnitt 74). In eine Zellenvariable kann man Variablen mit verschiedenen Datentypen zusammenpacken.

```
>> Z = {x,s}
Z = [3] 'Ich bin ein String'
```

Die Zellenvariable Z besteht aus einem numerischen Skalar und einem String. Zellenvariablen können mit geschweiften Klammern erzeugt werden.

1.2. Magische Quadrate (magic)

Magische Quadrate sind interessante Matrizen. Mit

```
>> help magic
```

erhalten wir den folgenden Text:

MAGIC(N) is an N-by-N matrix constructed from the integers 1 through N^2 with equal row, column, and diagonal sums. Produces valid magic squares for all N > 0 except N = 2.

Wir erzeugen das magische Quadrat der Ordnung drei.

Das Kommando sum(A) summiert die Elemente jeder Spalte und erzeugt die Ausgabe

```
ans = 15 15 15
```

Das Kommando sum(A') transponiert (Zeilen und Spalten vertauschen) die Matrix A zunächst, berechnet dann die Spaltensummen und bestätigt, dass auch die Zeilensummen gleich 15 sind:

Funktionen dürfen geschachtelt werden! Auch die Summe der Diagonalelemente ist 15:

```
>> sum(diag(A))
ans =
   15
```

Die Gegendiagonale hat auch die Summe 15. Die Gegendiagonale ist für die Lineare Algebra weniger interessant, deshalb ist deren Ansrechbarkeit auch trickreicher.

```
>> sum(diag(flipud(A)))
ans =
    15
```

Die Funktion flipud vertauscht die Zeilen von oben nach unten. Warum ist die Summe

gleich 15? Die Anwort ist: Die Summe der ersten neun ganzen Zahlen ist 45 und da jede Spaltensumme gleich sein muss, gilt 45/3 = 15. Wieviele magische Quadatrate der Ordnung drei gibt es? Antwort: Acht! Es gibt acht Drehungen und Spiegelungen der Matrix A. Genauso viele Möglichkeiten, wie eine Folie auf den Overhead Projektor zu legen. Hier sind sie:

SIC.						
8	1	6	8	3	4	
3	5	7	1	5	9	
4	9	2	6	7	2	
6	7	2	4	9	2	
1	5	9	3	5	7	
8	3	4	8	1	6	
2	9	4	2	7	6	
7	5	3	9	5	1	
6	1	8	4	3	8	
4	3	8	6	1	8	
9	5	1	7	5	3	
2	7	6	2	9	4	

Man kann sie wie folgt erzeugen.

```
for k=0:3
   rot90(A,k)
   rot90(A',k)
end
```

Nun ein etwas Lineare Algebra. Es ist

```
>> det(A)
ans =
-360
```

und die Inverse ist

```
>> X = inv(A)

X =

0.1472 -0.1444 0.0639

-0.0611 0.0222 0.1056

-0.0194 0.1889 -0.1028
```

Diese sieht vertrauter aus, wenn wir die Matrixelemente als Brüche schreiben:

```
>> format rat

>> X

X =

53/360 -13/90 23/360

-11/180 1/45 19/180

-7/360 17/90 -37/360
```

Jetzt kann man die Determinante im Nenner erkennen. Mit format short können wir wieder auf die Defaultausgabe zurückschalten.

Die Norm einer Matrix, die Eigenwerte und die singuären Werte sind wichtige Größen einer Matrix. Hier sind sie für das magische Quadrat der Ordnung drei:

```
>> r = norm(A)

r =

15.0000

>> e = eig(A)

e =

15.0000

4.8990

-4.8990

>> s = svd(A)

s =

15.0000

6.9282

3.4641
```

Wir sehen, dass der magische Summenwert 15 in allen drei Größen vorkommt.

Bis jetzt haben wir alle unsere Berechnungen in Gleitpunktarithmetik durchgeführt. Diese Arithmetik wird im wissenschaftlichen und ingenieurmäßigen Rechnen am meisten verwendet, insbesondere für "große" Matrizen. Für eine (3, 3)-Matrix können wir die Berechnungen leicht symbolisch wiederholen. Dabei verwenden wir die *Symbolic Toolbox*, die auf MAPLE basiert, siehe Abschnitt 67. Die Anweisung

```
>> A = sym(A)
A =
[ 8, 1, 6]
[ 3, 5, 7]
[ 4, 9, 2]
```

konverteiert die Matrix A zu einer symbolischen Matrix. Die Kommandos

```
sum(A), sum(A'), det(A), inv(A),
eig(A), svd(A)
```

produzieren die entsprechenden symbolischen Resultate.

Ein Beispiel eines magischen Quadrats der Ordnung vier findet sich in der Renaissanceradierung *Melancholie* des deutschen Künsters Albrecht Dürer. Diese Radierung steht uns in Matlab elektronisch zur Vefügung. Die Anweisungen

```
>> load durer
>> whos
```

ergeben die Ausgabe

Name	Size	Bytes	Class		
X caption map	648x509 2x28 128x3		double char double		
Grand total is 330272 elements					

Hierbei ist X die Bildmatrix des Grauwertbildes und in der Variablen map ist die Graustufenskalierung enthalten. Das Bild wird mit den Anweisungen

- >> image(X)
- >> colormap(map)
- >> axis image

erzeugt, siehe Abbildung 1. Betrachtet man das

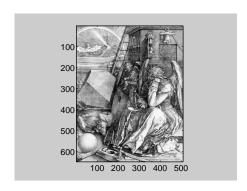


Abbildung 1: Radierung von A. Dürer

Bild genauer, so stellt man fest, dass sich in ihm tatsächlich ein magisches Quadrat befindet. Zoomen Sie mit der Lupe in den rechten oberen Teil des Bildes und Sie können das magische Quadrat der Ordnung vier gut erkennen. Mit den Anweisungen

- >> load detail
- >> image(X)
- >> colormap(map)
- >> axis image

erhalten wir in einer höheren Auflösung den rechten oberen Teil des Bildes mit dem magischen Quadrat, siehe Abbildung 2. Die Anwei-

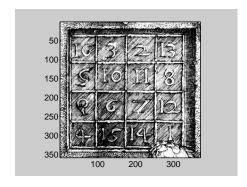


Abbildung 2: Ausschnitt der Radierung

sung

erzeugt das folgende magische Quadrat der Ordnung vier

A	=				
	16	2	3	13	
	5	11	10	8	
	9	7	6	12	
	4	14	15	1	

Die Aufrufe sum(A), sum(A'), sum(diag(A)) und sum(diag(flipud(A))) liefern jeweils den Wert 34 und zeigen somit, dass A ein magisches Quadrat ist.

Dieses magische Quadrat ist aber nicht das Gleiche wie in Dürers Radierung. Wir brauchen aber nur die zweite und dritte Spalte vertauschen. Das geht so:

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

Dürer hat wahrscheinlich dieses magische Quadrat gewählt, weil es in der Mitte der letzten Zeile die Zahl 1514 enthält, was das Jahr ist, in dem er diese Zeichnung getan hat.

Es stellt sich heraus, dass es 880 magische Quadrate der Ordnung vier und 275 305 224 magische Quadrate der Ordnung fünf gibt. Es ist bisher ein ungelöstes mathematische Problem, die Anzahl der verschiedenen magischen Quadrate der Ordnung 6 oder größer anzugeben.

Die Determinante von A ist null. Deshalb hat die Matrix A auch keine Inverse. Das heißt, es gibt magische Quadrate, die singulär sind. Welche? Der Rang einer quadratischen Matrix ist die Anzhal der linear unabhängigen Spalten (oder Zeilen). Eine (n, n)-Matrix ist genau dann singulär, wenn der Rang kleiner n ist. Die Anweisungen

```
for n=1:24
    r(n) = rank(magic(n));
end
[(1:24)' r']
```

erzeugen eine Tabelle in der man mit zunehmender Ordnung den Rang ablesen kann.

```
1
         1
 2
         2
 3
         3
         3
 4
         5
         5
 6
         7
 7
 8
         3
 9
         9
         7
10
11
        11
12
         3
13
        13
14
         9
15
        15
16
         3
17
        17
18
        11
19
        19
20
         3
21
        21
22
        13
23
        23
24
         3
```

Schauen Sie sorgfältig auf die Tabelle. Ignorieren Sie den Fall n=2, denn dann liegt kein magisches Quadrat vor. Können Sie Strukturen erkennen? Mit Hilfe eines Säulendiagramms erkennt man die Strukturen besser. Die Anweisungen

```
>> bar(r)
>> title('Rang magischer Quadrate')
```

erzeugen das Bild in Abbildung 3. Die Beobachtungen sind folgende:

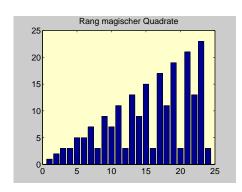


Abbildung 3: Rang magischer Quadrate

- Ist n = 3, 5, 7, ..., so haben die Matrizen vollen Rang. Sie sind also regulär und besitzen jeweils eine Inverse.
- Ist n = 4, 8, 12, ..., so hat die Matrix den Rang drei. Diese Matrizen sind also "stark" singulär.
- Ist $n = 6, 10, 14, \ldots$, so hat die Matrix den Rang n/2 + 2. Auch diese Matrizen sind singulär, nicht aber so stark wie die vorhergehende Klasse.

Mit edit magic können Sie sich den Function-File anschauen, der die magischen Quadrate erzeugt. Dort können Sie auch die obigen drei Fallunterscheidungen wieder erkennen.

1.3. Grafik

MATLAB verfügt über mächtige Grafikfähigkeiten. Dieser Abschnitt zeigt erste Schritte, siehe Abschnitt 40 für weitere Einzelheiten.

Die Funktion plot verfügt über viele grafische Möglichkeiten; sie eine der grundlegenden Grafikfunktionen in Matlab. Sind x und

y zwei Vektoren der gleichen Länge, so öffnet der Befehl plot(x,y) ein Grafikfenster (Figure) und zeichnet die Elemente von x gegen die Elemente von y, das heißt er verbindet die Punkte (x(i),y(i)) durch gerade Linien. Es entsteht ein Polygonzug. Der erste Vektor x bildet die Koordinaten entlang der x-Achse und der zweite Vektor y die Koordinaten entlang der y-Achse.

Die Anweisungen

```
>> k = 1; r = 0.6; y0 = 0.01;
>> x = linspace(0,20);
>> y = k./(1+(k/y0-1)*exp(-r*x));
>> plot(x,y), grid,
```

zeichnen den Graf der logistischen Wachtumstumsfunktion von 0 bis 20 mit den Parametern k = 1 (Tragfähigkeit), r = 0.6 (Wachstumsrate) und der Anfangsbedingung $y_0 = 0.01$ (Anfangsbestand), siehe Abbildung 4. Die darin

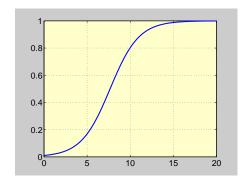


Abbildung 4: Logistisches Wachstum

vorkommenden Rechenoperationen ./, * und / werden in Abschnitt 36.3 erklärt.

2. Allgemeines

MATLAB ist ein sehr leistungsfähiges Softwaresystem für alle Arten von Berechnungen. Der Name MATLAB kommt von MATRIX LABORATORY und verweist auf die zwei Überlegungen, die bei der Entwicklung eine Rolle gespielt haben. Grundelemente sind Matrizen und ihre Manipulation, die in numerischen Verfahren optimal eingesetzt werden können, gleichzeitig verfolgt man über Laboratory den Gedanken der Entwicklung und Erweiterung. MATLAB ist ein interaktives Matrix-orientiertes Softwaresystem, in dem sich Probleme und Lösungen in vertrauter mathematischer Schreibweise darstellen lassen.

Mittlerweile gibt es außer Matrizen bzw. zweidimensionalen Feldern (Arrays) weitaus komplexere Datenstrukturen. Alle Datenstrukturen lassen sich unter dem Oberbegriff des mehrdimensionalen Arrays einordnen. Aus Matlab ist sozusagen nun ein Arrays (Array Laboratory) geworden. Das numerische zweidimensionale Array, also die klassische Matrix, ist in diesem Konzept nur noch ein Spezialfall, aber natürlich ein sehr wichtiger.

Typische Anwendungen sind:

- Numerische Berechnungen aller Art.
- Entwicklung von Algorithmen.
- Modellierung, Simulation und Entwicklung von Prototypen technischer und wirtschaftlicher Probleme.
- Analyse, Auswertung und grafische Darstellung von Datenmengen; Visualisierungen.
- Wissenschaftliche und technische Darstellungen.

• Applikationsentwicklung mit Aufbau einer grafischen Benutzerschnittstelle.

In den siebziger Jahren wurde in den USA eine intensive Aktivität zur Entwicklung hochqualitativer Software gestartet, NATS-Projekt. 1976 lag als Ergebnis dieser Bemühungen das Softwarepaket Eispack zur Lösung algebraischer Eigenwertprobleme vor [26]. Im Jahr 1975 begannen die Arbeiten an einem effizienten und portablen Softwarepaket zur Lösung linearer Gleichungssysteme. Das Ergebnis war das Softwarepaket LINPACK [2]. LINPACK und EISPACK gewährleisteten lange Zeit die zuverlässige und portable Lösung von Problemen der Linearen Algebra. Um diese beiden Pakete leichter handhabbar zu machen. wurde Matlab geschrieben. Damit bestand auch die Möglichkeit, ausgereifte Software effizient in der Lehre - zunächst in der (Numerischen) Linearen Algebra, später und jetzt in vielen anderen Bereichen - einzusetzen. Zur Geschichte von Matlab siehe http://www. mathworks.com/company/newsletters/ news_notes/clevescorner/dec04.html.

Der Einsatz von Matlab lohnt sich. Neben den sonst üblichen Lehrbuchbeispielen können kompliziertere und praxisbezogene Aufgaben schon im Ausbildungsprozess bearbeitet werden. Matlab erhöht die Leistungsfähigkeit, Probleme aus Wirtschaft, Technik und Natur zu lösen, und erhöht die Motivation sich mit Mathematik zu beschäftigen.

Der Umfang von Matlab ist in den letzten Jahren stark angestiegen. Informationen über die neuste Version und andere Hinweise finden Sie unter http://www.mathworks.de.

Die drei Hauptkomponenten von MATLAB sind:

- Berechnung
- Visualisierung
- Programmierung

Berechnung. Matlab verfügt über eine numerische – qualitativ hochwertige – Programmsammlung. Dem Benutzer bleibt es dadurch erspart, Standardalgorithmen neu programmieren zu müssen. Er kann auf grundlegende, gut ausgetestete Programme zurückgreifen und darauf aufbauend eigene Algorithmen realisieren.

Visualisierung. MATLAB verfügt über moderne Visualisierungsmöglichkeiten. Dadurch ist der Benutzer in der Lage, Daten auf verschiedene Art und Weise darzustellen.

Programmierung. Matlab verfügt über eine eigene höhere Programmiersprache. Der Benutzer hat somit die Möglichkeit, die Funktionalität von Matlab durch eigene Programme beliebig zu erweitern. Dies kann dadurch geschehen, dass er Matlab-Programme schreibt – sogenannte m-Files – oder C/C++, Fortran bzw. Java-Codes einbindet. Dadurch stellt Matlab ein offenes System dar.

Die grundlegenden Datenelemente von Mat-LAB sind Matrizen bzw. mehrdimensionale Arrays (Felder), die nicht dimensioniert werden müssen. Dadurch lassen sich viele technische Aufgabenstellungen, vor allem wenn sie mit Matrizen oder Vektoren dargestellt werden können, mit einem Bruchteil des Zeitaufwandes lösen, der für die Programmierung in einer skalaren, nicht interaktiven Sprache wie Fort-RAN oder C/C++ erforderlich wäre.

Im Verlauf mehrerer Jahre und durch Beiträge vieler Benutzer hat sich MATLAB zu sei-

nem heutigen Umfang entwickelt. In Hochschulen ist Matlab das bevorzugte Lehrmittel für Grund- und Aufbaukurse in Mathematik, Ingenieurswissenschaften, Naturwissenschaften und Wirtschaftswissenschaften. In der Industrie findet Matlab immer mehr Zuwachs in Forschung, Entwicklung, Datenauswertung und Visualisierungen aller Art. Folgende Punkte tragen außerdem zum Erfolg von Matlab bei:

Syntax. Matlab besitzt eine benutzerfreundliche, intuitive Syntax, die kurz und einfach ist. Sie lehnt sich stark an die mathematischen Schreibweisen an. Auch einen umgekehrten Prozess kann man beobachten. Matlab nimmt Einfluß auf mathematische Beschreibungen, siehe zum Beispiel [3].

Toolboxen. In Form von sogenannten Toolboxen lässt sich der Funktionsumfang von Matlab auf vielen Gebieten erweitern. Unter anderem stehen folgende Toolboxen zur Verfügung: Extended Symbolic Math Toolbox, Financial Toolbox, Image Processing Toolbox, Neural Network Toolbox, Optimization Toolbox, Partial Differential Equation Toolbox, Signal Processing Toolbox, Spline Toolbox, Statistics Toolbox und Wavelet Toolbox. Darüber hinaus stellt Matlab eine Schnittstelle zur numerischen Programmbibliothek NAG (http://www.nag.com) bereit.

Matrizen. Grundlage von Matlab sind reelle und komplexe (einschließlich dünn besetzter) Matrizen. Die Einführung einer Matrix als grundlegendes Datenelement hat sich nicht nur in der (numerischen) Mathematik, sondern auch in vielen anderen rechnerorientierten Bereichen als sehr vorteilhaft herausgestellt.

Symbolisches Rechnen. Durch die (Exten-

ded) Symbolic Math Toolbox ist es innerhalb der Matlab-Umgebung möglich, symbolisch zu rechnen. Dadurch kann der Benutzer symbolische und numerische Berechnungen miteinander verknüpfen. In Abschnitt 67 wird diese Toolbox genauer beschrieben.

Prototyprealisierung. In der Praxis kommt es vor, dass man – aus den verschiedensten Gründen heraus – darauf angewiesen ist, Algorithmen in anderen Programmiersprachen, wie zum Beispiel C/C++, Fortran, Pascal oder Java, zu implementieren. Aber auch dann ist es vorteilhaft, einen Prototyp des Verfahrens in Matlab zu realisieren, da dies meist sehr schnell möglich ist, bevor man den Algorithmus überträgt bzw. automatisch übertragen lässt (Zum Beispiel mit Hilfe des Matlab C/C++ Compilers).

Handle Graphics (Grafiken bearbeiten). Das Matlab-Grafiksystem umfasst Hochsprachenbefehle für die Darstellung von zwei- und dreidimensionalen Datenstrukturen, für die Bildverarbeitung, für Trickbewegungen und Präsentationsgrafiken. Hierzu gehören auch einfache Befehle, mit denen sich Grafiken kundenspezifisch gestalten oder auch vollständig grafische Benutzerschnittstellen für eigene Applikationen aufbauen lassen.

Bibliothek von mathematischen Funktionen. Matlab verfügt über eine umfangreiche Sammlung von mathematischen Algorithmen und Funktionen. Diese Funktionalität reicht von elementaren Funktionen über Eigenwertberechnungen bis hin zur schnellen Fourier-Transformation.

Application Program Interface (API). Diese Anwenderschnittstelle ermöglicht die Erstellung von Programmen in C/C++ und Fortran,

um sie dann in Matlab einzubinden.

SIMULINK. SIMULINK – ein Partnerprogramm zu MATLAB – ist ein blockorientiertes, interaktives System zur Simulation linearer und nichtlinearer dynamischer Systeme. Es handelt sich um ein mausgesteuertes Grafikprogramm, das ein Modell eines technischen, natürlichen oder wirtschaftlichen Systems, das als Blockdiagramm (Blockschaltbild) auf dem Bildschirm darzustellen ist, unter dynamischen Einwirkungen nachbildet. Es kann für lineare, nichtlineare, zeitkontinuierliche oder zeitdiskrete Prozesse eingesetzt werden. Grundlage sind MATLAB-Funktionen zur Lösung gewöhnlicher Differenzialgleichungen (DGL-Löser, ODE-Löser).

Blocksets sind Ergänzungen zu SIMULINK, die weitere Bausteinbibliotheken für Spezialanwendungen bereitstellen.

STATEFLOW ist eine Erweiterung von SIMULINK zur Simulation ereignisorientierter Modelle (Endliche Zustandsautomaten, Finite State Machines).

Real-time Workshop ist ein Programm, mit dem sich aus den Blockdiagrammen ein C-Code bilden lässt, der von einer Vielzahl von Echtzeitsystemen abgearbeitet werden kann.

Wir verwenden den Begriff Funktion im Folgenden doppeldeutig. Zum Einen sind mathematische Funktionen gemeint und zum Anderen handelt es sich um Matlab-Funktionen, also um Unterprogramme. Aus dem Zusammenhang sollte aber immer klar sein, um welchen Typ Funktion es sich handelt.

3. Was macht den Erfolg von MATLAB aus?

MATLAB hat gegenüber der traditionellen numerischen Programmierung (wie zum Beispiel mit FORTRAN, C/C++ oder dem Aufruf von numerischen Bibliotheken) folgende Vorteile:

- Matlab verfügt über eine benutzerfreundliche und intuitive Syntax; die Syntax ist kurz und einfach.
- Die numerischen Programme zeichnen sich durch eine hohe Qualität aus.
- In einer eingebauten höheren Programmiersprache lassen sich Algorithmen schnell und leicht realisieren.
- Datenstrukturen erfordern minimale Aufmerksamkeit; zum Beispiel müssen Arrays nicht deklariert werrden, bevor man sie benutzt.
- Ein interaktives Arbeiten erlaubt schnelles experimentieren und leichte Fehlersuche.
- Matlab verfügt über mächtige, benutzerfreundliche und qualitativ hochwärtige Grafik- und Visualisierungsmöglichkeiten.
- Matlab m-Files sind für eine große Klasse von Plattformen kompatibel.
- Es bestehen Erweiterungsmöglichkeiten durch Toolboxen (Signalverarbeitung, symbolische Rechnungen usw.).
- Über das Internet sind viele m-Files von anderen Benutzern zu bekommen.

Wir geben hier nur eine Einführung in die Mächtigkeit von MATLAB. Für ausführlichere Darstellungen bezüglich MATLAB und Mathematik (numerisch und symbolisch) ver-

weise ich Sie auf unser Buch [7] bzw. auf [9] und [23] und natürlich auf die MATLAB-Dokumentationen [14, 15, 16, 17].

4. Starten und beenden

Bei vielen kommandoorientierten Rechnersystemen wird Matlab durch das Kommando matlab gestartet. Oder – bei grafischen Oberflächen – klickt man nach dem Start auf ein entsprechendes Matlab-Icon. Bei manchen Installationen ist es auch möglich, dass Sie Matlab aus einem Menü heraus aufrufen können. In jedem Fall sollten Sie den Matlab-Prompt » sehen (bzw. EDU»). Mit dem Kommando quit (exit) verlassen Sie Matlab. Weitere Hinweise finden Sie in den Matlab-Handbüchern. Gegebenenfalls müssen Sie Ihren Systemmanager nach lokalen Installationseigenschaften befragen.

5. Die Arbeitsoberfläche

Wir geben eine Übersicht über die Matlab Arbeitsoberfläche (Desktop) und ihrer einzelnen Elemente: den Help Browser, Arrayeditor und Editor/Debugger.

Nach dem Start öffnet sich die Matlab Arbeitsoberfläche, die auch als Desktop bezeichnet wird, siehe Abbildung 5. Die Arbeitsoberfläche von Matlab besteht aus dem Command Window zur Befehlseingabe, dem Current Directory mit dem Inhlatsverzeichnis, dem Workspace mit den Variablen des Speicherbereiches des Command Windows, der Command History, verschiedener Menü-

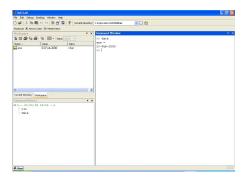


Abbildung 5: Die Arbeitsoberfläche

zeilen sowie dem Start Button. Die einzelnen Komponenten können in ihrer Positionierung und Größe mit der Maus verändert und durch Festhalten der Namensliste mit der linken Maustaste umgruppiert werden. Mit der Dock-Eigenschaft lassen sich die einzelnen Komponenten abkoppeln und in unabhängige Fenster wandeln. Figure Windows, Editor/-Debugger und Arrayeditor lassen sich an das Command Window ankoppeln und so ein größeres Dokument erzeugen. Dabei bestehen unterschiedliche Möglichkeiten der Anordnung, beispielsweise nebeneinander oder hintereinander. Probieren Sie es aus! Für häufig sich wiederholende Aufgaben lassen sich eigene Shortcuts definieren und in die Liste der Shortcuts einfügen.

Der Start Button erlaubt den direkten Zugriff auf unterschiedliche Funktionalitäten, nicht nur unter Matlab, sondern auch auf Funktionalitäten von Toolboxen, Simulink, Stateflow und Blocksets. Hier befindet sich auch ein einfacher Zugang zu den erwähnten Shortcuts oder interessanten Matlab Seiten, Hilfeseiten und Demos, um nur eine kleine Auswahl

zu nennen.

5.1. 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 Prompt

>>

Befehle können auch abgekürzt werden und mit der Tab-Taste ergänzt werden. Zu früheren Eingaben kann mit den Kursortasten zurückgeblättert werden. Erzeugte Variablen werden im Workspace Window aufgelistet, Befehle in der Kommando History.

5.2. Command History

Im Command History werden die eingegebene 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 wrden. Die linke Maustaste geneinsam mit der Sift- oder 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.

5.3. Der Workspace Browser

Der Workspace Browser zeigt die aktuellen Variablen des Base-Speicherbereichs (Command Window) oder beim Debuggen die des zugehörigen Funktionsspeicheraumes an. Die Variablen sind alphabetisch geordnet, können aber auch durch Anklicken der Spaltenüberschriften nach den damit verküpften Eigenschaften umgeordnet werden.

Die Menüleiste des Workspace Browser bietet mit dem Plot-Zeichen einen bequemen Zugang zu grafisch gesteuertem Plotten einer oder mehrerer Variablen.

Ist der Workspace Browser im Vordergrund, weist die Desktop-Menüleiste zusätzlich das Menü *Graphics* auf. Wählt man dort *Plot Tools*, so öffnet sich die Figure-Umgebung mit Figure Palette, Plot Browser, Property Editor und Figure Toolbar. Unter der Figure Palette sind alle Variablen des Workspace aufgelistet und können damit auch direkt geplottet werden.

Der Workspace Browser stellt eine grafische Darstellung des whos Kommandos dar. Mit dem Befehl who kann man herausfinden, welche Variablen momentan im Workspace gespeichert sind. Um Daten zu sichern kann man die Funktion save verwenden. Mit load kann man gespeicherte Daten in den Workspace laden. Die Tabelle 1 zeigt weitere Funktionen zum Verwalten des Workspace.

Aufgabe 1 (Workspace verwalten) Angenommen Sie möchten die Variablen a und b aus dem Workspace löschen. Wie geht das?

Lösung: Dies können Sie mit clear a und clear b tun. Falls Sie sich Tipparbeit sparen

Funktion	Beschreibung
clear	Löscht Variablen
clear all	Löscht alles
load	Daten laden
save	Daten speichern
who	Zeigt Variablen
whos	Zeigt mehr als who

Tabelle 1: Workspace verwalten

wollen, geben Sie clear a b ein. Hilfe erhalen Sie mit doc clear (help clear). \odot ... \odot

Aufgabe 2 (Workspace verwalten) Schreiben Sie einen Datenfile mit dem Namen data.in, der die folgenden Daten enthält:

1.0	5.0	
2.1	6.3	
3.2	6.9	
3.9	8.1	
5.1	9.1	
1		

Erzeugen Sie im Matlab Workspace eine Matrix A mit diesen Daten. Ändern Sie die zweite Zeile von A zu 2.0 und 6.2 und schreiben Sie die modifizierte Matrix A in einen ASCII-Datenfile mit dem Namen data.out.

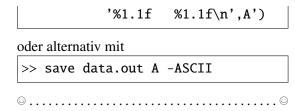
Lösung: Das Einlesen erfolgt mittels

```
>> A = load('data.in');
```

Die Manipulation der Matrix A erfolgt durch

und das Schreiben in eine Datei mit Namen data.out erfolgt mit

```
>> fprintf('data.out',...
```



5.4. Current Directory

Der Current Directory Browser hat gegenüber den Vorgängerversionen an Funktionalität hinzugewonnen und geht weit über das reine Auflisten der Dateien des aktuellen Verzeichnisses hinaus. Durch Anwählen kann direkt in Unterverzeichnisse gesprungen werden.

Unter *View* kann eine Auswahl der aufgelisteten Files nach ihrer Filekennung erfolgen. Files werden durch Doppelklicken ausgeführt. Dah heißt beispielsweise zu m-Files oder Textfiles öffnet sich der Editor, fig-Files werden dargestellt, bei mat-Files werden die Daten geladen und HTML-Files im Web Browser dargestellt.

Das Menü *Find Files* (Fernglassymbol) öffnet eine grafische Umgebung zum Suchen anch Dateien oder Begriffen in Dateien.

Mit filebrowser wird der Current Directory Browser aus dem Command Window heraus aktiviert.

6. Der Help Browser

Matlab verfügt über ein sehr umfangreiches Hilfe- und Dokumentationsfenster, den *Help Browser*, siehe Abbildung 6. Der Help Browser kann aus dem Desktop entweder unter dem Menüpunkt *Help* oder über das ?-Symbol oder

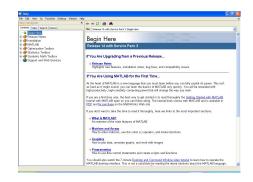


Abbildung 6: Der Help Browser

mit dem Befehl doc geöffnet werden. Der Help Browser besteht aus dem Navigator und auf der rechten Seite aus dem eigentlichen Hilfetext. Im Navigator stehen die Kartenreiter *Contents*, *Index*, *Search* und *Demos* zur Auswahl. Das Docksymbol erlaubt wieder das Anbinden an das Desktop-Fenster.

Unter Contents verbirgt sich eine vollständige Dokumentation aller installierten Produkte bestehend aus den User Guides. Zu den User Guides gehört auch eine vollständige Funktionsübersicht der einzelnen Matlab Funktionen/Kommandos, die stets aus einer Übersicht der erlaubte Syntax, aus einer ausführlichen Beschreibung, gegebenenfalls ergänzenden Hinweisen, einem Beispielteil und einem Verweis auf verwandte Matlab Funktionen/Kommandos sowie in einigen Fällen ergänzende Literaturangaben besteht. Unter Contens findet sich auch das Kapitel *Printable Dokumentation*, das direkt auf die *MathWorks* Internetseite mit den pdf-Dokumenten verweist.

Der Kartenreiter Index stellt ein alphabetisch geordnetes Register zur Verfügung und Search ein Suchfenster. Die Treffer werden im Navigatorteil aufgelistet und durch anklicken im rechten Textfenster dargestellt. Dabei wird unter allen installierten Produkten gesucht. Soll die Suche auf betsimmte Toolboxen oder Blocksets eingeschränkt werden, so kann unter den Preferences im Menü File ein Produktfilter genutzt werden.

Unter Demos werden Beispiele zu verschiedenen Funktionalitäten bereit gehalten.

7. Der Array Editor

Der Array Editor dient dem Visualisieren und interaktiven Editieren von Workspace-Variablen und wird durch Klicken auf die Variable im Workspace geöffnet, siehe Abbildung 7. Alternativ kann der Array Editor

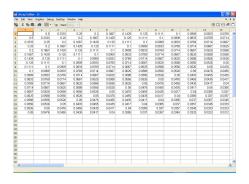


Abbildung 7: Der Array Editor

mittels openvar ('VarName') aus dem Command Window geöffnet werden. VarName ist der Name der Variablen. Im Gegensatz zu älteren Versionen lassen sich ab dem Release 7 auch Zellen- und Strukturvariablen (siehe Abschnitt 74) im Array Editor bearbeiten. Der Array Editor lässt sich wie bereits er-

wähnt am Matlab Desktop andocken. Spalten im Editor lassen sich kopieren, löschen und teilen. Excel-Daten lassen sich unter Windows-Betriebssystemen mit Copy und Paste in den Array Editor kopieren. Das Plot-Symbol erlaubt grafisch unterstützt das Plotten von Daten, dabei steht ein umfangreicher Auswahlkatalog mit Linienplots, Histogrammen, Höhenlinienplots, 3D-Grafiken usw. zur Verfügung. Neue Variablen lassen sich mausgesteuert aus bestehenden Daten erzeigen.

8. Der Editor/Debugger

Der Matlab Editor und Debugger, siehe Abbildung 8 dient dem Schreiben von Matlab

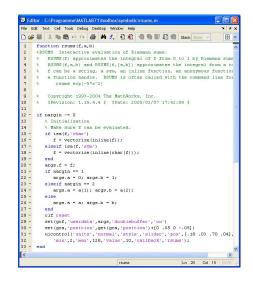


Abbildung 8: Der Editor/Debugger

Scripts und Functions sowie dem grafische Debuggen. Der Editor hebt dabei MATLAB Schlüsselwörter oder Kommentare in unterschiedlichen Farben hervor, erkennt aber auch andere Formate wie beispielsweise HTML. Aufgerufen wird der Editor entweder mit

>> editor Fname

zum Editieren des Files Fname oder aus dem Desktop unter $File \rightarrow New \rightarrow m$ -file oder durch Klicken auf eine Datei im Current Directory Browser.

Unter dem Menüpunkt *Edit* des Editors befindet sich ein *Search and Replace*-Fenster, mit dem Begriffe im lokalen File oder auch außerhalb gesucht und gegebenenfalls durch einen neuen Begriff ersetzt werden können.

Bei der Entwicklung eines Programms treten prinzipiell zwei Fehlerarten auf: Syntax- und Laufzeitfehler. Syntaxfehler sind meist leicht zu beheben. Laufzeitfehler treten bei einem syntaktisch korrekten Programm während der Ausführung auf und zeigen sich entweder in einem Programmabsturz oder in offensichtlich falschen Resultaten. In solchen Fällen kann es notwendig sein, sich den Ablauf des Programms während der Ausführung genauer anzusehen. Diese Aufgabe übernimmt ein sogenannter Debugger, ein Programm, mit dem andere Programme während der Ausführung an bestimmten Stellen, so genannte Breakpoints, unterbrochen und untersucht werden können. Mit Hilfe der Menüeinträge unter Debug oder durch Klick auf das Icon mit dem roten Punkt können Breakpoints gesetzt oder gelöscht werden. Der Debugger kennt verschiedene Arten von Berakpoints, von denen wir auf den Standard-Breakpoint kurz eingehen wollen.

Ein Standard-Breakpoint wird durch einen roten Punkt im Quelltext angezeigt. Er wird in

die Zeile gesetzt, in der der Cursor steht. Bei der Ausführung des Programms erfolgt dann ein automatischer Wechsel in den Editor und das Programm wird an der Stelle des Breakpoints angehalten. Die bis dahin belegten Variablenwerte können anschließend bequem betrachtet werden, indem man mit dem Mauszeiger auf die jeweilige Variable zeigt. Die Werte werden in einem automatisch ausgeklappten Fenster angezeigt (wenn in den Preferences eingestellt). Alternativ können die Variablen auch im Command Window oder mit Hilfe des Kontext-Menüs der reechten Maustaste im Array Editor angezeigt werden. Die Variablenwerte können allerdings nicht nur angezeigt werden. Darüber hinaus ist es möglich, die Werte zur Programmlaufzeit zu verändern, was zu Testzwecken sehr nützlich ist.

Meine Empfehlung ist es, sich mit dem Debugger vetraut zu machen, insbesondere wenn man größere Programme entwickeln will.

Neben dem Debugger gibt es noch weitere Werkzeuge, die die Programmentwicklung unterstützen, den Code-Check mit *M-Lint* und den *Profiler*. Mit dem Profiler sind umfangreiche Laufzeitanalysen möglich, die sicherlich nur dem fortgeschrittenen Anwender von Nutzen sind. Mit dem M-Lint Programm (im Menü *Tools*) kann ein Bericht erstellt werden, in dem der Programmierer zum Beispiel über nicht verwendete Variablen, syntaktische Fehler oder sonstige Ungereimtheiten im Code informiert wird. Dies ist auch für den Anfänger nützlich.

9. Plots

Bilder (Images) oder Abbildungen werden im Figure Window dargestellt. Neben seiner Aufgabe, die Ebene für die grafische Darstellung bereitzustellen, bietet das Figure Window noch weitere Eigenschaften. Unter File besteht die Möglichkeit mit Generate m-File das grafische Layout in einem m-File abzuspeichern und so bei zukünftigen vergleichbaren Aufgaben auf ein vorgefertigtes Plotlayout in Form einer Matlab Funktion zuzugreifen. Unter View wird als wichtigestes Instrument der Property Editor aufgerufen, mit dem interaktiv Eigenschaften der Grafik verändert und sämtliche Eigenschaften mit dem Property Inspector angesprochen werden können. Insert erlaubt das interaktive Einfügen von zum Beispiel Legende und Farbbalken. Unter Tools finden sich Eigenschaften wie Pan zum interaktiven Verschieben von Plotlinien und insbesondere auch das Basic Fitting und Data Statistic Tool.

automatisch erkannt wird, die Zahl der Kopfzeilen vorgegeben werden. Numerische Daten werden dann als Array, Textdaten wie beispielsweise Spaltenüberschriften als Zellenvariablen abgespeichert.

11. Matlab unterbrechen

Mit ctrl-c können Sie Matlab jederzeit unterbrechen.

12. Lange Eingabezeilen

Ist Ihre Eingabezeile lang, so können Sie diese mit drei Punkten beenden ... und in der nächste Zeile fortfahren.

```
>> s = 1+1/2+1/3+1/4+1/5+...
1/6+1/7+1/8+1/9+1/10
s =
2.9290
```

10. Der Import Wizard

Der Import Wizard (Zauberer) lässt sich über den Befehl uiimport, durch Doppelklicken auf einen Datenfile im Current Directory Window oder über File → Import Data öffnen. In diesem Fall wird über einen File Browser der entsprechende Datenfile ausgewählt. Der Import Wizard erlaubt das interaktive Einlesen von Ascu-Files, binären mat-Dateien, Excel-Files, wav-Dateien, Bilddateien oder HDF-Daten, um nur Einige zu nennen. Bei formatierten Daten kann ein geeigneter Spaltenseparator angeklickt werden, und, falls dies nicht

13. Eine Sitzung aufzeichnen

Das Kommando

```
>> diary MeineSitzung
```

sorgt dafür, dass der folgende Bildschirmtext komplett in der Datei MeineSitzung aufgezeichnet wird. Die Aufzeichung können Sie anhalten, wenn Sie diary off eingeben. MeineSitzung ist nur eine Beispieldatei; sie können selbstverständlich jeden zulässigen Dateinamen angeben.

14. Das help-Kommando

Das help-Kommando ist eine einfache Möglichkeit, Hilfe über eine Matlab-Funktion im Command Window zu erhalten. Hierzu gibt man help und den Funktionsnamen, das Kommando oder das Symbol ein.

Das folgende Beispiel zeigt, wie man sich Informationen über die eingebaute MATLAB-Funktion sqrt verschafft.

```
>> help sqrt
SQRT Square root.
SQRT(X) is the square ...
```

Auffallend ist, dass in der Erklärung der Name der sqrt-Funktion groß geschrieben ist. Dies dient lediglich dazu, diesen Namen vom übrigen Text abzusetzen. Der richtige Name ist sqrt, klein geschrieben. Matlab unterscheidet zwischen Groß- und Kleinbuchstaben, deshalb liefert die Eingabe help SQRT eine Fehlermeldung.

```
SQRT.m not found.
```

Matlab-eigene Funktionsnamen bestehen stets aus Kleinbuchstaben. Nur im Hilfetext werden sie groß geschrieben.

Das help-Kommando ist nur geeignet, wenn man den Namen der Funktion kennt, zu der man Hilfe sucht. Was aber, wenn man ihn nicht kennt?

Alle Matlab-Funktionen sind in logische Gruppen (Themen) eingeteilt, und die Matlab-Verzeichnisstruktur basiert auf dieser Eintei-

lung. Gibt man help alleine ein, so wird diese Gruppierung angezeigt.

```
>>> help
HELP topics

matlab\general - General ...
matlab\ops - Operators ...
matlab\lang - Programming ...
matlab\elmat - Elementary ...
usw.
```

Mit help elfun (doc elfun) zum Beispiel erhalten Sie eine Liste aller elementarer mathematische Funktionen in MATLAB.

15. Das doc-Kommando

Eine konfortablere Hilfe erhalten Sie mit dem doc-Kommando. Beispielsweise erhalten Sie mit doc sin eine HTML-Dokumentation über die sin-Funktion. Mit doc elfun bekommen Sie eine Liste aller elementaren mathematischen Funktionen, die in Matlab realisiert sind.

16. Demos

Durch den Aufruf

```
>> demo
```

wird das Hilfe-Fenster geöffnet und Sie können sich Demos über Matlab und seine Toolboxen anschauen. Dort finden Sie in Form von Video-Tutorials die neuen Features der neusten Matlab-Version. Weitere Info erhalten Sie mit doc demo (help demo) oder doc demos

(help demos).

17. Das lookfor-Kommando

Basierend auf einem Schlüsselwort können Sie mit dem lookfor-Kommando nach Funktionen suchen. Dabei wird die erste Zeile des help-Textes jeder Matlab-Funktion zurückgegeben, die das entsprechende Schlüsselwort enthält. Zum Beispiel gibt es in Matlab keine Funktion mit dem Namen inverse. Somit ist die Antwort auf

>> help inverse

folgende:

inverse.m not found.

Aber der Aufruf

>> lookfor inverse

liefert – in Abhängigkeit der installierten Toolboxen – folgendes:

INVHILB Inverse Hilbert matrix.
IPERMUTE Inverse permute array ...

ACOS Inverse cosine.

ACOSH Inverse hyperbolic cosine.

ACOT Inverse cotangent.

ACOTH Inverse hyperbolic ...

ACSC Inverse cosecant.

ACSCH Inverse hyperbolic cosecant.

ASEC Inverse secant.

ASECH Inverse hyperbolic secant.

ASIN Inverse sine.

ASINH Inverse hyperbolic sine.

ATAN Inverse tangent.

ATAN2 Four quadrant inverse ...

ATANH Inverse hyperbolic tangent. ERFCINV Inverse complementary ... ERFINV Inverse error function.

INV Matrix inverse.

PINV Pseudoinverse.

IFFT Inverse discrete Fourier transform.

usw...

Will man, dass alle help-Zeilen durchsucht werden, so muss man im Aufruf die Option -all verwenden:

>> lookfor inverse -all

Falls Sie Hilfe zu einem Begriff suchen und den entsprechenden englischen Ausdruck nicht kennen, verwenden Sie ein Wörterbuch. Auf meiner Homepage finden Sie einen Link zu einem Mathematischen Wörterbuch, das Ihnen weiterhelfen kann, siehe http://www.hs-ulm.de/gramlich.

Aufgabe 3 (lookfor-Kommando) Mit welchem Befehl wird das Command Window (Kommandofenster) gelöscht?

Lösung: Mit dem Aufruf

>> lookfor command

18. Alle Funktionen?

Alle Matlab-Funktionen finden Sie im *Help Browser*. Geben Sie doc ein, dann finden Sie unter Help Navigator Contents Matlab in

Function – Alphabetical List eine alphabetische Auflistung aller Matlab-Funktionen und in Functions – Categorial List alle Funktionen in Gruppen eingeteilt.

system verwenden, so können Sie den *Path Browser* über die Schaltfläche des Command Windows, über Set Path im Menü File oder über das Kommando pathtool öffnen.

19. Wichtige Funktionen?

Wichtige und häufig vorkommende Funktionen habe ich Ihnen im Anhang B abgedruckt.

20. Der Path Browser

MATLAB ermittelt die Art, in der die aufgerufenen Funktionen auszuführen sind, über einen Suchpfad, das ist eine geordnete Liste von Verzeichnissen. Mit dem Kommando path erhält man die aktuelle Liste von Verzeichnissen, in denen MATLAB sucht. Dieser Befehl kann auch dazu verwendet werden, um Verzeichnisse aus dem Suchpfad zu löschen oder anzuhängen. Mit dem *Path Browser* (siehe Abbildung 9) kann der Pfad bequem geändert oder ausgege-

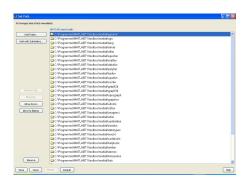


Abbildung 9: Der PathBrowser

ben werden. Falls Sie ein Windows Betriebs-

21. Den Datenträger verwalten

Die Funktionen aus der Tabelle 2 zeigen Möglichkeiten, wie man den Datenträger verwalten kann.

Kommando	Beschreibung
cd	Verzeichnis wechseln
cd oder pwd	Zeigt Verzeichnispfad
delete	Löscht Filename
dir oder ls	Zeigt Files
exist	Überprüft Existenz
matlabroot	Zeigt Matlab-Wurzel
type	Type
what	M, mat und mex-Files
which	Lokalisiert Funktionen

Tabelle 2: Den Datenträger verwalten

22. Wie man weitere Systeminformationen erhält

Die Tabelle 3 zeigt Matlab-Funktionen, die

Funktion	Beschreibung
computer	Typ des Rechners
hostid	Identifikationsnummer
license	Lizenznummer

Tabelle 3: Systeminformationen

man verwenden kann, um Informationen über 25. Einfaches Rechnen den eigenen Computer zu erhalten.

23. Neuigkeiten und Versionen

Weitere Informationen und Demonstrationen zu Matlab finden Sie in Tabelle 4.

Funktion	Beschreibung
bench	Benchmarks
ver	Versionen
version	Versionsnummer
whatsnew	Release Notes

Tabelle 4: Weitere Infos

24. Voreinstellungen

Es gibt mehrere m-Files, die die Voreinstellungen von Matlab festlegen und beim Start ausgeführt werden. Diese Files befinden sich im Unterverzeichnis local im Verzeichnis toolbox. Der File Contents enthält eine Übersicht.

Zur Startzeit führt Matlab automatisch den File matlabre aus. Dieser File setzt die Matlab Pfade, legt die Default-Größe der Figures fest und setzt einige weitere Standardeinstellungen. Veränderungen sollten bei Bedarf nicht in dieser Datei, sondern im startup File durchgeführt werden. Dieser ist dafür gedacht, dass Sie dort eigene Funktionen und Kommandos definieren, die zur Startzeit ausgeführt werden sollen.

Addition +, Subtraktion -, Multiplikation *, Division / und Potenzieren ^ sind grundlegende Rechenoperationen. Hier ein paar Beispiele:

```
>> 3+4
ans =
     7
>> 9/3
ans =
>> 2^8
ans =
   256
```

Matlab nennt das Ergebnis ans (kurz für answer).

> Geben Sie dem Ergebnis keinen Variablennamen (siehe Abschnitt 28), so wählt Matlab standardmäßig (default) den Variablennamen ans.

Schließen Sie die Eingabezeile mit einem Strichpunkt (Semikolon) ab, so rechnet Mat-LAB zwar, unterdrückt aber die Ausgabe:

```
>> 1+2;
```

Im Workspace können Sie erkennen, dass Mat-LAB die Variable ans angelegt hat, den Wert der Variablen ans jedoch nicht ausgibt.

> Ein Semikolon am Ende der Eingabe unterdrückt die Bildschirmausgabe!

Aufgabe 4 (Einfaches Rechnen) Was ist 0^0 in MATLAB?

Lösung: Es ist $0^0 = 1$ in Matlab. $\odot \dots \odot$

26. Welche arithmetische Operation hat Vorang?

Die arithmetischen Operationen von Matlab genügen den gleichen Vorrangsregeln wie in vielen Computersprachen und Taschenrechnern. Grob gesprochen gelten die üblichen Rechenregeln "Punktrechnung vor Strichrechnung". Die Regeln sind in Tabelle 5 aufgezeigt (Eine komplette Tabelle für alle Matlab-Operationen zeigt die Tabelle 6). Für Operatoren, die auf einer Ebene stehen, ist der Vorrang von links nach rechts geregelt. Klammern können immer verwendet werden, um den Vorrang entsprechend abzuändern.

```
>> 2^10/10

ans =

102.4000

>> 2+3*4

ans =

14

>> -2-3*4

ans =

-14

>> 1+2/3*4

ans =

3.6667

>> 1+2/(3*4)

ans =

1.1667
```

Aufgabe 5 (Rechnen) Ermitteln Sie das Ergebnis von

```
>> 3+4/5*6
```

zunächst mit Bleistift und Papier und überprüfen Sie es dann mit MATLAB.

Lösung: Mit den Vorrang-Regeln gilt: $3+4/5 \cdot 6 = 3 + (4/5) \cdot 6 = 3 + (0.8 \cdot 6) = 3 + 4.8 = 7.8$. Hier die Bestätigung in MATLAB:

```
>> 3+4/5*6
ans =
7.8000
```

Aufgabe 6 (Rechnen) Ermitteln Sie das Ergebnis von

zunächst mit Bleistift und Papier und überprüfen Sie es dann mit Matlab.

Lösung: Mit den Vorrang-Regeln gilt: $(48/3) - (3^2) = 16 - 9 = 7$. Hier die Bestätigung in MATLAB:

Priorität Operator

1 (höchste) Potenzieren (^)
2 Unäres Plus (+)
unäres Minus (-)
3 Multiplikation (*)
Division (/)
4 (niedrigste) Addition (+)
Subtraktion (-)

Tabelle 5: Vorrang-Tabelle

Priorität	Operator
1 (höchste)	Transponieren (.')
	Potenzieren (.^)
	kongugiert complex (')
	Matrix-Potenzieren(^)
2	Unäres Plus (+)
	unäres Minus (-)
	logische Negation (~)
3	Multiplikation (.*)
	rechte Division (./)
	linke Division (.\)
	Matrix-Multiplikation (*)
	rechte Matrix-Division (/)
	linke Matrix-Division (∖)
4	Addition (+)
	Subtraktion (-)
5	Doppelpunktoperator (:)
6	Kleiner (<)
	kleiner oder gleich (<=)
	größer (>)
	größer oder gleich (>=)
	gleich (==)
	nicht gleich (~=)
7	Logisches und (&)
8 (niedrigste)	Logisches oder ()

Tabelle 6: Vorrang-Tabelle

27. Zahlen und Formate

Matlab verarbeitet Zahlen in der üblichen Dezimalschreibweise, wobei wahlweise ein Dezimalpunkt und ein positives oder negatives Vorzeichen verwendet werden können. In der wissenschaftlichen Notation bezeichnet der Buchstabe e eine Skalierung um Zehnerpotenzen. Zulässige Zahlen sind zum Beispiel:

4	101	0.0001
9.84757	1.5e-12	8.997
3i	-3.4j	4e3i

Alle Zahlen werden intern im double-Format (Langformat) gemäß der Spezifikation durch die Gleitpunktnorm der IEEE abgespeichert. Matlabs Zahlenausgabe folgt mehreren Regeln. Ist das Ergebnis ganzzahlig, so wird eine ganze Zahl ausgegeben. Wenn das Ergebnis eine reelle Zahl ist, dann gibt MATLAB das Resultat standardmäßig auf 4 Dezimalen gerundet aus. Ist das Matrixelement größer als 10³ oder kleiner als 10^{-3} , so wird es in exponentieller Form auf dem Bildschirm dargestellt. Sollen Zahlen in einem anderen Format ausgegeben werden, so hilft das Matlab-Kommando format, siehe doc format (help format). Die Tabelle 7 gibt mögliche numerische Zahlenformate an.

Kommando	Beispiel: pi	
format short	3.1416	
format long	3.14159265358979	
format bank	3.14	

Tabelle 7: Zahlenausgabe in MATLAB

28. Variablen und Konstanten

Ein Variablenamen muss mit einem Buchstaben beginnen und darf aus maximal 63 Buchstaben, Zahlen und Unterstrichen bestehen (Bis Release 6.1 lag der Wert noch bei 31). namelengthmax liefert die maximale Länge (63 Zeichen), die zur Unterscheidung eines Namens (Variablen, Dateien, usw.) erlaubt ist, zurück.

```
>> namelengthmax
ans =
63
```

Umlaute sind nicht erlaubt! Erlaubt sind zum Beispiel

```
MeineVariable Anna x1 X3 z23c1 My_Var
```

Nicht erlaubt sind

```
Meine-Variable 2Var \$2 &x
```

Mit der Funktion isvarname können Sie die Gültigkeit eines Variablenamens überprüfen.

```
>> isvarname('My_Var')
ans =
    1
>> isvarname('2Var')
ans =
    0
```

1 (wahr) wird zurückgegeben, falls ein gültiger Variablenname vorliegt, ansonsten 0 (falsch). Für weitere Infos, siehe auch Abschnitt 46.

Aufgabe 7 (Variablennamen) Wieviel verschiedene Matlab Variablenamen stehen in folgender Zeile?

anna	ANNA	anNa	aNna	anna

Lösung: Vier verschiedene Variablennamen.

Achten Sie in Matlab unbedingt auf Groß-und Kleinschreibung!

Darüber hinaus gibt es vordefinierte (vorbelegte) Variablen, siehe Tabelle 8. Achtung! Sie

Spezielle Variable	Bedeutung
ans	Resultat (Default)
computer	Identifiziert
eps	Maschinengenauigkeit
i	Imaginäre Einheit
Inf, inf	Infinity
j	Imaginäre Einheit
NaN	Not-a-Number
pi	Kreiszahl $\pi \approx 3.14$

Tabelle 8: Spezielle Variablen

können diese spezielle Variablen überschreiben; vermeiden Sie dies aber, wenn immer möglich.

Vermeiden Sie das Überschreiben spezieller Variablen!

Weitere Infos unter doc elmat (help elmat) bzw. doc lang (help lang).

29. Komplexe Zahlen

Komplexe Zahlen können wie folgt eingegeben werden:

```
3.0000 + 4.0000i
z2 =
1.0000 - 3.0000i
```

Wie Sie sehen kann die imaginäre Zahl i mit $i^2 = -1$ als i oder j eingegeben werden, wobei das Multiplikationszeichen * wahlweise verwendet werden kann. Die Operatoren +, -, * und / sind bei Verwendung komplexer Zahlen die komplexe Addition, komplexe Subtraktion, komplexe Multiplikation und die komplexe Division. Hier zwei Beispiele:

```
>> z1+z2

ans =

    4.0000 + 1.0000i

>> z1*z2

ans =

    15.0000 - 5.0000i
```

Die Wurzel der komplexen Zahl 1 + 2i berechnet man mit der Funktion sqrt

```
>> sqrt(1+2i)
ans =
1.2720 + 0.7862i
```

Die konjugiert komplexe Zahl zu 1 + 3i ist

```
>> conj(1+3i)
ans =
1.0000 - 3.0000i
```

Den Absolutbetrag und Phasenwinkel einer komplexen Zahl z kann mit abs bzw angle berechnet werden. Weitere Funktionen, die beim Rechnen mit komplexen Zahlen von Bedeutung sein können, siehe doc elfun.

Aufgabe 8 (Komplexe Zahlen) Gegeben ist die komplexe Zahl z = 4i/(1 + i). Wie lautet die

zu z konjugiert komplexe Zahl? Wie groß sind Absolutbetrag und Phasenwinkel von z? Lösung:

```
>> z = 4i/(1+i);

>> conj(z)

ans =

    2.0000 - 2.0000i

>> phi = angle(z)*180/pi

phi =

    45

>> a = abs(z)

a =

    2.8284
```

Aufgabe 9 (Komplexe Zahlen) Finden Sie heraus, wie man den Real- und Imaginärteil einer komplexen Zahl berechnet. Berechnen Sie dann den Real- und Imaginärteil von z = (3 + 2i)/(1 - i).

Lösung: Mit lookfor complex findet man die Funktionen real und imag. Damit gilt

```
>> z = (3+2i)/(1-i);
>> real(z), imag(z)
ans =
0.5000
ans =
2.5000
```

30. IEEE-Arithmetik und double

Standardmäßig ist in MATLAB ist der IEEE-Standard 754 mit doppelt genauer Gleitpunktarithmetik realisiert. So wird jede Zahl default-

mäßig in den Datentyp double konvertiert. Jede Zahl vom Datentyp double belegt einen Speicherplatz von 64 Bits. Von Null verschiedene positive Zahlen liegen daher ungefähr zwischen 10⁻³⁰⁸ und 10⁺³⁰⁸ und die relative Rechengenauigkeit (unit roundoff) ist $2^{-53} \approx$ $1.11 \cdot 10^{-16}$. Das wesentliche Merkmal der relativen Rechengenauigkeit ist, dass sie eine relative Fehlerschranke für das Konvertieren einer reellen Zahl in eine Gleitpunktdarstellung und auch eine Schranke für den relativen Fehler darstellt, der entsteht, wenn man zwei Gleitpunktzahlen addiert, subtrahiert, multipliziert oder dividiert, oder die Quadratwurzel aus einer Gleitpunktzahl zieht. Grob gesagt: MATLAB speichert und führt elementare Rechenoperationen mit einer Genauigkeit von ungefähr 16 Dezimalstellen durch.

Die Funktion eps (machine precision) gibt den Abstand von 1.0 zur nächst größeren Gleitpunktzahl zurück.

```
>> eps
ans =
2.2204e-016
```

Dieser Abstand ist 2^{-52} , also zweimal der relativen Rechengenauigkeit. Da Matlab den Ie-EE-Standard realisiert, erzeugt jede Rechnung eine Gleitpunktzahl, womöglich aber in einem besonderen Format. Ist das Ergebnis einer Berechnung größer als realmax, dann tritt ein Overflow ein und das Resultat ist Inf, was für unendlich (infinity) steht. Ist das Resultat kleiner als -realmin, so kommt -inf heraus.

```
>> realmax
ans =
1.7977e+308
```

```
>> -1.1*realmax
ans =
   -Inf
>> 1.2*realmax
ans =
   Inf
```

Ist eine Rechnung mathematisch nicht definiert, so ist das Resultat NaN, was für Not a Number steht. Die Ausdrücke 0/0, inf/inf und 0*inf sind von dieser Art.

```
>> 0/0
Warning: Divide by zero.
ans =
    NaN
>> inf/inf
ans =
    NaN
>> 0*inf
ans =
    NaN
```

Hat man einmal ein NaN erzeugt, so pflanzt sich dies im Laufe der Rechnung fort.

```
>> 3+NaN
ans =
NaN
>> NaN-NaN
ans =
NaN
>> 0*NaN
ans =
NaN
```

Die Funktion realmin gibt die kleinste positive normalisierte Gleitpunktzahl zurück. Jede Rechnung, deren Ergebnis kleiner als realmin ist, erzeugt einen Underflow und wird auf Null

gesetzt, wenn sie kleiner als eps*realmin ist oder erzeugt eine nichtnormale Zahl (subnormal number) mit führendem Bit 0 in der Mantisse.

```
>> realmin
ans =
    2.2251e-308
>> realmin*eps
ans =
    4.9407e-324
>> realmin*eps/2
ans =
    0
```

Die Funktion computer gibt den Computer-Typ zurück, auf dem Matlab läuft. Der Rechner auf dem diese Zeilen und die Matlab-Codes geschrieben werden, produziert folgende Ausgabe

```
>> computer
ans =
PCWIN
```

Aufgabe 10 (Arithmetik) Berechnen Sie:

- (a) e^{700}
- (b) e^{710}

Beschreiben Sie die Resultate.

Lösung:

In (b) wird als Inf ausgegeben, da das Ergebnis die größte positive Gleitkommazahl in Matlab übersteigt. Diese Zahl ist

```
>> realmax
ans =
1.7977e+308
```

31. Nicht double-Datentypen

Außer double werden in Matlab auch noch andere Datentypen zur Verfügung gestellt, um Zahlen zu speichern und mit diesen zu rechnen. Diese sind

- single
- int8 und uint8
- int16 und uint16
- int32 und uint32

Diese Datentypen sind insbesondere dann vorteilhaft, wenn Speicherplatz gespart werden soll, also zum Beispiel wenn Bilder gespeichert und verarbeitet werden sollen (Bildverarbeitung).

Ganzzahlige Datentypen können ganze Zahlen in einem bestimmten Bereich speichern. Zum Beispiel ist uint8 ein ganzzahliger Datentyp der im Speicher acht Bits benötigt, um Zahlen im Bereich von 0 bis 255 zu spreichern; $2^8 = 256$. Die Tabelle 9 zeigt die ganzzahligen Datentypen mit Speicherbedarf und Wertebereich. Die folgende Anweisung erzeugt die Variable x vom Datentyp uint8 und ordnet ihr die Zahl 7 zu.

Klasse	Speicherbedarf	Wertebereich
uint8	8 Bit	0 bis $2^8 - 1$
int8	8 Bit	-2^7 bis $2^7 - 1$
uint16	16 Bit	0 bis $2^{16} - 1$
int16	16 Bit	-2^{15} bis $2^{15} - 1$
int32	32 Bit	-2^{31} bis $2^{31} - 1$
uint32	32 Bit	0 bis $2^{32} - 1$

Tabelle 9: Ganzzahlige Datentypen

```
>> x = uint8(7)
x = 7
```

Hier ein paar Rechnungen:

```
>> x+4
ans =
   11
>> x+pi
ans =
   10
>> x+253
ans =
  255
>> x+254
ans =
  255
>> x-8
ans =
    0
>> x-9
ans =
```

Die Rechnungen zeigen mehrere Eigenschaften. Das Ergebnis ist stets vom Datentyp

uint8 auch dann, wenn eine double-Zahl hinzuaddiert wird. Diese wird zuvor gerundet. Wird die größte Zahl 255 überschritten, so wird das Ergebnis gleich dieser größten Zahl gesetzt. Analog verhält es sich, wenn das Ergebnis kleiner als Null ist.

Aufgabe 11 (int8-Rechnungen) Finden Sie heraus, wie eine double-Zahl auf den Datentyp int8 gerundet wird.

Lösung: Die Rechnungen

zeigen die Antwort. Matlab rundet auf die nächste ganze Zahl. Liegt die Zahl in der Mitte, so wird aufgerundet, falls die Zahl positiv und abgerundet, falls die Zahl negativ ist. $\odot \ldots \odot$

Mit den Funktionen intmax und intmin können Sie die größte bzw. kleinste Zahl des jeweiligen Datentyps ermitteln.

```
>> intmax('int32')
ans =
2147483647
```

Um eine Zahl mit dem Datentyp single zu speichern, braucht man nur halb so viel Spei-

cher wie mit dem Datentyp double. Dies zei- 32.1. Keine Deklaration notwendig gen die folgenden Zeilen.

```
>> a = single(5);
>> b = 5;
>> whos
Name Size Bytes
                   Class
     1x1
                   single array
h
                8
                   double array
     1 \times 1
```

single-Zahlen können mit double-Zahlen verknüpft werden. Das Ergebnis ist single.

```
>> single(4.1)+double(3)
ans =
    7.1000
>> whos
Name Size Bytes
                 Class
ans
     1x1
                  single array
```

Ganzahlige und single-Datentypen können nicht verknüpft werden.

Siehe doc datatypes, [16], [19] und Abschnitt 74 für weitere Informationen.

32. Merkmale von Matlab

Matlab verfügt über drei wesentliche Merkmale, die das System von anderen modernen Programmierumgebungen unterscheidet. Wir führen sie in diesem Anschnitt ein und erklären sie in den folgenden Abschnitten ausführlicher.

Variablen müssen in Matlab nicht deklariert werden, bevor sie verwendet werdn. Dies gilt sowohl für Arrays als auch für Skalare. Darüberhinaus werden die Dimensionen der Arrays bei Bedarf automatisch vergrößert. Durch

wird ein Zeilenvektor x mit drei Nullkoordinaten angelegt und durch

zu einem Vektor der Länge 5 erweitert. An jetzt gibt es einen Zeilnvektor x der Länge 5.

32.2. Variable Argumentenliste

Matlab verfügt über eine große Sammlung von Funktionen. Diese können kein Eingabeargument oder mehrere Eingabeargumente und kein Ausgabeargument oder mehrere Ausgabeargumente haben. Zwsichen Ein- und Ausgabeargument wird deutlich unterschieden. Die Eingabeargumente stehen rechts vom Funktionsnamen in runden Klammern und die Ausgabeargumente stehen links vom Funktionsnamen in eckigen Klammern.

In diesem Beispiel sind a und b sind Ausgabeund x, y, z die Eingabeargumente. Sowohl die Eingabe- als auch die Ausgabeargumente sind variabel, so dass unter Umständen nur ein Teil der Argumente spezifiziert werden muss. Wir zeigen dies an ein paar Beispielen.

Die Funktion norm berechnet die Euklidische Norm (gewöhnliche Länge, 2-Norm) eines Vektors:

```
>> x = [3 4];
>> norm(x)
ans = 5
```

Eine andere Norm, die sogenannte 1-Norm (Summe der Beträge der Koordinaten), kann ebenfalls mit der Funktion norm berechnet werden, indem 1 als zweites Eingabeargument angegeben wird.

```
>> norm(x,1)
ans = 7
```

Wir sehen. Wird kein zweites Eingabeargument angegeben, so wird standardmäßig die 2-Norm verwendet. Die Funktion max hat mehrere Ausgabeargumente. Mit einem Eingabeund einem Ausgabeargument gibt max die größte Koordinate eines Vektors zurück.

```
>> max(x)
ans =
4
```

Gibt man ein zweites Ausgabeargument an, so wird zusätzlich der Koordinatenindex der größte Koordinate mit ausgegeben.

```
>> [m,k] = max(x)
m =
4
k =
```

2

32.3. Komplexe Arrays und Arithmetik

Der fundamentale Datentyp ist in MATLAB ein mehrdimensionales Array bestehend aus komplexen Zahlen, wobei Real- und Imaginärteil Gleitpunktzahlen sind, die doppelt genau gespeichert sind. Wichtige Spezialfälle sind Matrizen (zweidimensionale Arrays), Vektoren und Skalare. Die meisten Berechnungen werden in Gleitpunktarithmetik durchgeführt und das in komplexer Arithmetik, wenn Daten komplex auftreten. Dies steht im Unterschied zu C/C++ und Java, wo nur reelle Zahlen und reelle Arithmetik unterstützt werden.

MATLAB verfügt jedoch auch über ganzzahlige Datentypen (siehe Abschnitte 31 und 74), die jedoch hauptsächlich aus Speicherplatzeffizienzgründen anstatt zu Berechnungen verwendet werden.

33. Mathematische Funktionen

MATLAB verfügt über viele mathematische Funktionen, siehe doc elfun (help elfun) für grundlegende Funktionen und doc specfun (help specfun) für spezielle Funktionen. Zu den grundlegenden Funktionen gehören die

- trigonometrische Funktionen,
- Exponentialfunktionen,
- Logarithmusfunktionen,
- hyperbolische Funktionen,

- Potenzfunktionen,
- Wurzelfunktionen,
- Restbildungsfunktionen,
- Vorzeichenfunktionen,
- Rundungsfunktionen.

Unter den speziellen Funktionen findet man zum Beispiel die

- Besselfunktionen,
- Hankelfunktionen,
- Fehlerintegrafunktion,
- Exponentialintegralfunktion,
- Gammafunktion.

Die Tabelle 10 zeigt trigonometrische Funk-

Funktion	Beschreibung
acos	Inverser Kosinus
asec	Inverser Sekans
acsc	inverser Kosekans
asin	Inverser Sinus
atan	Inverser Tangens
cos	Kosinus
cot	Kotangens
csc	Kosekans
sec	Sekans
sin	Sinus
tan	Tangens

Tabelle 10: Trigonometrische Funktionen

tionen. Alle trigonometrische Funktionen (mit der Ausnahme atan2) akzeptieren als Argumente komplexe Arrays, die punktweise ausgewertet werden. Die Arumente werden im Bogenmaß (Radiant, rad) erwartet. Für Berechnungen in Grad (Gradmaß, °) dienen die mit einem d am Ende des Namens ergänzten

trigonometrischen Funktionen: acosd, cosd, asind, sind, usw.

Die folgenden Zeilen bestätigen die Ergebnisse $cos(\pi/2) = 0$ und $cos(90^\circ) = 0$ in Matlab.

Aufgabe 12 (Trigonometrische F.) Bestätigen Sie die Tabelle

und die Tabelle

$$\begin{array}{c|ccccc} \phi \text{ (in Grad)} & 90 & 180 & 270 & 360 \\ \hline \cos \phi & 0 & -1 & 0 & 1 \\ \end{array}$$

Lösung: Hier die Bestätigung:

```
>> x = pi*[1/2 1 3/2 2];

>> cos(x)

ans =

0.0000 -1.0000 -0.0000 1.0000

>> phi = 180*[1/2 1 3/2 2];

>> cosd(phi)

ans =

0 -1 0 1
```

Andere Funktionen sind zum Beispiel sqrt, exp oder log.

Aufgabe 13 (Logarithmen) Finden Sie heraus, welche Logarithmusfunktionen Ihnen in Mat-LAB zur Verfügung stehen?

Mathematisches Symbol	MATLAB
$ln = log_e$	log
$lb = log_2$	log2
$lg = log_{10}$	log10

Tabelle 11: Logarithmusfunktionen

33.1. Rundungsfunktionen

In Matlab gibt es die folgenden Rundungsfunktionen:

- ceil(x) Rundet x zur nächsten ganzen Zahl auf (Runden nach Unendlich).
- fix(x) Wählt von x den ganzzahligen Anteil (Runden nach Null).
- floor(x) Rundet x zur nächsten ganzen Zahl ab (Runden nach minus Unendlich).
- round(x) Rundet x zur nächsten ganzen Zahl.

Aufgabe 14 (Rundungsfunktionen) Berechnen Sie die folgenden Ausdrücke per Hand und überprüfen Sie Ihre Ergebnisse mit MATLAB.

- (a) round(-2.6)
- (b) fix(-2.6)

- (c) floor(-2.6)
- (d) ceil(-2.6)
- (e) floor(ceil(-2.6))

Zeichnen Sie die Funktionen im Intervall [-3,3]!

Lösung:

```
>> round(-2.6), fix(-2.6),
ans =
        -3
ans =
        -2
>> floor(-2.6), ceil(-2.6),
ans =
        -3
ans =
        -2
>> floor(ceil(-2.6))
ans =
        -2
```

Wir zeichnen die Funktion round.

```
>> x = linspace(-3,3,1000);
>> y = round(x);
>> plot(x,y)
```

33.2. Verwendung mathematischer Funktionen in MATLAB

Liegt Ihnen der Funktionsterm einer mathematischen Funktion vor, so können Sie diesen Matlab auf zwei Arten bekannt machen:

 Sie definieren den Funktionsterm in einer function als m-File, siehe Abschnitt 44.
 Zum Beispiel erklärt man die quadratische Funktion $f(x) = x^2$, $x \in \mathbb{R}$ durch folgenden function-File f.m:

```
function y = f(x)
y = x.^2;
```

• Sie definieren den Funktionsterm in der Kommandozeile durch einen String und Vorstellung eines @-Zeichens, siehe Abschnitt 39. Zum Beispiel definiert man die quadratische Funktion $f(x) = x^2$, $x \in \mathbb{R}$ durch

```
>> f = @(x) x.^2;
```

Achten Sie darauf, Funktionsterme gleich in vektorieller Form zu definieren, da diese meist vektoriell ausgewertet werden bzw. weil Funktionen wie zum Beispiel quad (eine Funktion zur numerischen Integration, siehe Abschnitt 61) dies auch so verlangen. Funktionen mit mehreren Variablen können ebenso erklärt werden.

Weitere Informationen finden Sie mit doc function und doc function_handle (help function bzw. help function_handle). Für symbolische Funktionen, siehe Abschnitt 67.

Aufgabe 15 (Funktionen) Definieren Sie in Matlab die Funktion $f(x, y) = -xye^{-2(x^2+y^2)}$, $(x, y) \in \mathbb{R}^2$ und werten Sie diese an den Stellen (0, 0) und (1, 1) aus.

Lösung: Dies kann man zum Beispiel wie folgt erreichen:

```
>> f = @(x,y) -x.*y.*...
exp(-2*(x.^2+y.^2));
>> f(0,0), f(1,1)
ans =
```


34. Vektoren

Vektoren (Zeilen- oder Spaltenvektoren) sind spezielle Matrizen. Zeilenvektoren sind (1, n)-Matrizen und Spaltenvektoren sind (m, 1)-Matrizen. Aussagen über Matrizen sind demnach auch für Vektoren interessant. Wir verweisen auf Abschnitt 36; dort werden Matrizen und somit auch Vektoren ausführlich behandelt. Matrizen (und somit Vektoren) sind grundlegende Bausteine von Matlab.

Nehmen wir an, es sollen die Quadratwurzeln der Zahlen 0, 2, 4, 6, 8, 10 berechnet werden. Hier kommt uns Matlab sehr entgegen. Wir fassen die *x*-Werte zu einem Zeilenvektor (eindimesionales Array, Liste) zusammen:

Vektoren werden in eckigen Klammern (brackets) eingegeben. Die einzelenen Elemente (Koordinaten) des Vektors sind dabei durch Leerzeichen oder Kommas zu trennen. Die Funktionswerte für die einzelnen Werte von *x* müssen nun nicht einzeln berechnet werden, sondern können alle auf einmal mit

```
>> y = sqrt(x)
y =
0 1.4142 2.0000 2.4495
2.8284
```

erhalten werden. Die Matlab-Funktion sgrt angewendet auf einen Vektor x gibt nämlich die Anweisung: Nimm von jedem Element von x die Wurzel und schreib das Ergebnis als entsprechendes Element eines neuen Vektors (hier y genannt). Man spricht von vektorielle Auswertung oder allgemein: In Matlab kann man vektoriell programmieren! Diese einfache Art, einen Befehl (hier das Wurzelziehen) elementweise anzuwenden, ist eine große Stärke von Matlab. Sie ermöglicht eine rasche Verarbeitung von großen Datenmengen. Der Vektor x kann kürzer eingegeben werden: x = 0:2:8. Die Schreibweise bedeutet: Beginne mit 0 und zähle 2 dazu, dann zähle wieder 2 dazu, usw. bis die Grenze 8 erreicht ist. Diese Anweisung ist bei der Eingabe von Vektoren mit vielen Elementen sehr nützlich.

Aufgabe 16 (Vektoren erzeugen) Erzeugen Sie einen Zeilenvektor mit der ersten Koordinate 29 und der letzten 1, wobei sich die Koodinaten absteigend um 2 unterscheiden sollen.

Lösung: Dies kann man wie folgt erreichen:

```
>> x = 29:-2:0
x =
Columns 1 through 11
29 27 25 usw.
Columns 12 through 15
7 5 3 1
```

Ist die Schrittweite gleich 1, so kann man die Angabe der Schrittweite weglassen. Für weiter Informationen zu diesem Doppelpunktoperator, siehe doc colon (help colon). Siehe auch doc ops (help ops).

Aufgabe 17 (Vektoren erzeugen) Geben Sie die Vektoren a und b mit den Elementen $-10, -8, \ldots, 10$ und $10, 9, 8, \ldots$ mit kurzen Anweisungen ein.

Lösung: Das geht wie folgt:

```
>> a = -10:2:10; b = 10:-1:0;
```

Aufgabe 18 (Vektoren erzeugen) Erzeugen Sie einen Vektor y, der die Funktionswerte des natürlichen Logarithmus an den Stellen 1, 3, 5, 7 enthält. Was ist y(1)?

Lösung: Das geht wie folgt:

```
>> x = 1:2:7;
>> y = log(x)
y =
0 1.0986 1.6094 1.9459
```

y(1) ist die erste Koordinate des Vektors y, also y(1)=0. \odot

Mit der Funktion linspace können Vektoren erzeugt werden, indem man den Wert der ersten, letzten, sowie die Anzahl der Komponenten (drittes Argument im Aufruf) vorgibt. Mit dem Doppelpunktoperator : ist es dagegen möglich, die Schrittweite direkt anzugeben, nicht aber die Anzahl der Koordinaten. Sind a und b reelle Zahlen, dann erzeugt x =linspace(a,b,n) einen Zeilenvektor x der Länge n mit den Koordinaten $x_k = a + (b - a) \cdot \frac{k-1}{n-1}$, wobei $k = 1, \ldots, n$ ist.

```
>> x = linspace(1,2,3)

x =

1.0000 1.5000 2.0000

>> x = linspace(10,40,4)

x =
```

```
10 20 30 40
>> x = linspace(1,1.2,3)
x =
    1.0000   1.1000   1.2000
>> x = linspace(0,1,3)
x =
    0   0.5000   1.0000
```

Mit der Funktion logspace ist es möglich, einen Vektor zu erzeugen, dessen Komponenten logarithmischen Abstand haben. Allgemein ist $\mathbf{x} = \text{logspace}(\mathbf{a}, \mathbf{b}, \mathbf{n})$ äquivalent zu $x_k = 10^{a+(b-a)\cdot\frac{k-1}{n-1}}$, wobei $k = 1, \ldots, n$ und a, b reelle Zahlen sind.

```
>> x = logspace(-2,2,4)

x =

0.0100   0.2154   4.6416   100.0000

>> x = logspace(1,3,3)

x =

10   100   1000
```

Zusammenfassend gilt: Vektoren können auf drei Arten erzeugt werden:

- Wir geben die Koordinaten explizit ein.
- Wir verwenden eingebaute Matlab-Operatoren oder Funktionen, wie zum Beispiel: : oder linspace.
- Wir lesen einen entsprechenden Datenfile ein, siehe Abschnitt 38.3.

35. Vektorenoperationen

Vektoren sind spezielle Matrizen. Für Vektorenoperationen siehe die Abschnitte 36.3 und 48.

36. Matrizen

Ein rechteckiges Zahlenschema mit m Zeilen und n Spalten heißt (m,n)-Matrix. Vektoren sind spezielle Matrizen. Ist n=1, so liegt ein Spaltenvektor vor und ist m=1, so handelt es sich um einen Zeilenvektor. Skalare sind (1,1)-Matrizen. Es ist üblich, eine Matrix in eckige oder runde Klammern zu setzen; wir wählen die eckige Schreibweise.

Matrizen sind in Matlab (und darüberhinaus) fundamental. In der Version 3 von Matlab und in früheren Versionen war die (komplexe) Matrix überhaupt der einzige Datentyp. Nun gibt es in Matlab aber mehrere Datentypen, siehe Abschnitt 74 und Matrizen sind spezielle mehrdimensionale numerische Arrays, siehe Abschnitt 73. Liegt eine Matrix vor und stehen nicht mathematische Operationen im Vordergrund, so nennt man eine Matrix ein zweidimensionales Array. Dann steht das datentechnische, effiziente Arbeiten im Vordergrund. Steht die Mathematik im Vordergrund, dann siehe auch die Abschnitte zur Linearen Algebra 48 und 50.

Mit doc elmat (help elmat) erhalten Sie viele Infos rund um Matrizen; unter doc ops (help ops) sind die Operatoren zu finden.

36.1. Matrizen erzeugen

Es gibt mehrere Möglichkeiten, Matrizen in Matlab zu erzeugen.

Ist m = 2 und n = 3, so liegt eine (2, 3)-Matrix

vor, zum Beispiel

$$A = \left[\begin{array}{ccc} 1 & \sqrt{2} & -2 \\ 7 & -3 & \pi \end{array} \right].$$

Die Matrix A können wir nun Zeile für Zeile wie folgt in Matlaß eingeben

Die Zeilen werden durch ein Semikolon und die Spalten durch ein Leerzeichen getrennt. Spalten können auch durch ein Komma getrennt werden. Besteht eine Matrix nur aus einer Zeile, so liegt eine Zeilenmatrix bzw. ein Zeilenvektor vor. Analog spricht man von einer Spaltenmatrix bzw. von einem Spaltenvektor, wenn die Matrix nur eine Spalte hat. Eine Zeilenmatrix hat die Größe (1, n) und eine Spaltenmatrix (m, 1).

Will man nun einzelne Elemente der Matrix A ändern, so kann dies auf zwei Arten geschehen. Die Anweisung A(1,3) = 5 ändert zum Beispiel das Elemente $a_{13} = -2$ der Matrix A zu $a_{13} = 5$ ab. Eine zweite Möglichkeit dies zu tun besteht darin, den *Workspace Browser* zu verwenden und das Symbol für die Matrix A anzuklicken. Der Array-Editor wird geöffnet und Sie können die Elemente interaktiv ändern.

Besteht eine Matrix nur aus einer Zeile, so liegt eine Zeilenmatrix bzw. ein Zeilenvektor vor. Einen Zeilenvektor mit Zahlen gleichen Abstands kann man zum Beispiel wie folgt erzeugen

Die Schrittweite muss nicht notwendigerweise Eins sein. Im folgenden Beispiel ist die Schrittweite 0.2:

```
>> x = 1.3:0.2:1.8
x =
1.3000 1.5000 1.7000
```

Mit der Funktion size können Sie stets die Größe einer Matrix bestimmen.

```
>> size(A)
ans =
2 3
```

Macht man bei einer Matrix A aus den Zeilen Spalten und aus den Spalten Zeile, so entsteht die transponierte Matrix A^{T} . In Matlab erreicht man dies mit dem '-Operator.

Nützliche und häufig verwendete Matrizen stellt Matlab als eingebaute Funktionen zur Verfügung, siehe Tabelle 12. Um sie zu erzeugen, muss man nur die Größe angeben. Die Funktion ones erzeugt eine Matrix mit lauter Einsen.

Funktion	Bedeutung
zeros	Nullmatrix
ones	Einsmatrix
eye	Einheitsmatrix
rand	Zufallsmatrix
randn	Zufallsmatrix

Tabelle 12: Elementare Matrizen

1 1 1

Die Nullmatrix wird mit der Funktion zeros erzeugt.

Eine (n,n)-Matrix heißt quadratische Matrix; dann genügt ein Argument, um zum Beispiel die (3,3)-Einheitsmatrix mit der Funktion eye zu erzeugen.

Aufgabe 19 (Symmetrische Matrizen) Erzeugen Sie eine 10×10 symmetrische Matrix S mit Zufallswerten zwischen 0 und 8! *Lösung:*

Diagonalmatrizen werden mit der Funktion diag erzeugt.

Aufgabe 20 (Diagonalmatrizen) Erzeugen Sie eine Diagonalmatrix mit 1, 2, 3, 4 und 5 auf der Diagonalen!

Lösung:

Aufgabe 21 (Matrizen) Wir betrachten die folgende Matrix *A*:

$$A = \begin{bmatrix} 5.5 & 0.4 & 3.1 \\ 9.4 & 5.5 & 3.3 \\ -0.3 & 4.6 & -4.3 \\ 0.4 & -4.6 & 9.0 \\ 5.0 & 5.5 & 7.7 \end{bmatrix}$$

Wie kann man in Matlab die Größe von A bestimmen? Geben Sie die Ordnung (Größe) dieser Matrix an!

Lösung: Die Ordnung der Matrix A ist (5,3).

Außer den elementaren Matrizen stellt Mat-LAB weitere spezielle Matrizen zur Verfügung, siehe Tabelle 13. Diese Matrizen haben inter-

Funktion	Bedeutung
compan	Begleitmatrix
gallery	Hіднамѕ-Testmatrizen
hadamard	Hadamard-Matrix
hankel	Hankel-Matrix
hilb	Hilbert-Matrix
invhilb	Inverse Hilbert-Matrix
magic	Magische Quadrate
pascal	Pascal-Matrix
rosser	Symmetrische Testmatrizen
toeplitz	Töplitz-Matrix
vander	Vandermonde-Matrix
wilkinson	Wilkinson-Matrix

Tabelle 13: Spezielle Matrizen

essante Eigenschaften und werden häufig verwendet, um Algorithmen zu testen. Eine besonders interessante Matrix ist die Hilbert-Matrix, deren (i, j)-tes Element 1/(i+j-1) ist. Diese Matrix kann durch hilb erzeugt werden; ihre Inverse mit invhilb. Die Funktion magic erzeugt magische Quadrate.

Mit der load Funktion besteht eine weitere Möglichkeit, Matrizen zu erzeugen.

36.2. Der Doppelpunkt

Um effizient mit Matrizen arbeiten zu können, steht der Doppelpunkt : zur Vefügung.

Mit ihm lassen sich Vektoren effizient erzeugen. Sind i und j zwei ganze Zahlen, so wird mit i: j ein Zeilenvektor bestehend aus ganzen Zahlen von i nach j mit Schrittweite 1 erzeugt. Soll die Schrittweite s sein, so ist die Syntax: i:s:j. Diese Konstruktion funktioniert auch, wenn die Zahlen nicht notwendig ganz sind. Wir geben Beispiele.

Die folgenden Beispiele zeigen, wie man aus einer gegebenen Matrix, einzelne Elemente, ganze Zeilen oder Spalten sowie Untermatrizen anspricht. Zunächst wird von der Matrix

$$A = \left[\begin{array}{cc} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{array} \right]$$

das Element a_{31} in der dritten Zeile und ersten Spalte zurückgegeben, dann die zweite Zeile, danach die erste Spalte und schließlich eine (2, 2)-Untermatrix.

Mit dem Doppelpunkt kann man eine Matrix in einen "langen" Vektor umwandeln. Die Elemente werden spaltenweise aneinandergefügt.

Aus einem Vektor lässt sich ebenfalls eine Matrix mit spezieller Struktur konstruieren.

Die folgenden Anweisungen zeigen, wie man aus einer Matrix die zweite Spalte streicht. Hierzu benutzt man die leere Matrix []. Vollkommen analog kann man Zeilen streichen.

Wir fassen zusammen:

- A(z,s): Adressiert eine (Unter-) Matrix von A, die durch den Indexvektor z für die Zeilen und durch den Indexvektor s für die Spalten bestimmt ist.
- A(z,:): Adressiert eine (Unter-) Matrix von A, die durch den Indexvektor z für die Zeilen und durch alle Spalten bestimmt ist.
- A(:,s): Adressiert eine (Unter-) Matrix von A, die durch alle Zeilen und durch den Indexvektor s für die Spalten bestimmt ist.
- A(:): Adressiert alle Matrixelemente von A als einen (langen) Spaltenvektor, indem alle Spalten aneinandergefügt werden.
- A(i): Adressiert eine (Unter-) Matrix von A, die durch den einzigen Indexvektor i bestimmt ist, indem A als (langer) Spaltenvektor interpretiert wird.

Aufgabe 22 (Doppelpunkt) Gegeben sei die folgende Matrix *A*:

$$\mathbf{A} = \begin{bmatrix} 5.5 & 0.4 & 3.1 \\ 9.4 & 5.5 & 3.3 \\ -0.3 & 4.6 & -4.3 \\ 0.4 & -4.6 & 9.0 \\ 5.0 & 5.5 & 7.7 \end{bmatrix}$$

Was ist A(:,2), A(3,:), A(4:5,2:3)? Überprüfen Sie Ihr Resultat in MATLAB.

Lösung: A(:,2) ist die zweite Spalte und A(3,:) ist die dritte Zeile von A. A(4:5,2:3)

ist die Untermatrix

$$\left[\begin{array}{cc} -4.6 & 9.0 \\ 5.5 & 7.7 \end{array} \right].$$

Aufgabe 23 (Doppelpunkt) Erzeugen Sie mit der Matlab-Funktion rand eine 5 × 5-Zufallsmatrix *A*. Welches sind die Werte der folgenden Ausdrücke? Überlegen Sie sich die Resultate, bevor Sie die Rechnung am Computer durchführen.

Aufgabe 24 (Zeile löschen) Löschen Sie von der Matrix

$$\mathbf{A} = \left[\begin{array}{rrr} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array} \right]$$

die zweite Zeile.

Lösung:

36.3. Matrizen- und Arrayoperationen

Die Tabelle 14 zeigt Matrizenoperationen in Matlab. Es handelt sich hierbei um Operationen der Matrizenalgebra (Lineare Algebra). Die Operatoren + und - können eingesetzt

Symbol	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
^	Potenzieren

Tabelle 14: Matrizenoperationen

werden, um Matrizen miteinander zu addieren bzw. zu subtrahieren. Der Operator * mulitpliziert zwei Matrizen miteinander, wenn die Multiplikation definiert ist, das heißt wenn die Matrizen die entsprechenden Größen haben.

Ist A eine (m, n)-Matrix und x ein n-Spaltenvektor, so ist das Matrix-Vektor-Produkt Ax definiert.

Aufgabe 25 (Matrizenoperationen) Es seien A, B, C und D die folgenden definierten Matrizen.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -1 & 2 \\ 4 & -2 \\ 7 & -1 \end{bmatrix}$$

$$\boldsymbol{C} = \left[\begin{array}{cc} 1 & 5 \\ -5 & 3 \end{array} \right]$$

und

$$\mathbf{D} = \left[\begin{array}{rrr} 4 & 3 & -2 \\ 1 & 0 & 5 \\ 2 & -1 & 6 \end{array} \right].$$

Berechnen Sie zunächst per Hand folgende Matrizenalgebra. Geben Sie die Matrizen dann in Matlab ein und vergleichen Sie die jeweiligen Resultate.

- (a) $\boldsymbol{A} + \boldsymbol{B}$
- (b) **B**+**C**
- (c) **D**A
- (d) 2A 3B
- (e) A^{T}
- (f) C^2

Lösung:

(b) B + C ist nicht definiert.

◎......

Sind a und b Skalare, so ist a+b die gewöhnliche Addition, a-b die gewöhnliche Subtraktion, a*b die gewöhnliche Multiplikation und a b die gewöhnliche Potenzierung. a/b ist die gewöhnliche Division: $a \div b$. Sie wird auch als rechte Division bezeichnet, weil es auch eine linke Division a\b gibt mit der Bedeutung $b \div a$. Die Operatoren aud haben für Matrizen A und B folgende Bedeutung: A\B ist eine Lösung A der Matrizengleichung A ist eine Lösung A der Matrizengleichung A. Für weiter Einzelheiten siehe Abschnitt A

Matrizenaddition und -subtraktion sind elementweise definiert (im Sinne einer Arrayoperation). Die Multiplikation (Potenzierung) ist jedoch nicht elementweise definiert, also nicht im Sinne einer Arrayoperation. In vielen Anwendungen (siehe zum Beispiel Abschnitt 40) ist es jedoch notwendig, Matrizen elementweise zu verknüpfen. Daher sind in Matlab die Operatoren *, ^, \ und / auch elementweise definiert. Damit eine Operation elementweise durchgeführt werden kann, muss ein Punkt davorgesetzt werden, also .*, .^, .\ und ./.

Die Tabelle 15 fasst die Arrayoperationen (ele-

Symbol	Bedeutung
+	Addition
-	Subtraktion
.*	Multiplikation
./	rechte Division
.\	linke Division
.^	Potenzieren

Tabelle 15: Elementweise Operationen

mentweise Operationen) zusammen.

Unterscheiden Sie zwischen Matrizen- und Arrayoperationen. Die Matrizenoperationen sind im Sinne der Matrizenalgebra (Lineare Algebra), während die Arrayoperationen elementweise zu verstehen sind.

Das folgende Beispiel zeigt eine Arraymultiplikation:

Aufgabe 26 (Operationen) Berechnen Sie A^2 und A.^2 von der Matrix

Lösung: Es ist

Aufgabe 27 (Arrayoperationen) Geben Sie jeweils den Vektor c an, nachdem Sie die folgenden Operationen ausgeführt haben. Überprüfen Sie Ihre Ergebnisse in Matlab.

```
(a) c = b+a-3;
```

(b)
$$c = a./b$$
;

(c)
$$c = 2*a+a.b;$$

(d)
$$c = 2.^b+a;$$

(e)
$$c = 2*b/3.*a;$$

Analog für die anderen Operationen. Manchmal ist es notwendig, Matrizen elementweise zu multiplizieren (Hadamard-Produkt), dann hilft der .*-Operator.

```
>> C = [1 2; 3 4];

>> D = [5 6; 7 8];

>> C.*D

ans =

5 12

21 32
```

Die konjugiert Transponierte der Matrix A er- in Matlab das direkte Produkt von hält man durch A'. Ist A eine reelle Matrix, so ist A' einfach die Transponierte. Die Transponierte ohne Konjugation erhält man mit A.'.

Im Spezialfall, wenn x und y Spaltenvektoren sind, ist x'*y das Skalarprodukt. Dies lässt sich auch mit dot(x,y) berechnen. Mit der Funktion cross kann man das Kreuzprodukt ausrechnen. Für weiter Einzelheiten siehe Abschnitt 48.

Aufgabe 29 (Matrizenoperationen) Es seien zwei Vektoren a und b wie folgt definiert:

$$a = [2,4,6]$$
 $b = [1,2,3]$

Führen Sie die folgenden Matlab-Operationen durch. Welche sind definiert und welche nicht? Erklären Sie! Was sind die Resultate?

Die Funktion kron berechnet das Kronecker-Produkt (direktes Produkt) zweier Matrizen, siehe auch [5].

Aufgabe 30 (Direktes Produkt) Berechnen Sie

$$\mathbf{A} = \left[\begin{array}{rrr} 1 & 2 & 3 \\ 3 & 1 & -2 \\ 5 & -1 & 0 \end{array} \right]$$

und

$$\boldsymbol{B} = \left[\begin{array}{rrr} 1 & 3 \\ -1 & 5 \\ 2 & -2 \end{array} \right].$$

Lösung: Das direkte Produkt ist auch unter dem Namen Kronecker-Produkt bekannt und kann mit der Funktion kron berechnet werden.

Aufgabe 31 (Vektoren, Signale) Gegeben ist das kontinuierliche Signal (die kontinuierliche Funktion)

$$s(t) = \sin(2\pi t)e^{-0.3t}.$$

Erzeugen Sie für t = 0, 0.1, 0.2, ..., 10 das dazugehörige diskrete Signal (Vektor). Wieviel Werte hat das Signal?

Lösung:

```
>> t = 0:0.1:10;
>> sd = sin(2*pi*t).*exp(-0.3*t);
```

Wird ein Skalar in MATLAB zu einer Matrix addiert oder Subtrahiert, so wird die Zahl zu jedem Matrixelement addiert bzw. subtrahiert. Das Gleiche gilt auch bei Mulitplikation und Division mit einer Zahl. Beispiel:

```
>> [1 2 3; 4 5 6]/2
ans =
0.5000 1.0000 1.5000
2.0000 2.5000 3.0000
```

Die meisten mathematischen Funktionen, siehe Abschnitt 33, akzeptieren als Eingabeargument eine Matrix. Dann wird diese Funktion elementweise ausgeführt. Funktionen einer Matrix im Sinne der Linearen Algebra, wie zum Beispiel die Exponentialfunktion, tragen am Ende des Namens ein m: expm, funm, logm, sqrtm. Für die Matrix A = [2 1; 0 2] gilt:

```
>> sqrt(A)
ans =
    1.4142
               1.0000
               1.4142
>> sqrtm(A)
ans =
    1.4142
               0.3536
               1.4142
>> ans*ans
ans =
    2.0000
               1.0000
               2.0000
         0
```

36.4. Matrizenmanipulationen

Die Tabelle 16 zeigt Funktionen mit denen man

Funktion	Bedeutung
reshape	Ändert Ordnung
diag	Diagonalmatrix
blkdiag	Blockdiagonalmatrix
tril	Untere Dreiecksmatrix
triu	Obere Dreiecksmatrix
fliplr	Vertauscht
flipud	Vertauscht
rot90	Drehung um 90°

Tabelle 16: Matrizenmanipulationen

Matrizen manipulieren kann.

Die Funktion reshape ändert die Ordnung einer Matrix: reshape(A,m,n) erzeugt eine (m,n)-Matrix aus der Matrix A, wobei die Elemente spaltenweise extrahiert werden.

Die Funktionen tril und triu extrahieren eine Dreiecksmatrix aus einer gegebenen Matrix.

Aufgabe 32 (Dreiecksmatrizen) Erzeugen Sie eine 6 × 6 obere (untere) Dreiecksmatrix mit Zufallszahlen zwischen 0 und 1! *Lösung:*

Aufgabe 33 (Matrizenmanipulationen) Gegeben seien die folgenden Matrizen:

$$\mathbf{A} = \left[\begin{array}{cccc} 0 & -1 & 0 & 3 \\ 4 & 3 & 5 & 0 \\ 1 & 2 & 3 & 0 \end{array} \right]$$

und

$$\boldsymbol{B} = \left[\begin{array}{cccc} 1 & 3 & 5 & 0 \\ 3 & 6 & 9 & 12 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{array} \right].$$

Bestimmen Sie die Rückgabewerte und überprüfen Sie diese dann in MATLAB.

- (a) rot90(B)
- (b) rot90(A,3)
- (c) fliplr(A)
- (d) reshape(A,4,3)
- (e) triu(B)
- (f) diag(rot90(B))

Aufgabe 34 (circshift) Verschieben Sie jede Zeile der Matrix

$$\mathbf{A} = \left[\begin{array}{rrr} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array} \right]$$

um eine Zeile nach unten und die letzte in die erste Zeile.

Lösung:

7	8	9	
1	2	3	
4	5	6	

Aufgabe 35 (fliplr) Spiegeln Sie von der

Aufgabe 35 (fliplr) Spiegeln Sie von der Matrix

$$\mathbf{A} = \left[\begin{array}{rrr} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array} \right]$$

die Spalten.

Lösung:

36.5. Datenanalye

Die Tabelle 17. zeigt grundlegende Funktionen zur Analyse von Daten. Der einfachste Gebrauch dieser Funktionen ist, wenn die Daten in Vektoren gegeben sind. Zum Beispiel:

Funktion	Bedeutung
max	Maximales Element
min	Minimales Element
mean	Arithmetischer Mittelwert
median	Zentralwert (Median)
std	Standardabweichung
var	Varianz
sort	Sortiert
sum	Berechnet Summenwert
prod	Berechnet Produktwert
cumsum	Summiert
cumprod	Multipliziert
diff	Bildet Differenz

Tabelle 17: Datenanalye

Die Funktion sort sortiert in aufsteigender Ordnung; absteigende Ordnung erhält man durch das Extraargument 'descent'. Sind die Vektoren komplex, so wird nach dem Absolutwert geordnet. NaNs werden von dem Funktionen min und max ignoriert und sort setzt sie an das Ende.

Sind die Eingabeargumente Matrizen, so arbeiten die Funktionen zur Datenanalyse spaltenweise. Ist zum Beispiel

so ist

```
5 2 2
```

Aufgabe 36 (Datenanalye) Gegeben seien die Vektoren

```
x = [0 \ 3 \ -2 \ 7];

y = [3 \ -1 \ 5 \ 7];
```

und die Matrix

$$A = [1 \ 3 \ 7; \ 2 \ 8 \ 4; \ 6 \ -1 \ -2];$$

Bestimmen Sie folgende Ausdrücke, zunächst mit Bleistift und Papier, danach mit Matlab.

```
max(x); min(A);
min(x,y); mean(A);
median(x); cumprod(A);
sort(2*x+y); sort(A);
```

Lösung: Es ist

```
>> max(x)
ans =
>> min(A)
ans =
     1
           -1
                 -2
\gg \min(x,y)
ans =
                 -2
                         7
>> mean(A)
ans =
    3.0000
               3.3333
                           3.0000
>> median(x)
    1.5000
>> cumprod(A)
ans =
     1
            3
                  7
     2
           24
                 28
```

Aufgabe 37 (Datenanalye) Bestimmen Sie die Werte der folgenden Ausdrücke. Überprüfen Sie Ihre Ergebnisse dann in MATLAB.

$$\mathbf{B} = \left[\begin{array}{rrr} 1 & 0 & 4 \\ 0 & 0 & 3 \\ 8 & 7 & 0 \end{array} \right]$$

- (a) any(B)
- (b) find(B)
- (c) all(any(B))
- (d) any(all(B))
- (e) finite(B(:,3))
- (f) any(B(1:2,1:3))

Lösung:

```
>> any(B)
ans =
     1
            1
                   1
>> find(B)
ans =
     1
     3
     6
     7
     8
>> all(any(B))
```

```
ans =
     1
>> any(all(B))
ans =
>> finite(B(:,3))
ans =
     1
     1
     1
>> any(B(1:2,1:3))
ans =
     1
                  1
```

Aufgabe 38 (Größtes Element) Bestimmen Sie von der Matrix

$$\mathbf{A} = \left[\begin{array}{rrr} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array} \right]$$

das größte Element.

Lösung: Dies kann man wie folgt erreichen:

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> max(max(A))
ans =
     9
```

37. Vektorielle Programmierung

Eine der großen Vorzüge von Matlab besteht darin, dass man Ausdrücke vektoriell programmieren und so auf Schleifen weitgehend verzichten kann. Was ist damit gemeint? Wir zeigen die Idee am Besten an einem einfachen onsterm $y = x^2$ im Punkt x = 1.4 auswerten, so geht das wie folgt

```
>> x = 1.4;
>> y = x^2;
```

und in der Variablen y ist der Wert 1.96 gespeichert. Will man nun aber zum Beispiel in hundert verschiedenen x-Werten die entsprechenden y-Werte (ein Vektor mit 100 Koordinaten) berechnen, so braucht man in Mat-LAB keine Schleife, sondern kann gleichzeitig alle hundert y-Werte ausrechnen, indem man den ^.-Operator verwendet, der die Potenzierung zweier Matrizen (Vektoren) elementweise durchführt. Hier ein Beispiel mit der gleichen Funktionsgleichung $y = x^2$ wie oben:

```
>> x = linspace(0,3,100);
\gg y = x.^2;
```

In der ersten Anweisung werden hundert x-Werte (ein Vektor mit hundert Koordinaten) erzeugt (hundert äquidistante Punkte im Intervall [0, 3]) und in der zweiten Anweisung wird der Vektor y mit hundert Koordinaten generiert, der die entsprechenden Funktionswerte enthält. Diese Vorgehensweise hat den Vorteil, dass man auf Schleifen verzichten kann und außerdem einen übersichtlicheren Programmcode erhält. Eingebaute MATLAB-Funktionen erlauben als Eingabeparameter fast immer Vektoren. Daher gilt:

> Programmieren Sie vektoriell, wenn immer dies möglich ist!

Der Befehl vectorize kann in der ein oder anderen Situation hilfreich sein, siehe doc vectorize (help vectorize). In Abschnitt

Beispiel. Will man zum Beispiel den Funkti- 47 gehen wir auf diese Thematik nochmals ausführlicher ein.

> Aufgabe 39 (Vektorielle Prog.) Berechnen Sie e^{-x^2} in 200 äquidistanten Punkten zwischen -3 und 3 und speichern Sie die Werte in der Variablen y.

Lösung: Hier ein möglicher Lösungsweg:

```
>> x = linspace(-3,3,200);
\gg y = exp(-x.^2);
```

38. Ein- und Ausgabe

In diesem Abschnitt diskutieren wir, wie durch einen Benutzer Daten eingelesen werden können, wie man Informationen auf dem Bildschirm ausgeben und Dateien lesen und schreiben kann. Wie man eine ganze Sitzung in einer Datei auszeichnen kann, wissen wir bereits aus Abschnitt 13. Wie Grafiken gespeichert und gedruckt werden können, steht in Abschnitt 40.13. Zur Dateneingabe siehe auch Abschnitt 10.

38.1. Benutzereingabe

Mit der Funktion input kann der Benutzer interaktiv Daten eingeben.

```
>> x = input('Startwert: ')
Startwert: 1.5
x =
    1.5000
```

In diesem Beispiel wartet MATLAB auf die Eingabe eines Wertes durch den Benutzer. Ich habe 1.5 eingegeben und dieser Wert wird dann der Variablen x zugewiesen. Die Eingabe wird als String interpretiert, wenn ein zweites Argument 's' angehängt wird:

```
>> MeinTitel = input('Titel: ','s')
Titel: Versuch 5
MeinTitel =
Versuch 5
```

38.2. Bildschirmausgabe

Wir wissen bereits, dass MATLAB immer dann eine Ausgabe am Bildschirm macht, wenn auf das Semikolon verzichtet wird, siehe Abschnitt 25. Außerdem kann mit den Funktion format die Ausgabe kontrolliert werden. Eine noch bessere Kontrolle über die Ausgabe erhält man aber durch die Verwendung weiterer Funktionen.

Als erste Funktion ist disp zu nennen. Ohne Namen und Gleichheitszeichen gibt diese den Wert einer Variablen am Bildschirm an. Hier ein Beispiel:

```
>> disp('Die (2,2)-Hilbert-Matrix')
Die (2,2)-Hilbert-Matrix
>> disp(hilb(2))
    1.0000    0.5000
    0.5000    0.3333
```

Mehr Ausgabemöglichkeiten hat man mit der Funktion fprintf. In dem Beispiel

```
>> fprintf('%6.3f\n',pi)
3.142
```

ist das Zeichen % der Beginn einer Formatbezeichnung. 6.3f bedeutet, dass der Wert der Variablen mit der Feldgröße (zur Verfügung gestellte Gesamtbreite) 6, 3 Nachkommastellen und in Fließpunktdarstellung ausgegeben wird. Das Zeichen \n gibt einen Zeilenvorschub an, ohne dieses die folgende Ausgabe in die aktuelle Zeile geschrieben werden würde. Siehe doc fprintf (help fprintf) für weitere Informationen und Beispiele.

Die Funktion sprintf ist analog zur Funktion fprintf, aber die Ausgabe geschieht als Zeichenkette, siehe doc sprintf (help sprintf) für weitere Einzelheiten.

Aufgabe 40 (Tabellen) Schreiben Sie einen Script-File ExpTabelle und erzeugen Sie auf dem Bildschirm die folgende Ausgabe der Exponentialfunktion e^x . Wählen Sie 21 äquidistante Punkte im Intervall [0,1]. Verwenden Sie dazu die Funktionen disp und sprintf oder fprintf.

Werte	der exp-Funktion	
k	exp(x(k))	
1	1.0000	
2	1.0513	
3	1.1052	
4	1.1618	
5	1.2214	
6	1.2840	
7	1.3499	
8	1.4191	
9	1.4918	
10	1.5683	
11	1.6487	
12	1.7333	
13	1.8221	
14	1.9155	

15	2.0138
16	2.1170
17	2.2255
18	2.3396
19	2.4596
20	2.5857
21	2.7183

Lösung: Der Inhalt des Script-Files könnte etwa wie folgt lauten:

```
x = linspace(0,1,21);
y = [1:21; exp(x)];
disp('Werte der exp-Funktion')
disp(' ')
disp(' k exp(x(k))')
disp(' -----')
disp(sprintf('%6.0f %15.4f\n',y))
```

©.....

38.3. Dateien lesen und schreiben

Eine große Anzahl von Funktionen stehen zur Verfügung, um formatierte Textdateien aus einer Datei zu lesen oder in eine Datei zu schreiben. Hierbei können Sie zwischen binärem und lesbarem Ascıı-Format wählen. Binärdateien haben gegenüber Ascıı-Dateien die Vorteile, dass sie weniger Speicherplatz benötigen und schneller gelesen und beschrieben werden können. Ihr Nachteil ist, dass sie nicht einfach mit einem Editor gelesen werden können. Die Tabelle 18 zeigt vier Funktionen. Eine komplette Funktionsliste erhalten Sie mit doc iofun (help iofun).

Wir zeigen nun an einem Beispiel, wie man Daten in einen File schreibt und lesen sie an-

Funktion	Ascii	binär
fscanf	×	
fprintf	×	
fread		×
fwrite		×

Tabelle 18: Dateien lesen und schreiben

schließend wieder heraus. Bevor man mit einer Datei arbeiten kann, muss man sie mit der Funktion fopen öffnen. Das erste Argument ist der Dateiname und für das zweite Argument hat man mehrere Möglichkeiten unter anderem die Zeichen 'r' für Lesen (read) und 'w' für Schreiben (write). Durch den Aufruf von fopen wird eine Fileidentifizierung zurückgegeben, die in den folgenden Anweisungen verwendet werden kann, um den File zu spezifizieren. Die Daten werden mit der Funktion fprintf geschrieben, wobei als erstes Argument die Fileidentifizierung anzugeben ist. Das Script

```
>> a = [30 40 50];
>> fid = fopen('MeinOutput','w');
>> fprintf(fid,...
    '%g m/h = %g km/h\n',[a;8*a/5]);
>> fclose(fid);
```

erzeugt im File MeinOutput die Zeilen

```
30 m/h = 48 km/h
40 m/h = 64 km/h
50 m/h = 80 km/h
```

die für drei Geschwindigkeiten eines Fahrzeugs in Meilen pro Stunde die entsprechenden Werte in Kilometer pro Stunde angeben.

Der File kann wie folgt zurückgelesen werden.

Hierbei liest die Funktion fscanf die Daten in dem angegebenen Format (hier %g, general floating point number), wobei die Strings 'm/h =' und 'g km/h' übergangen werden. Dies wiederholt sich so lange, bis der gesamte File gelesen wurde und die Zahlen im Vektor x gespeichert wurden.

39. Function Functions

In Matlab hat sich der Begriff function function eingebürgert. Damit ist eine Matlab-Function gemeint, die ihrerseits als Eingabeargument eine Function benötigt. (Matlab-Functions werden wir noch ausfrührlich behandeln). Ein Beispiel einer Function Function ist fplot, mit der man Graphen von mathematischen Funktionen zeichnen kann. Diese Function benötigt als erstes Eingabeargument eine Function. Wir zeigen die Arbeitsweise an einem Beispiel.

Geplottet werden soll die Funktion $f(x) = x^2$, $x \in \mathbb{R}$ im Intervalle [-2, 2]. Diese kann in Matland durch

```
>> f = @(x) x.^2;
```

definiert werden. Der Aufruf

```
>> fplot(f,[-2,2])
```

zeichnet dann die Funktion im Intervalle von -2 bis 2. Man sieht, dass das erste Argument von fplot selbst eine Funktion ist, nämlich in diesem Fall die selbstdefinierte Funktion $f(x) = x^2$, $x \in \mathbb{R}$. Für eingebaute Funktionen entfällt natürlich deren Definition. Hier ein Beispiel mit der eingebauten Sinusfunktion sin. Die folgende Zeile plottet die Sinusfunktion im Intervall von -3 bis 3:

```
>> fplot(@sin,[-3,3])
```

Andere Function Functions sind zum Beispiel fzero, quad, ode45, usw. Wir werden diese und andere Funktionen in den späteren Abschnitten ausführlicher besprechen. Alle Function Functions sind im Verzeichnis funfun untergebracht, siehe doc funfun (help funfun).

40. Grafik

Twenty years ago, we did not interact with computers graphically; now, everything is graphical. In the next twenty years an equally great change will occur as we begin to communicate with machines by speech.

Lloyd N. Trefethen

Matlab verfügt über moderne und mächtige Visualisierungsmöglichkeiten. Dies ist einer der Gründe für den Erfolg von Matlab.

Das Visualisieren von Daten ist typisch im praktischen Einsatz von Matlab, während das Zeichnen von explizit bekannten Funktionen sehr von Nutzen in der Lehre ist. Eine Vielzahl von Gestaltungsmöglichkeiten statistische Daten darzustellen stehen bereit. Auf der Homepage von *The Mathworks* steht ein Online-Grafikhandbuch zur Verfügung, siehe [15].

Wenn wir nun grafische Fähigkeiten von Mat-LAB zeigen, so handelt es sich um die Beschreibung grafischer Funktionen, die man zur Visualisierung von Daten im *Command Window* eingibt. Fast alle hier aufgezeigten Merkmale können Sie nach Öffnen einer Figur durch Mausklick über den *Plot Browser*, *Figure Palette*, *Property Editor*, usw. beeinflussen (siehe Menü-Item *Show Plot Tools* in der geöffneten Figur).

40.1. 2D-Grafik

Wir beginnen die grafischen Möglichkeiten von Matlab mit 2D-Grafiken (zweidimensionalen Grafiken). Eine häufig verwendete Funktion ist die plot-Funktion (doc plot). Dieser werden im einfachsten Fall Koordinaten von Punkten aus der Ebene übergeben, die dann von plot durch gerade Linien verbunden werden. Ein einfaches Beispiel soll dies erläutern.

Hierzu nehmen wir an, dass die Messung des zeitlichen Verlaufs der Abkühlung einer Flüssigkeit die Werte aus der Tabelle 19 ergab. Wir wollen dieses Messergebnis nun grafisch darstellen. Hierzu speichern wir die Zeitpunkte im Vektor x und die Temperaturwerte in y, also

$$x = [0 \ 0.5 \ 1 \ 1.5 \ 2 \ 2.5 \ 3 \ 3.5 \ 4];$$

Zeitpunkt in min	Temperatur in 0C
0.0	62
0.5	55
1.0	48
1.5	46
2.0	42
2.5	39
3.0	37
3.5	36
4.0	35

Tabelle 19: Abkühlung einer Flüssigkeit

$$y = [62 55 48 46 42 39 37 36 35];$$

Der Befehl

erzeugt ein Grafikfenster und zeichnet die Elemente von x gegen die Elemente von y und verbindet diese Punkte geradlinig. Die Abbildung 10 zeigt das Ergebnis.

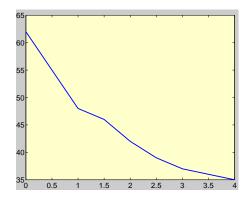


Abbildung 10: Abkühlung einer Flüssigkeit

Wir zeichnen nun den Graph der explizit gegebenen Funktion $f(x) = \sin(x)$ auf dem Intervall

 $[0, 2\pi]$. Dazu müssen drei Dinge getan werden:

1. Einen Vektor x erzeugen, der das Intervall $[0, 2\pi]$ diskretisiert:

$$0 = x_1 < x_2 < \cdots < x_n = 2\pi$$

2. Die Funktion muss an jedem Diskretisierungspunkt ausgewertet werden:

$$y_k = f(x_k)$$
 $k = 1:n$

3. Ein Streckenzug muss gezeichnet werden, der die Punkte $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ verbindet.

Das folgende Script zeigt die Realisierung:

```
>> n = 20;
>> x = linspace(0,2*pi,n);
>> y = sin(x);
>> plot(x,y)
```

Hierzu haben wir das Intervall $[0,2\pi]$ in 20 äquidistante Punkte eingeteilt und die Werte dem Vektor x zugeordnet. Die Sinusfunktion ist eine eingebaute Matlab-Funktion, die Vektoren als Argumente verarbeiten kann. Dadurch wird der Vektor y erzeugt. Mit grid zeichnen wir noch ein Gitter und geben mit title der Abbildung noch eine Überschrift.

```
>> grid
>> title('Die Sinusfunktion im
Intervall [0,2\pi]')
```

Die Abbildung 11 zeigt das Ergebnis.

Aufgabe 41 (2D-Grafik) Erzeugen Sie mit einem einzigen plot-Befehl die Graphen der Funktionsterme $\sin(kx)$ über dem Intervall $[0, 2\pi]$ für k = 1:5.

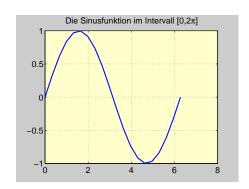


Abbildung 11: Ein einfacher Plot

Lösung: Dies kann man zum Beispiel mit den beiden folgenden Methoden erreichen.

```
%-Methode 1:
x = linspace(0, 2*pi);
plot(x, sin(x), x, sin(2*x), x, ...
sin(3*x), x, sin(4*x), x, sin(5*x))
%-Methode 2:
x = linspace(0, 2*pi);
Y = [\sin(x); \sin(2^*x); \sin(3^*x); ...
     sin(4*x);sin(5*x)];
plot(x,Y)
%-Methode 3:
x = linspace(0,2*pi);
plot(x,sin(x))
hold on
for k=2:5
   plot(x, sin(k*x), '-')
hold off
```

Sufgabe 42 (2D-Grafik) Zeichnen Sie den

Graph der Funktion

$$rect(x) = \begin{cases} 1 & |x| \le 0.5\\ 0 & sonst \end{cases}$$

über dem Intervall [-3, 3].

Lösung: Wir definieren zunächst die Funktion *rect* in einem Function-File und zeichnen dann mit plot den Graph der Funktion.

```
function y = rect(x)
n = length(x);
y = zeros(n,1);
y = (x<0.5)-(x<-0.5);</pre>
```

Die folgenden Befehle zeichnen den Graph.

```
x = linspace(-3,3,1000);
y = rect(x);
plot(x,y)
axis([-3 3 -0.5 1.5])
```

©.....

Aufgabe 43 (2D-Grafik) Geben Sie folgende Befehle in Matlab ein.

```
x = linspace(0,1,200);
y = sqrt(1-x.^2);
```

Stellen Sie nun den Kreis $x^2 + y^2 = 1$ grafisch dar ohne dabei zusätzliche Berechnungen anzustellen.

Lösung: Die Eingabe lautete

```
x = linspace(0,1,200);
y = sqrt(1-x.^2);
```

Dabei sei zu bemerken, das $x.^2$ und nicht x^2 verwendet wurde. Hierbei wird durch den Punkt sichergestellt, dass das Quadrieren Komponentenweise erfolgt und nicht die Vektoroperation $x \cdot x$ ausgefürt wird. Man will nun

den Kreis in \mathbb{R}^2 zeichnen.

plot(x,y) liefert nur den Graphen im ersten Quadranten. Also kann man durch geeignete Spiegelung den Graph auch in den anderen drei Quadranten zeichnen. Der Kreis kann in Mat-LAB etwa wie folgt gezeichnet werden:

```
>> plot(x,y)
>> hold on
>> plot(-x,y)
>> plot(x,-y)
>> plot(-x,-y)
>> axis equal
>> hold off
```

Der Befehl hold on sorgt dafür, dass die Resultate aller plot-Befehle erhalten bleiben. Um die Bedeutung des Befehls axis equal zu verstehen können Sie help axis verwenden.

Mit der Funktion subplot besteht die Möglichkeit, mehrere Bilder untereinander und/oder nebeneinander in einer Figur zusetzen, siehe doc subplot (help subplot).

40.2. 3D-Grafik

Die zur plot-Funktion analoge Funktion für die 3D-Welt ist die plot3-Funktion. Die Verwendung von plot3 ist ähnlich wie die von plot, wobei statt Paare nun Tripel von Daten einzugeben sind. Das folgende Beispiel zeigt den einfachsten Gebrauch: plot3(x,y,z) zeichnet eine Kurve im Raum, indem die Punkte x(i),y(i),z(i) in der vorgegebenen Reihenfolge verbunden werden. Das Ergebnis ist in Abbildung 12 zu sehen.

```
t = -5:0.005:5;
x = (1+t.^2).*sin(20*t);
y = (1-t.^2).*cos(20*t);
z = t;
plot3(x,y,z), grid on
xlabel('x(t)'), ylabel('y(t)'),...
zlabel('z(t)')
title('\it{plot3-Beispiel}',...
'FontSize',14)
```

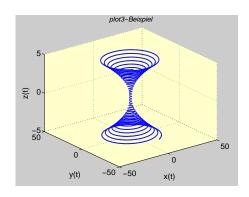


Abbildung 12: 3D-Plot mit plot3

In diesem Beispiel haben wir die Funktionen xlabel, ylabel, title und entsprechend zlabel zur Beschriftung der Abbildung verwendet. Die Notation \it im title-Kommando stammt aus TEX, um Italic-Text zu erzeugen. Farben, Marken und Linienstile können in der gleichen Weise wie bei der plot-Funktion beeinflusst werden. So erzeugt zum Beispiel plot3(x,y,z,'r-') eine rot gestrichelte Linie. Beachten Sie, dass für 3D-Plots box off gesetzt ist. Mit box on können Sie Ihrem Plot eine Box hinzufügen.

Will man den Graph eines Funktionsterms f(x, y) mit den beiden unabhängigen Variablen

x, y zeichnen, so muss man die Funktionswerte auf einem zweidimensionalen Gitter in der x, y-Ebene auswerten. Das Gitter kann mit der Funktion meshgrid erzeugt werden; anschließend kann man den Graph mit mesh, surf usw. zeichnen.

Als Beispiel soll der Graph der Funktion $f(x,y) = -xye^{-2(x^2+y^2)}$ über dem Bereich $[-2,2] \times [-2,2]$ gezeichnet werden.

```
>> [X,Y] = meshgrid(-2:0.1:2, ...
-2:0.1:2);
>> f = -X.*Y.*exp(-2*(X.^2+Y.^2));
>> mesh(X,Y,f)
>> xlabel('x'), ylabel('y'), ...
zlabel('f(x,y)')
```

Die Abbildung 13 zeigt das Ergebnis.

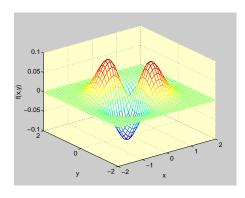


Abbildung 13: $f(x, y) = -xye^{-2(x^2+y^2)}$

Aufgabe 44 (3D-Grafik) Zeichnen Sie den Graph der Funktion

$$f(x_1, x_2) = \begin{cases} -\frac{1}{2}x_2 + 2 & \text{für } x_1, x_2 \ge 0\\ 2 & \text{sonst.} \end{cases}$$

im Bereich $(x_1, x_2) \in [-2, 2]^2$.

Lösung: Den Graph kann man wie folgt plot- 40.3. Funktionsdarstellungen

Mehr Informationen über Visualisierungsmöglichkeiten findet man mit doc graph2d (help graph2d), doc graph3d (help graph3d) und doc specgraph (help specgraph).

Aufgabe 45 (3D-Grafik) Zeichnen Sie den Graph des Funktionterms

$$f(x, y) = \frac{1}{5}\cos(x) + y\exp(-x^2 - y^2)$$

mit den Funktionen mesh und ezmesh über dem Quadrat $-3 \le x \le 3, -3 \le y \le 3$.

Lösung:

oder als Einzeiler mit ezmesh.

Kennt man den Funktionsterm einer reellwertigen Funktion einer reellen Variablen, so kann man mit den Funktionen fplot und ezplot (easy plotting) den Graph einfacher darstellen als mit plot. Die Funktion fplot verlangt den Funktionsterm in einem m-File (Abschnitt 44) oder als Function Handle (doc function_handle). Die Funktion ezplot erwartet den Funktionsterm in Hochkomma oder als symbolisches Objekt, siehe Abschnitt 67.

Der Aufruf

plottet die Funktion

$$f(x) = e^{-x^2}, \quad x \in \mathbb{R}$$

im Intervall [-3, 3]. Mit

plottet man im Intervall [-2, 2] die eingebaute humps-Funktion

$$f(x) = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.04} - 6, \quad x \in \mathbb{R}.$$

Die Abbildung 14 zeigt den Graph von humps im Intervall [-2, 2].

Aufgabe 46 (Funktionsdarstellungen) Zeichnen Sie den Graph des Funktionsterms

$$f(x) = \sin(x^2) - 2\cos(x)$$

über dem Intervall (0,5) mit den Funktionen plot, fplot und ezplot.

Lösung: Dies erreicht man wie folgt:

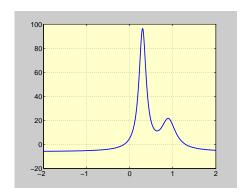


Abbildung 14: Graph der humps-Funktion

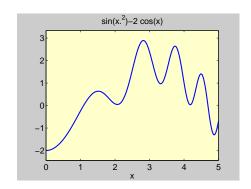


Abbildung 15: Graph

```
x = linspace(0,5);
f = sin(x.^2)-2*cos(x);
plot(x,f)
%-oder:
fplot('sin(x^2)-2*cos(x)',[0,5])
%-bzw.
f = @(x) sin(x.^2)-2*cos(x);
fplot(f,[0,5])
%-oder:
ezplot('sin(x^2)-2*cos(x)',[0,5])
%-bzw.
f = @(x) sin(x.^2)-2*cos(x);
ezplot(f,[0,5])
```

Der Graph ist in Abbildung 15 dargestellt. ⊚ ⊚

Die Abbildung 16 zeigt vier verschiedene Darstellungen einer Funktion mit zwei Variablen. Es handelt sich hier um die sogenannte peaks-Funktion, die in MATLAB bereits vordefiniert ist.

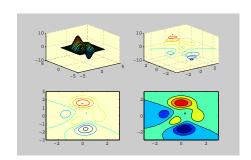


Abbildung 16: Darstellungen der peaks-Funktion

Es ist die Funktion

$$f(x,y) = 3(1-x)^{2}e^{-x^{2}-(y+1)^{2}}$$
$$-10(x/5-x^{3}-y^{5})e^{-x^{2}-y^{2}}$$
$$-1/3e^{-(x+1)^{2}-y^{2}}, \quad (x,y) \in \mathbb{R}^{2}$$

Die peaks-Funktion geht durch Translationen und Skalierungen aus der Gaussschen Normalverteilungsfunktion hervor. Das Bild links oben in Abbildung 16 zeigt den Graph, rechts oben Kurven gleicher Höhe (Höhenschittbilder), links unten ein paar Höhenlinien und rechts daneben farbig ausgefüllte Höhenlinien der peaks-Funktion. Die Figur wurde mit den folgenden Anweisungen erzeugt:

```
[X,Y,Z] = peaks(30);
subplot(2,2,1), surf(X,Y,Z),
subplot(2,2,2), contour3(X,Y,Z),
subplot(2,2,3), contour(X,Y,Z),
subplot(2,2,4), contourf(X,Y,Z),
```

Die Abbildung 17 zeigt den Graph der Funkti-

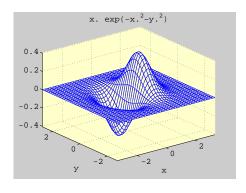


Abbildung 17: Graph

on $f(x, y) = xe^{x^2+y^2}$, $(x, y) \in \mathbb{R}^2$, wobei hier die Funktion colormap zum Einsatz kommt und

dafür sorgt, dass der Graph (das Netz) blau ist. Die Funktion colormap erlaubt es, Daten mit Farbtabellen (color map) zu visualisieren. Mit der Funktion colorbar können Sie sich die aktuelle Farbtabelle in der entsprechenden Figur anzeigen lassen. Die Figur wurde mit Hilfe der Anweisungen erzeugt:

```
fh = @(x,y) x.*exp(-x.^2-y.^2);
ezmesh(fh,40)
colormap([0 0 1])
```

Die Abbildung 18 zeigt Höhenlinien der

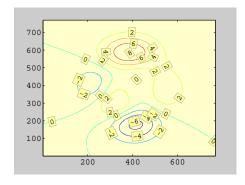


Abbildung 18: Geglättete Höhenlinien der peaks-Funktion

peaks-Funktion, wobei die Höhenlinien nun mit der Funktion interp2 geglättet sind. Außerdem bekommen die Höhenzahl einen leicht gelblichen Hintergrund mit einem leicht grauen Rahmen. Die Figure wurde mit Hilfe der Anweisungen erzeugt:

```
Z = peaks;
[C,h] = contour(interp2(Z,4));
text_h = clabel(C,h);
set(text_h,'BackgroundColor',...
[1 1 .6],'Edgecolor',[.7 .7 .7])
```

40.4. Parametrisierte Kurven

Mit Hilfe der Funktionen ezplot und ezplot3 lassen sich Kurven in Parameter-darstellung in zwei und drei Dimensionen darstellen. Als Beispiel einer ebenen Kurve betrachten wir eine dreiblättrige Blütenblatt-kurve (Trochoide) Sie hat die Parameterform $x = \cos(3t)\cos(t), y = \cos(3t)\sin(t), t \in [0, 2\pi]$. Mit Hilfe der Anweisung

```
r1 = @(t) cos(3*t).*cos(t);
r2 = @(t) cos(3*t).*sin(t);
ezplot(r1,r2,[0,2*pi]), grid;
```

erzeugt man die Abbildung 19.

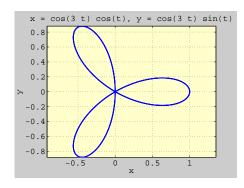


Abbildung 19: Blütenblattkurve

Die Anweisungen

```
r1 = @(t) exp(-0.2*t).*cos(t);
r2 = @(t) exp(-0.2*t).*sin(t);
r3 = @(t) t;
ezplot3(r1,r2,r3,[0,20],'animate')
```

erzeugen die räumliche Kurve in Abbildung 20. Durch das zusätzlich Argument animate im Funktionsaufruf erhält man eine Animati-

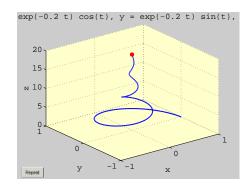


Abbildung 20: Räumliche Kurve

on der räumlichen Kurven in dem Sinn, dass ein roter Punkte vom Anfagang- zum Endpunt läuft

Aufgabe 47 (Räumliche Kurve) Plotten Sie die räumliche Kurve

$$x(t) = (1 + t2) \sin(20t)$$

$$y(t) = (1 + t2) \cos(20t)$$

$$z(t) = t$$

für $t \in [-5, 5]$.

Lösung: Die Kurve kann zum Beispiel wie folgt geplottet werden.

```
t = -5:0.005:5;
x = (1+t.^2).*sin(20*t);
y = (1+t.^2).*cos(20*t);
z = t;
plot3(x,y,z),
grid on, xlabel('x(t)'),
ylabel('y(t)'), zlabel('z(t)')
```

Aufgabe 48 (Ebene Kurve) Plotten Sie die Zy-

kloide (Rollkurve)

$$x(t) = t - \sin t$$
$$y(t) = 1 - \cos t$$

für $t \in [0, 4\pi]$.

Lösung: Die Kurve kann zum Beispiel wie folgt geplottet werden.

Aufgabe 49 (Ebene Kurve) Zeichnen Sie die Kurve

$$x = \sin(-t) + t$$
$$y = 1 - \cos(-t)$$

in der x, y-Ebene für $0 \le t \le 4\pi$.

Lösung: Es handelt sich um eine Zykloide.

Aufgabe 50 (Ebene Kurve) Zeichnen Sie die Kurve

$$x = \sin(-t)$$
$$y = 1 - \cos(-t)$$

in der x, y-Ebene für $0 \le t \le \pi$.

Lösung: Es handelt sich um einen Kreisbogen.

Aufgabe 51 (Räumliche Kurve) Zeichnen Sie die Schraubenlinie (Helix, zylindrische Spirale)

$$x = \cos(t)$$
$$y = \sin(t)$$
$$z = t$$

im x, y, z-Raum für $0 \le t \le 20\pi$. Animieren Sie!

Lösung: Die Schraubenlinie mit Animation erhält man wie folgt:

Aufgabe 52 (Räumliche Kurve) Zeichnen Sie die konische Spirale

$$x = (1-t)\cos(t)$$

$$y = (1-t)\sin(t)$$

$$z = t$$

im x, y, z-Raum für $0 \le t \le 20\pi$. Animieren Sie!

Lösung: Die konische Spirale mit Animation erhält man wie folgt:

```
>> ezplot3('(1-t)*cos(t)','(1-t)*sin(t)','t',...
[0,20*pi],'animate')
```

40.5. Parametrisierte Flächen

Mit den Matlab-Funktion ezmesh und ezsurf können parametrisierte Flächen im \mathbb{R}^3 dargestellt werden.

Ein Torus entsteht, wenn ein Kreis um eine Achse rotiert, die in der Ebene des Kreises, aber außerhalb des Kreises verläuft. Eine Parameterdarstellung eines Torus ist:

$$x = (a + b\cos\theta)\cos\phi$$
$$y = (a + b\cos\theta)\sin\phi$$
$$z = b\sin\theta$$

Der folgende Script zeichnet einen Torus für a = 10 und b = 4, siehe Abbildung 21.

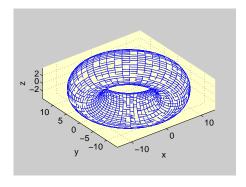


Abbildung 21: Ein Torus

Aufgabe 53 (Parametrisierte Flächen) Welche Fläche entsteht durch den folgenden Aufruf:

```
ezsurf('2*cos(u)*cos(v)',...
'2*sin(u)*cos(v)','2*sin(v)',...
```

$$[-pi/2,pi/2]$$
), axis equal

Lösung: Es entsteht eine Halbkugel. ◎ ◎

Aufgabe 54 (Parametrisierte Flächen) Welche Fläche entsteht durch den folgenden Aufruf:

Lösung: Es entsteht eine Wendelfläche, siehe Abbildung 22. Eine Wendelfläche entsteht,

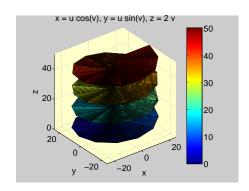


Abbildung 22: Wendelfläche

wenn eine Strecke um eine zu ihr orthogonale Achse geschraubt wird. Wir nehmen an, dass diese Achse gleich der z-Achse des Koordinatensystems ist. Hat die Strecke die Länge l und ist ϕ der Winkel, den sie mit der positiven x-Achse einschliesst, so ist eine Parameterdarstellung der Wendelfläche durch

$$x = lt \cos \phi, \ y = lt \sin \phi, \ z = \frac{c}{2\pi},$$

für $0 \le \phi \le \pi, \ 0 \le t \le 1$ gegeben. Der Parameter c bezeichnet die Ganghöhe, das heißt den Höhengewinn bei einer vollen Umdrehung. \odot

40.6. Implizite Kurven

Mit der Funktion ezplot lassen sich auch implizit definierte Kurven darstellen. Der folgende Script gibt einige Beispiele:

```
subplot(2,3,1)
ezplot('x^2/15^2+y^2/9^2-1',...
[-15,15]
hold on; plot([-12,12],[0,0],'ro')
hold off; grid; pause;
subplot(2,3,2)
ezplot('-x^2/4+y^2/25-1',[-15,15])
grid; pause;
subplot(2,3,3)
ezplot('x^4+y^4-14^4',[-15,15])
grid; pause;
subplot(2,3,4)
ezplot('y^2-x^3/9+6*x+10',[-15,15])
grid; pause;
subplot(2,3,5)
f = @(x,y) (x.^2+y.^2).^2-14^2*...
    (x.^2-y.^2);
ezplot(f,[-15,15])
grid; pause;
subplot(2,3,6)
ezplot('abs(x)+abs(y)-14',...
[-15,15]); grid;
```

40.7. Implizite Flächen

Implizit definierte Flächen können ebenfalls dargestellt werden. Hier ein Beispiel, siehe Abbildung 23.

```
[x,y,z] = meshgrid(-2:0.1:2);
p = patch(isosurface(x,y,z,v,0));
```

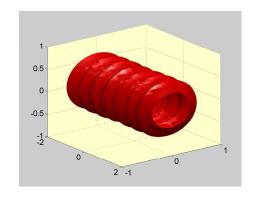


Abbildung 23: Implizite Fläche

```
view(34,40), grid on,
set(p,'FaceColor','red'),
set(p,'EdgeColor','None'),
camlight, lighting gouraud,
```

40.8. Koordinatenachsen skalieren

Beachten Sie, dass Matlab die x- und y-Achse (und natürlich auch die z-Achse im 3D Fall) automatisch skaliert. Wollen Sie diesen Automatismus nicht, so können Sie mit axis (doc axis) "per Hand" die Achsen begrenzen. Zum Beispiel erzeugt axis([-4,4,-2,2]) ein Koordinatensystem, dessen x-Achse von -4 bis 4 und deren y-Achse von -2 bis 2 begrenzt ist. Wenn Sie möchten, dass die x- und y-Achse gleich lang, also quadratisch sind, dann müssen Sie axis square eingeben (Quadratische Bildfläche). Wollen Sie dagegen, dass die xund y-Achse die gleiche Skalierung haben, so geht das mit dem Befeht axis equal. Die $v = \sin(10*x)/8 + \text{sqrt}(z.^2 + y.^2) - 0.5$; Abbildung 24 zeigt dies anhand des Bereichs $[-4, 4] \times [-2, 2]$.

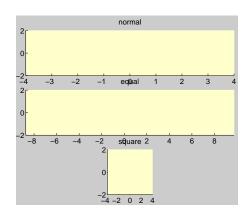


Abbildung 24: Skalierungen

Um logarithmische Skalierungen zu erhalten, stehen spezielle Funktionen zur Verfügung, siehe Tabelle 20.

Name	Beschreibung
loglog	Logarithmisches KO-System
semilogx	x-Achse logarithmisch
semilogy	y-Achse logarithmisch

Tabelle 20: Logarithmische Skalierungen

Aufgabe 55 (Skalierungen) Zeichnen Sie den Graph der Funktion $y = 3e^{-1/2x}$, $x \in \mathbb{R}$ in einem rechtwinkligen Koordinatensystem von x = 0 bis x = 10, wobei die y-Achse logarithmisch skaliert sein soll.

Lösung: Mit

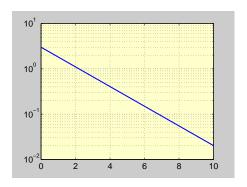


Abbildung 25: $y = 3e^{-1/2x}$, $x \in \mathbb{R}$

Aufgabe 56 (Skalierungen) Zeichnen Sie den Graph der Funktion $y = \sqrt{x}$, $x \in \mathbb{R}$ in einem rechtwinkligen Koordinatensystem von x = 1 bis x = 1000, wobei die x- und y-Achse logarithmisch skaliert sein soll.

Lösung: Mit

```
>> x = 1:1000;
>> y = sqrt(x);
>> loglog(x,y); grid;
```

erhalten wir die Abbildung 26. Der Graph der

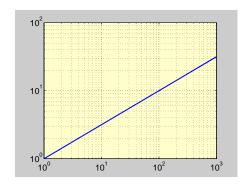


Abbildung 26: $y = \sqrt{x}, x \ge 1$

Quadratwurzelfunktion ist wie erwartet eine

40.9. Zwei y-Achsen

Mit der Funktion plotyy können wir Datensätze mit zwei y-Achsen zeichnen; die eine Achse links, die Andere rechts. Als Beispiel betrachten wir die Funktion $y = 3e^{-1/2x}$, $x \in \mathbb{R}$. Zeichnet man die Graph dieser Funktion in ein gewöhnliches kartesisches Koordinatensystem mit gleicher Skalierung, so erhält man den typischen Verlauf einer abfallenden Exponentialfunktion. Skaliert man die y-Achse jedoch logarithmisch, so ist der Graph eine Gerade. Wir plotten diese beiden Kurven mit der Funktion plotyy nun in eine Figur. Nach den Eingaben

```
>> x = linspace(0,10);
>> y = 3*exp(-0.5*x);
>> plotyy(x,y,x,y,'plot','semilogy')
```

erhalten wir die Abbildung 27. Die linke y-

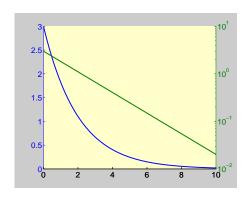


Abbildung 27: Zwei Skalierungen

Achse ist gewöhnlich linear skaliert, während die rechte y-Achse logarithmisch skaliert ist.

40.10. Koordinatentransformationen

Mit der Funktion cart2pol können kartesische Koordinaten in Polarbzw. Zylinderkoodinaten transformiert werden. Die Funktion pol2cart transformiert umgekehrt Polarbzw. Zylinderkoordinaten in kartesische Koordinaten. Mit cart2sph bzw. sph2cart können Transformationen von kartesischen zu Polarkoordinaten und umgekehrt durchgeführt werden.

Zum Zeichnen von Polarkoordinaten ist die Funktion polar geeignet.

40.11. Spezielle Grafikfunktionen

Weitere Grafikfunktionen findet man in der Tabelle 21. Insbesondere zur Darstellung statistischer Daten sind diese von großer Bedeutung.

Name	Beschreibung
bar	Balkendiagramm (vertikal)
barh	Balkendiagramm (horizontal)
bar3	3D-Balkendiagramm (vertikal)
bar3h	3D-Balkendiagramm (horizontal)
hist	Histogramm
pie	Kreisdiagramm
stem	Punkte mit Linien

Tabelle 21: Weitere Grafikfunktionen

40.12. Vektorfelder visualisieren

Mit der Funktion quiver (doc quiver, help quiver) können Sie Vektorfelder darstellen. Das Vektorfeld $f(x, y) = (-y, x), (x, y) \in \mathbb{R}^2$

ist in Abbildung 28 dargestellt. Das Vektorfeld

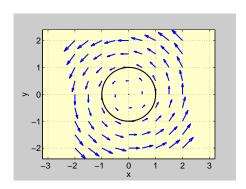


Abbildung 28: Vektorfeld

wurde mit

>> axis equal

erzeugt. Jeder Vektor ist tangential zu einem Kreis um den Ursprung. Die Länge des Vektors ist durch den Radius des Kreises gegeben. Das Vektorfeld kann als Geschwindigkeitsfeld eines Rades interpretiert werden, das sich gegen den Uhrzeigersinn dreht.

Aufgabe 57 (Vektorfeld) Stellen Sie das Vektorfeld

$$f: \mathbb{R}^2 \to \mathbb{R}^2$$
$$(x, y) \mapsto (x, y)$$

dar.

Lösung: Die Anweisungen

erzeugen die Abbildung 29. © Lösung: Die Anweisungen

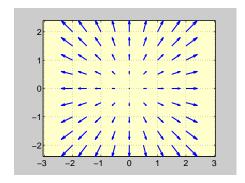


Abbildung 29: Vektorfeld

Die Funktion quiver3 stellt das dreidimensionale Analogon zur Funktion quiver dar. Damit lassen sich also dreidimensionale Vektorfelder darstellen.

Aufgabe 58 (Vektorfeld) Stellen Sie mit der Funktion quiver3 das Vektorfeld

$$f: \mathbb{R}^3 \to \mathbb{R}^3$$
$$(x, y, z) \mapsto (2, 0, 1)$$

dar.

Lösung: Die Anweisungen

Aufgabe 59 (Vektorfeld) Stellen Sie mit der Funktion quiver3 das Vektorfeld

$$f: \mathbb{R}^3 \to \mathbb{R}^3$$
$$(x, y, z) \mapsto (x, y, z)$$

dar.

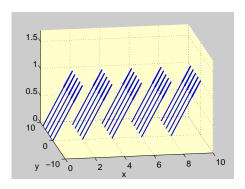


Abbildung 30: Vektorfeld

```
>> [X,Y,Z] = meshgrid(-8:4:8);
>> quiver3(X,Y,Z,X,Y,Z,0)
>> [X,Y,Z] = meshgrid(-2:1:2);
>> quiver3(X,Y,Z,X,Y,Z,0)
>> xlabel('x'), ylabel('y'),
```

erzeugen die Abbildung 31. \odot \odot

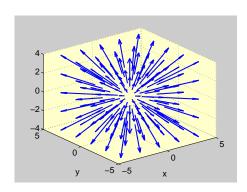


Abbildung 31: Vektorfeld

40.13. Grafiken importieren, exportieren und drucken

Zum Einlesen von Pixel-Grafiken bietet Mat-LAB die Funktion imread. Der Befehl image gibt die eingelesene Grafik in einer Figure aus. So erzeugen die folgenden Befehlszeilen die Figure in Abbildung 32, die das in jpg-Format vorliegende Bild gg.jpg enthält:

```
>> A = imread('gg.jpg','jpeg');
>> image(A)
```

Mögliche Formate sind bmp, cur, usw., siehe doc imread (help imread). Mit dem Befehl print (doc print) können Sie die Figure anschließend drucken oder in eine Datei exportieren.



Abbildung 32: Das bin ich

Hat man eine Grafik erzeugt, so will man diese häufig in einem bestimmten Grafikformat abspeichern und gegebenenfalls in ein Satz- oder Textverarbeitungssystem einbinden. Soll das Grafikformat Encapsulated PostScript sein (Dateiendung: eps), so geht das zum Beispiel wie folgt:

```
%-Erzeuge eine Grafik.
surf(peaks(30));
%-Setze den Hintergrund blau.
set(gcf,'Color','blue');
set(gcf,'InvertHardCopy','off');
%-Exportiere in File MyFile.eps
print -depsc2 MyFile.eps
```

Alternativ können Sie die Grafik auch über das Graphical User Interface exportieren. Dies geht wie folgt:

- Erzeugen Sie eine Figure, zum Beispiel mit surf(peaks(30));.
- 2. Wählen Sie aus dem *Edit* Menü die *Figure Properties*. Dies spricht die *Property Editor* Dialogbox an.
- 3. Wählen Sie den *Style* Panel, skrollen Sie auf die Farbe *Blue*, klicken Sie unten auf *Apply* und dann auf *OK*.
- 4. Wählen Sie aus dem *File* Menü das *Page Setup*. Dies spricht die *Page Setup* Dialogbox an.
- 5. Wählen Sie den Axes und Figure tap und klicken Sie auf Keep screen background color, damit Matlab nicht auf den weißen Hintergrund umschaltet, wenn Sie die Grafik expotieren.
- 6. Klicken Sie auf OK.
- 7. Wählen Sie aus dem File Menü den Unter-

- punkt *Export* aus, um die *Export* Dialogbox zu erzeugen.
- 8. Wählen Sie den Dateityp *EPS Level 2 Color*, geben Sie den Dateinamen *MyFile.eps* ein und speichern Sie Ihre Figur unter diesem Namen.

In Matlab werden vordefinierte Pixelbilder zur Verfügung gestellt. Mit dem Befehl

```
>> load gatlin
```

werden die Daten der Datei gatlin.mat geladen. Die Befehle

- >> image(X)
- >> colormap(map)
- >> axis image

erzeugen die Abbildung 33 eine Fotografie aus

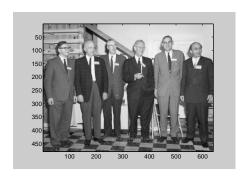


Abbildung 33: Pioniere der Mathematik

dem Jahr 1964 von sechs Pionieren der Numerischen Mathematik. Von links nach rechts handelt es sich um die Wissenschaftler Wilkinson, Givens, Forsythe, Householder, Henrici und Bauer.

Das Farbbild eines Clowns ist in Abbildung 34 zu sehen. Es kann analog der Da-

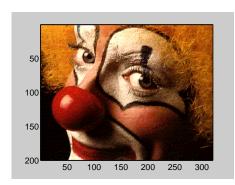


Abbildung 34: Clown

tei gatlin.mat aus der Datei clown.mat erzeugt werden. In [5] habe ich dieses Bild (als Grautonbild) für eine Anwendung der Singulärwertzerlegung in der Bildverabeitung verwendet.

Wir fassen die besprochenen Funktionen in Tabelle 22 zusammen.

Name	Bedeutung
imread	Pixel-Grafik einlesen
image	Pixel-Grafik in Figure plotten
print	speichern oder drucken
saveas	speichern

Tabelle 22: Grafiken importieren usw.

40.14. Animationen

Um Animationen herzustellen und durchzuführen, gibt es die Funktion movie, siehe doc movie (help movie).

40.15. Handle Graphics

Das Grafiksystem von Matlab stellt sogenannte Low-Level-Funktionen zur Verfügung, mit denen alle Aspekte des Grafiksystems kontrolliert werden können. Damit besteht die Möglichkeit, detaillierte Grafiken zu generieren. Die Kommandos set und get erlauben es, jedes Grafikobjekt anzusprechen. Mit doc graphics (help graphics) erhalten Sie eine komplette Übersicht über alle zur Verfügung stehenden Kommandos und Funktionen.

40.16. Graphical User Interface (GUI)

Das Matlab-Grafiksystem verfügt außer der Handle-Graphics über eine weitere objektorientierte Grafikkapazität: *Graphical User Interface (GUI)*. Damit hat man die Möglichkeit, Sliders, Buttons, Menüs und andere Grafikobjekte zu erzeugen, um so interaktive Benutzerschnittstellen zu generieren. Hierzu steht eine GUI-Entwicklungsumgebung zur Verfügung, siehe guide. Für weitere Einzelheiten siehe doc uicontrol (help uicontrol).

40.17. Geometrische Körper

Zum Zeichnen der geometrischen Körper Ellipsoid, Kugel und Zylinder stehen die Funktionen ellipsoid, sphere und cylinder zur Verfügung.

Mit [X,Y,Z]=ellipsoid(xm, ym, zm, xr, yr, zr, n) werden die Koordinatenwerte eines Ellisoids berechnet, die mit 3D-Grafikfunktionen dann geplottet werden können. Die ersten drei Argumente

xm, ym, zm legen das Zentrum und die Argumente xr, yr, zr die Halbachsen fest. Das optionale Argument n (Voreinstellung n=20) bestimmt die Auflösung bzw. die Dimension der 3D-Koordinatenarrays X, Y und Z.

Die Funktionen sphere und cylinder können ähnlich eingesetzt werden.

40.18. Flächenstücke (Patches)

patch ist eine *Low-Level-Funktion* zur Erzeugung von Flächenstücken (patches). Ein Patch-Objekt besteht aus Polygonen, die durch Eckpunkte definiert werden. Patches können verwendet werden, um reale Objekte geometrisch zu modellieren, wie zum Beispiel Flugzeuge oder Automobile.

Ein Beispiel zeigt Abbildung 35. Die gerade

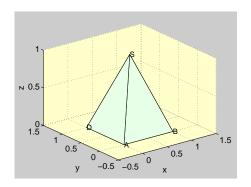


Abbildung 35: Pyramide

Pyramide mit der quadratischen Grundfläche *A*, *B*, *C*, *D* und der Spitze *S* wurde mit den folgenden Anweisungen erzeugt.

```
%-Ecken.
E = [0 0 0; 1 0 0; 1 1 0; 0 1 0;
0.5 0.5 1];
```

Aufgabe 60 (Geometrie) Gegeben ist die dreiseitige Pyramide mit der Grundfläche $A=(-12,0,-2),\ B=(12,-12,-2),\ C=(0,12,-2)$ und der Spitze S=(-4,4,6). Die Koordinaten sind bezüglich eines kartesischen Koordinatensystems gegeben. Eine Dreieckspyramide besitzt eine Inkugel, das heißt eine Kugel, die alle vier Seitenflächen der Dreieckspyramide berührt. Diese Dreieckspyramide besitzt die Inkugel mit dem Radius r=3 und dem Mittelpunkt M=(-3,3,1). Stellen Sie die Pyramide und die Inkugel dar.

Lösung: Das folgende Script löst das Problem.

Die Abbildung 36. zeigt die Situation grafisch.

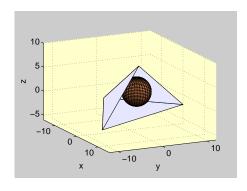


Abbildung 36: Pyramide mit Inkugel

41. Vergleichsoperatoren, Vergleichsfunktionen

Vergleichsoperatoren und Vergleichsfunktionen dienen dazu, zwei Matrizen elementweise hinsichtlich einer bestimmten Eigenschaft zu vergleichen. In Abhängigkeit davon, ob diese Eigenschaft besteht oder nicht, geben sie dann einen entsprechenden Wahrheitswert zurück, der in Bedingungen für Schleifen oder Verzweigungen weiterverwendet werden kann.

In Matlab gibt es – wie auch in C/C++, aber anders als etwa in Pascal – keinen expliziten Datentyp, der die Wahrheitswerte wahr und falsch speichern kann. Statt dessen wird ein numerischer Wert ungleich 0 als wahr und der Wert 0 als falsch betrachtet. Vergleichsoperatoren haben, hinter den arithmetischen, vor den logischen Operatoren, die zweithöchste Priorität bei der Abarbeitung von Ausdrücken.

42. Logische Operatoren und logische Funktionen

Logische Operatoren existieren im Prinzip in allen allgemein verwendbaren Programmiersprachen. Sie dienen dazu, Wahrheitswerte miteinander zu verknüpfen. In den meisten Sprachen haben sie Namen wie AND, oR und NOT und sind damit Schlüsselwörter für den Compiler. In MATLAB – wie auch in C/C++ – ist dies nicht der Fall; die logischen Operatoren sind hier aus Sonderzeichen aufgebaut.

42.1. Logische Operatoren

Werden Matrizen mit logischen Operatoren verknüpft, so geschieht dies komponentenweise. Die Tabelle 23 zeigt die logischen Operatoren.

Logische Operatoren	Beschreibung
&	logisches und
	logisches oder
~	logisches міснт

Tabelle 23: Logische Operatoren

Verknüpft man zwei Matrizen mit einem logischen und, so ist die entsprechende Ergebniskomponente 1, wenn die beiden Komponenten von Null verschieden sind.

```
>> x = [1 0 2 3 0 4];
>> y = [5 6 7 0 0 8];
>> x & y
ans =
1 0 1 0 0 1
```

42.2. Logische Funktionen

Verknüpft man zwei Matrizen mit einem exklusiven oder, so ist die entsprechende Ergebniskomponente 1 (WAHR), wenn eine der beiden Komponenten von Null verschieden ist. Andererseits ist die Ergebniskomponente 0 (FALSCH), wenn beide Komponenten 0 oder beide ungleich 0 sind.

```
>> x = [1 0 2 3 0 4];

>> y = [5 6 7 0 0 8];

>> xor(x,y)

ans =

0 1 0 1 0 0
```

Darüber hinaus gibt es zusätzliche Funktionen, die die Existenz spezieller Werte oder Bedingungen testen und ein logisches Resultat zurückgeben. Weitere Informationen über logische Operatoren und Funktionen findet man mit doc ops (help ops).

Aufgabe 61 (Logische Operatoren) Gegeben seien die Variablen a=5.5, b=1.5 und k=-3. Bestimmen Sie die Ergebnisse der folgenden Ausdrücke. Überprüfen Sie Ihre Resultate dann in MATLAB.

- (a) a < 10.0
- (b) a+b >= 6.5
- (c) k = 0
- (d) b-k > a
- (e) (a == 3*b)
- (f) -k <= k+6
- (g) a<10 & a>5
- (h) $abs(k)>3 \mid k<b-a$

43. Steuerstrukturen

The guts of Matlab are wirtten in C. Much of Matlab ist also written in Matlab, because it's a programming language.

Cleve Moler, 1999.

Programmiersprachen und programmierbare Taschenrechner erlauben es, den Ablauf eines Programms zu steuern. Man spricht von Steuerstruktur. MATLAB bietet vier Möglichkeiten, den sequentiellen Ablauf durch Verzweigungen und Schleifen zu ändern. Dies sind:

- for-Schleifen
- while-Schleifen
- Verzweigungen mit if
- Verzweigungen mit switch

43.1. for-Schleife

Das folgende Beispiel erzeugt in einer for-Schleife die ersten 20 Fibonacci-Folgenglieder.

```
>> f(1) = 0; f(2) = 1;
>> for i=3:20
```

```
f(i) = f(i-1)+f(i-2);
end
```

Der Zeilenvektor f beinhaltet die Zahlenwerte. Bekannterweise nähern sich die Quotienten zweier benachbarter Fibonacci-Zahlen der Zahl $(\sqrt{5} - 1)/2$. Sie können das mit f(1:19)./f(2:20) nachvollziehen.

Will man innerhalb einer Schleife eine Matrix (Vektor) erzeugen, so wie in dem Beispiel

```
>> for k = 1:11
 x(k) = (k-1)*(1/10);
end
```

so gibt es zwei Gründe dafür, die Matrizen zuvor mit zeros zu initialisieren.

- Durch zeros kann man festlegen, ob man einen Zeilen- oder Spaltenvektor erzeugen möchte bzw. welche Größe die Matrix haben soll. Dadurch wird man gezwungen, explizit über die Orientierung und Größe des Vektors bzw. der Matrix nachzudenken, und vermeidet so Fehler beim Operieren mit diesen.
- 2. Der Speichermanager hat durch diese Initialisierung weniger Arbeit. Betrachten wir hierzu obige erste for-Schleife und wie die Variable x zu einem 11-dimensionalen Zeilenvektor wird. Im ersten Schleifendurchlauf ist x ein Vektor der Länge 1 (ein Skalar). Im zweiten Durchlauf weist x(2) den Speichermanager an, x zu einem zweidimensionalen Vektor zu machen. Im dritten Durchlauf wird der Speichermanager durch x(3) angewiesen, x in einen Vektor der Länge 3 umzuformen. Dies setzt sich fort, bis das Ende der

for-Schleife erreicht ist und x 11 Koordinaten hat. Es ist eine Konvention in Mat-LAB, dass durch diese Konstruktionsweise ein Zeilenvektor entsteht.

Es ist daher effizienter, obige erste for-Schleife wie folgt zu programmieren:

```
>> x = zeros(1,11);
>> for k = 1:11
x(k) = (k-1)*(1/10);
end
```

Will man eine Matrix in einer Schleife erzeugen, dann sollte man sie zuvor initialisieren.

43.2. while-Schleife

In einer while-Schleife berechnen wir die Summe der ersten 100 Zahlen.

Der folgende Matlab-Code berechnet die Vektoren $z_{k+1} = Az_k$ mit Startvektor $z_0 = (1,0)$ und der stochastischen Matrix

$$\boldsymbol{A} = \left[\begin{array}{cc} 0.8 & 0.3 \\ 0.2 & 0.7 \end{array} \right].$$

Anschließend werden die Zustandsvektoren gezeichnet. Die Anzahl der Iterationen können Sie eingeben.

```
A = [0.8 0.3; 0.2 0.7];
z = [1;0]; x = z; k = 0;
kend = input('Iterationen: ');
while k < kend
    z = A*z;
    x = [x z];
    k = k+1;
end
plot(0:kend,x,'o-'), grid</pre>
```

43.3. if-Anweisung

Im folgenden Beispiel wird die Anweisung disp('a ist gerade') nur dann ausgeführt, wenn a durch 2 teilbar ist.

```
>> if ( rem(a,2) == 0 )
disp('a ist gerade')
end
```

43.4. switch-Anweisung

Hat im folgenden Beispiel die Variable x den Wert -1, so wird x ist -1 auf dem Bildschirm ausgegeben. Entsprechendes geschieht bei den anderen Fällen.

```
switch x
case -1
disp('x ist -1');
case 0
disp('x ist 0');
case 1
disp('x ist 1');
otherwise
disp('x ist ein anderer Wert');
```

end

Weitere Informationen über Steuerstrukturen findet man mit doc lang (help lang).

44. m-Files

Bisher wurden Anweisungen zeilenweise eingegeben und von Matlab verarbeitet. Diese interaktive Arbeitsweise ist unzweckmäßig für Algorithmen, die mehrere Programmzeilen benötigen und wieder verwendet werden sollen. Hierfür eignen sich sogenannte m-Files, die mit einem Editor erzeugt werden und unter einem Filenamen mit dem Kürzel .m abgespeichert werden. Es gibt zwei Arten von m-Files: die *Script-Files* und die *Function-Files*.

44.1. Script-Files

Ein Script-File (Kommando-File) ist eine Folge von gewöhnlichen Matlab-Anweisungen. Die Anweisungen in einem Script-File werden ausgeführt, wenn man den File-Namen ohne das Kürzel angibt. Ist zum Beispiel der File-Name versuch.m, so gibt man einfach versuch ein. Variablen in einem Script-File sind global, siehe Abschnitt 45. Auch kann ein Script-File einen anderen m-File aufrufen. Script-Files haben keine Ein- und Ausgabeargumente.

Wir geben ein einfaches Beispiel eines Script-Files (ErstesScript).

```
%-Script: ERSTES-SCRIPT
Daten = [2,10,1,12,-2,3,2]';
sort(Daten)
```

```
mean(Daten)
median(Daten)
std(Daten)
```

Dieser Script-File sortiert den Spaltenvektor Daten mit der Funktion sort und berechnet den arithmetischen Mittelwert (mean), den Median (median) und die Standardabweichung (std) der Werte im Datenvektor Daten. Gibt man den Namen ErstesScript nach dem MATLAB Prompt ein, so erhält man folgende Ausgabe:

```
ans =
-2
1
2
2
3
10
12
ans =
4
ans =
2
ans =
5.0662
```

Das folgende Beispiel zeigt ein Eigenwert-Roulette, welches darauf beruht, abzuzählen wieviele Eigenwerte einer reellen Zufallsmatrix reell sind. Ist die Matrix A reell und von der Ordnung 8, dann gibt es 0,2,4,6 oder 8 reelle Eigenwerte (die Anzahl muss gerade sein, weil komplexe Eigenwerte in komplexkonjugierten Paaren auftreten). Die beiden Zeilen

```
A = randn(8);
```

```
sum((abs(imag(eig(A))) < 0.0001));
```

erzeugen eine zufällig normalverteilte 8×8 -Matrix und zählen, wieviel Eigenwerte reell sind. Dies ist so realisiert, dass geprüft wird, ob der Imaginärteil dem Betrag nach kleiner als 10^{-4} ist. Jeder Aufruf erzeugt nun eine andere Zufallsmatrix und man erhält somit unterschiedliche Ergebnisse. Um ein Gefühl dafür zu bekommen, welche der fünf Möglichkeiten am wahrscheinlichsten ist, kann man folgenden Script ausführen.

```
%-Script-File: EIGENWERTROULETTE
n = 1000;
Anzahl = zeros(n,1);
for k=1:n
    A = randn(8);
    Anzahl(k) = ...
    sum(abs(imag(eig(A))) < 0.0001);
end
hist(Anzahl,[0 2 4 6 8]);
h = findobj(gca,'Type','patch');
set(h,'FaceColor','r',...
'EdgeColor','w')</pre>
```

Dieser Script-File erzeugt 1000 Zufallszahlen und zeichnet ein Histogramm der Verteilung der Anzahl der reellen Eigenwerte. Die Abbildung 37 zeigt ein mögliches Resultat. Wollen Sie sehen, wie Ihr Script-File den von Ihnen geschriebenen Code abarbeitet, so geben Sie echo on ein. Mit echo off können Sie den Vorgang wieder rückgängig machen.

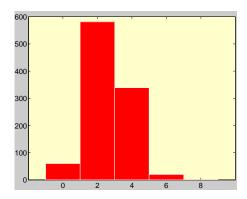


Abbildung 37: Histogramm zum Script

44.2. Function-Files

Wenn Sie intensiver mit MATLAB arbeiten, dann werden Sie bald feststellen, dass es nicht für alle ihre Wünsche eingebaute Funktionen oder Kommandos gibt. In diesem Fall können Sie sich aber Ihre eigene Funktion schreiben und damit die Funktionalität von MAT-LAB erweitern. Mit Hilfe von Function-Files (MATLAB-Funktionen, Funktionen) können Sie den Matlab-Funktionsvorrat erweitern. Variablen in Function-Files sind lokale Variablen. Function-Files haben demnach ihren eigenen nach außen nicht sichtbaren Workspace. Die Übergabe einzelner Variablen erfolgt über eine Parameterliste im Funktionsaufruf. Ein Function-File entspricht der subroutine bzw. function in der Programmiersprache Fortran, FUNCTION in C/C++ und procedure bzw. Function in Pascal. Haben Sie eine Funktion in Form eines Function-Files geschrieben, so können Sie diesen genauso aufrufen, wie die eingebauten Matlab-Funktionen. Für das Schreiben eines Function-Files sind verschiedene Dinge zu

beachten.

Damit ein File ein Function-File ist, muss er mit dem Schlüsselwort function beginnen, dann folgen die Ausgabeargumente (in eckigen Klammern), der Funktionsname und schließlich die Eingabeargumente (in runden Klammern). Die Form aller Function-Files ist

```
function [Out_1,...,Out_n] =
Name(In_1,...,In_m)
Irgendwelche Anweisungen >
```

wobei Name der vom Anwender anzugebende Funktionsname ist. Der Funktionsname (hier Name) muss auch der Filename sein, unter der der Function-File abgespeichert wird. Es ist möglich, dass ein Function-File keine Ausgabe- und/oder Eingabeargumente hat.

Die folgende Funktion ist ein Beispiel für einen Function-File.

```
function [V,D,r] = MatrixEig(A)
[m,n] = size(A);
if m==n
[V,D] = eig(A);
r = rank(A);
else
disp('Fehler: Die Matrix muss
quadratisch sein!')
end
```

Schreiben Sie sich die obigen Anweisungen in eine Datei mit dem Namen MatrixEig.m und definieren Sie eine Matrix A im Matlab-Workspace. Führen Sie

```
>> [V,D,r] = MatrixEig(A)
```

aus, dann stehen in der Matrix V die Eigenvektoren, in der Hauptdiagonalen von D die Eigenwerte und in r der Rang der Matrix A.

V,D,r sind die Ausgabe- und A das Eingabeargument der Funktion MatrixEig. Der von uns geschriebene Function-File MatrixEig ruft mehrer eingebaute Matlab-Funktionen auf: size, eig, usw.

Functions können sowohl von anderen Functions als auch von Scripts aufgerufen werden. Es ist eine Stärke von Matlab-Funktionen, mit einer unterschiedlichen Anzahl von Übergabeparameter zurechtzukommen. Hierzu exisitieren innerhalb einer Funktion die Variablen nargin und nargout (number argument in/out), die die Anzahl der übergebenen Parameter enthalten. Zusätzlich stehen noch die Befehle inputname zum Ermitteln der Variablennamen der Übergabeargumente und nargchk zum Überprüfen der korrekten Anzahl der übergebenen Argumente zur Verfügung. Siehe doc nargin, doc nargout, doc inputname und doc nargchk für entsprechende Beispiele. Für weitere Informationen über Funktionen siehe doc function (help function) und Abschnitt 51.

44.3. Namen von m-Files

Für m-Files gelten bezüglich der Namenswahl die gleichen Regeln wie für Variablen, siehe Abschnitt 28. Insbesondere wird zwischen Klein- und Großbuchstaben unterschieden. Weitere Infos finden Sie unter doc general und doc lang.

44.4. Editieren von m-Files

Sie haben zwei Möglichkeiten, um einen m-File zu erzeugen und zu editieren. Sie können einerseits mit einem Editor Ihrer Wahl arbeiten oder andererseits den eigebauten Mat-LAB-Editor/Debugger verwenden. Diesen können Sie mit dem Befehl edit aktivieren oder durch Anklicken der Menüoptionen File-New oder File-Open.

44.5. Zur Struktur eines m-Files

m-Files sollten, um sie auch nach längerer Zeit wieder verwenden zu können, gut dokumentiert sein (Software Engineering). Professionelle m-Files haben daher folgende Form:

```
function [Out,...] = Name(In,...)
% H1-Zeile
% Help Text
Help Text usw.
< Irgendwelche Anweisungen >
```

Die erste Zeile legt den Funktionsnamen und die Ein- und Ausgabeparameter fest. Dies haben wir bereits besprochen. Die zweite Zeile ist die sogenannte H1-Zeile. In ihr wird in einer Zeile das Programm beschrieben. Dies ist die erste Zeile, die auf dem Bilschirm ausgegeben wird, wenn Sie help Name eingeben. Außerdem sucht die lookfor-Funktion nur in dieser H1-Zeile und gibt nur diese aus. Geben Sie sich daher besondere Mühe, diese H1-Zeile kurz und prägnant zu schreiben. In weiteren anschließenden Zeilen können Sie den m-File weiter dokumentieren. Script-Files werden genauso gehandelt, dann aber entfällt natürlich die erste Zeile mit dem Schlüsselwort function. Weitere Informationen finden Sie in [17].

44.6. Blockkommentare

Seit Matlab 7 können Sie auch Blöcke von Codes auskommentieren, indem Sie den Code in zwei spezielle Kommentarzeielen setzen:

<Block> steht für eine beliebige Anzahl von Codezeilen. Matlab betrachtet dann alle Zeilen zwischen %{ und %} als Kommentar. Blockkommentare können auch geschachtelt werden.

44.7. Übungsaufgaben

Aufgabe 62 (Function-File) Schreiben Sie einen Function-File, der von einem Vektor \mathbf{x} den arithmetischen und geometrischen Mittelwert berechnet und die Werte zurückgibt.

Lösung: Die folgende Funktion tut das Gewünschte.

```
function [aM,gM] = Mittelwerte(x)
aM = mean(x);
gM = prod(x)./length(x);
```

Aufgabe 63 (Function-File) Schreiben Sie einen Function-File, der die Sprungfunktion (Einheitssprungfuktion, Heaviside-Funktion)

$$h(t) = \begin{cases} 0 & t < 0 \\ 1 & t \ge 0 \end{cases}$$

berechnet. Zeichnen Sie diese Funktion im Intervall [-2,2]. Vergleichen Sie die Funktion h mit der in Matlab definierten Funktion

heaviside. Wo gibt es Unterschiede? Lösung: Die Funktion h ist durch den Function-File

```
function y = h(t)
y = ( t>=0 );
```

definiert und mit den Anweisungen

erhält man die Abbildung 38.

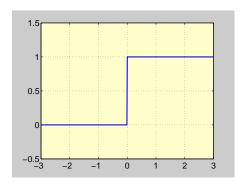


Abbildung 38: Sprungfunktion

Die Funktion heaviside aus Matlab ist an der Stelle t = 0 undefiniert, das heißt Matlab ordnet dem Nullpunkt NaN zu. \odot \odot

Aufgabe 64 (Function-File) Zeichnen Sie die Graphen der verschobenen Sprungfunktionen h(t+T), $t \in \mathbb{R}$ und h(t-T), $t \in \mathbb{R}$ für T=2 im Intervall [-4, 4].

Lösung: Die Funktionen können mit dem Function-File

```
function y = hT(t,T)
y = ( t>=-T );
```

berechnet werden. Die Anweisungen

```
subplot(2,1,1)
fplot(@hT,[-4,4,-0.5,1.5],[],...
[],[],2),
grid, title('h(t+2)')
subplot(2,1,2)
fplot(@hT,[-4,4,-0.5,1.5],[],...
[],[],-2),
grid, title('h(t-2)')
```

erzeugen die Abbildung 39. ©

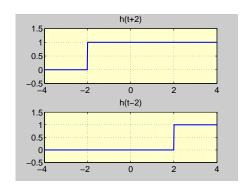


Abbildung 39: Verschobene Sprungfunktionen

Aufgabe 65 (Function-File) Schreiben Sie jeweils einen Function-File, um die folgenden stückweise definierten Funktionen zu berechnen:

(a)
$$rect(t) = \begin{cases} 1 & |t| \le 0.5 \\ 0 & sonst \end{cases}$$

(b)
$$ramp(t) = \begin{cases} 0 & t < 0 \\ t & sonst \end{cases}$$

(c)
$$g(t) = \begin{cases} 0 & t < 0 \\ \sin(\frac{\pi t}{2}) & 0 \le t \le 1 \\ 1 & 1 < t \end{cases}$$

Zeichnen Sie die Graphen der Funktionen im Intervall [-2, 2].

Lösung: Mit Hilfe der Sprungfunktion h lassen sich diese Funktionen geschlossen darstellen. Es ist $\operatorname{rect}(t) = h(t+0.5) - h(t-0.5),$ $t \in \mathbb{R}$, $\operatorname{ramp}(t) = th(t),$ $t \in \mathbb{R}$ und $g(t) = \sin(\frac{\pi t}{2})(h(t) - h(t-1)),$ $t \in \mathbb{R}$. Daher lassen sich diese drei Funktionen in Matlab wie folgt berechnen.

```
function y = rect(t)
y = h(t+0.5)-h(t-0.5);
```

```
function y = ramp(t)
y = t.*h(t);
```

```
function y = g(t)
y = sin(pi*t/2)*(h(t)-h(t-1))+...
(t>=1);
```

Die Anweisungen

```
subplot(3,1,1), title('rect')
fplot(@rect,[-2,2,-0.5,2]), grid
subplot(3,1,2), title('ramp')
fplot(@ramp,[-2,2,-0.5,2]), grid
subplot(3,1,3), title('g')
fplot(@g,[-2,2,-0.5,2]), grid
```

erzeugen die Abbildung 40. ◎

Aufgabe 66 (Function-File) Schreiben Sie einen Function-File, um folgende Funktion zu berechnen:

$$f(x) = \begin{cases} x & x < 0 \\ x^2 & 0 \le x < 2 \\ 4 & x \ge 2 \end{cases}$$

Testen Sie Ihre Funktion für die Werte x = -2, 1.5, 2 und 6. Zeichnen Sie die Funktion f

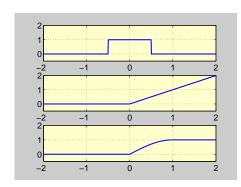


Abbildung 40: Graph der Funktionen

mit fplot über dem Intervall [-3, 3]! *Lösung:* Der folgende Function-File definiert die Funktion f.

```
function y = f(x)

y1 = x.*(x<0);

y2 = x.^2.*((x<2)-(x<=0));

y3 = 4*(x>=2);

y = y1+y2+y3;
```

Aufgabe 67 (Function-File) Schreiben Sie einen Function-File, um die Funktion $f(t) = t^{\frac{1}{3}}$, $t \in \mathbb{R}$ zu berechnen. Benutzen Sie diesen, um die Funktion f im Intervall [0,1] zu zeichnen.

Lösung: Mit dem Function-File

```
function y = f(t)
y = t.^(1/3);
```

Aufgabe 68 (Function-File) Zeichnen Sie die konstante Funktion f(x) = 5, $x \in \mathbb{R}$ im Inter-

vall [0, 2].

Lösung: Eine Möglichkeit ist:

```
>> f = @(x) 5;
>> fplot(f,[0,2])
```

Aufgabe 69 (Horner-Methode) Implementieren Sie die Horner-Methode in Matlab. Zunächst skalar und dann vektoriell.

Lösung: Hierzu wählen wir die folgende Darstellung des Polynoms

$$p(x) = a_1 + a_2x + a_3x^2 + \dots + a_nx^{n-1}$$

und nennen die skalare Matlab-Funktion Horner.

```
function p = Horner(a,x)
n = length(a);
p = a(n);
for k=n-1:-1:1
p = p*x+a(k);
end
```

Für das Polynom $p(x) = x^2 + 3x + 2$ liefert der Aufruf Horner([2,3,1],2) den Wert 12=p(2).

Nun die vektorielle Version.

```
function p = HornerVektoriell(a,x)
n = length(a);
p = a(n)*ones(size(x));
for k=n-1:-1:1
p = x.*p+a(k);
end
```

Für das Polynom $p(x) = x^2 + 3x + 2$ liefert der Aufruf Horner([2,3,1],[2,3]) die Werte 12=p(2) und 20 = p(3).

45. Globale und lokale Variablen

Die Variablen innerhalb jeder Funktion sind lokal, die in Script-Files sind global. Globale Variablen können aber auch mit global definiert werden. Um auf diese Variablen zugreifen zu können, muss diese Definition sowohl im Haupt-Workspace als auch in der Function erfolgen. Angezeigt werden die globalen Variablen mit whos global, gelöscht werden sie mit clear global.

46. Namenstest

Mit der Funktion exist können Sie überprüfen, ob ein Name bereits exsitiert. Mit iskeyword kann festgestellt werden, ob ein Schlüsselwort vorliegt. Eine Liste aller Schlüsselwörter erhält man mit

```
>> iskeyword
ans =
     'break'
     'case'
     'catch'
     'continue'
     'else'
     'elseif'
     'end'
     'for'
     'function'
     'global'
     'if'
     'otherwise'
     'persistent'
     'return'
     'switch'
```

```
'try'
'while'
```

47. Wie man effiziente Programme schreibt

Schleifen werden in Matlab ineffizient ausgeführt. Deshalb sollten Sie diese vermeiden, wo immer es geht. Nahezu alle Matlab-Funktion akzeptieren "vektorielle" Argumente, so dass man auf Schleifen häufig tatsächlich auch verzichten kann.

Angenommen Sie wollen die ersten 100 natürlichen Zahlen aufsummieren (nicht aber die Formel n(n + 1)/2 verwenden). In einer skalaren Programmiersprache wie zum Beispiel in C/C++ würde man wie folgt vorgehen:

```
int s = 0;
int n;
for (n=1;n<101;++n)
{
   s = s+n;
}
print(''%d\n'',s);</pre>
```

Die analoge Version dieses kleinen Programms in Matlab wäre:

```
s = 0;
for n=1:100
    s = s+n;
end
s
```

Dieser skalare Matlab-Code kann effizienter und übersichtlicher geschrieben werden:

```
N = 1:100;
s = sum(N)
```

Der erste Befehl erzeugt den Zeilenvektor N = [1,2,...,100]. Die zweite Anweisung summiert die Koordinaten des Vektors N auf. sum ist eine eingebaute Matlab-Funktion und verträgt Vektoren als Argumente. Viele Matlab-Funktionen können Vektoren oder Matrizen als Argumente verarbeiten. Dies lässt eine vektorielle Verarbeitung zu.

Die rationale Funktion

$$f(x) = \left(\frac{1 + \frac{x}{24}}{1 - \frac{x}{12} + \frac{x^2}{384}}\right)^8$$

stellt im Intervall [0, 1] eine Approximation an die Exponentialfunktion e dar.

Der folgende Script zeigt, wie man die Auswertung dieser Funktion in einer skalaren Programmiersprache wie zum Beispiel FORT-RAN oder C/C++ vornehmen müsste.

```
n = 200;
x = linspace(0,1,n);
y = zeros(1,n);
for k=1:n
    y(k) = ((1+x(k)/24)/(1-x(k)/12...
    +(x(k)/384)*x(k)))^8;
end
```

In Matlab aber sind Vektoroperationen erlaubt, das heißt die for-Schleife kann durch eine einzige vektorwertige Anweisung ersetzt werden. Der folgende Script-File zeigt eine vektorielle Implementierung der Funktion f. Der Übersichtlichkeit wegen splitten wir den Term f(x) in mehrere Terme auf.

```
n = 200;
x = linspace(0,1,n);
Zaehler = 1 + x/24;
Nenner = 1 - x/12 + (x/384).*x;
Quotient = Zaehler./Nenner;
y = Quotient.^8;
```

Um der Variablen y die entsprechenden Funktionswerte von f zuzuweisen, werden verschiedene bekannte und weniger bekannte Vektoroperationen durchgeführt: Vektoraddition, Vektorsubtraktion, skalare Multiplikation, punktweise Vektormultiplikation, punktweise Vektordivision und punktweise Vektorexponentiation.

Betrachten wir den Script-File genauer. Mat-LAB erlaubt es, einen Vektor mit einem Skalar zu multiplizieren. Dies zeigt der Term x/24. Dort wird jede Koordinate des Vektors x durch die Zahl 24 dividiert bzw. mit 1/24 multipliziert. Das Ergebnis ist ein Vektor mit der gleichen Länge und Orientierung (Zeile oder Spalte) wie der Vektor x. Im obigen Script ist x ein Zeilenvektor und somit ist x/24 ebenfalls ein Zeilenvektor. Durch die Anweisung 1+x/24 wird zu jeder Koordinate des neuen Vektors x/24 1 hinzuaddiert und der Variablen Zaehler zugeordnet. Dies ist natürlich keine Vektorraumoperation, aber eine nützliche Matlab-Eigenschaft. Wir betrachten nun die Variable Nenner. Hierbei bedeutet die Operation (x/384).*x eine punktweise Vektormultiplikation. das heißt jede Koordinate von x/384 wird mit jeder Koordinaten des Vektors x multipliziert. Beachten Sie, dass die Vektoren die gleiche Länge haben. Zum Ergebnis wird 1 hinzuaddiert und von jeder Koordinate x/12 subtrahiert, bevor das Ergebnis der Variablen Nenner zugeordnet wird. Die Anweisung Quotient = Zaehler./Nenner bedeutet punktweise Division, das heißt, jede Komponente des Vektors Zaehler wird durch die entsprechende Koordinate des Vektors Nenner dividiert und anschließend der Variablen Quotient zugeordnet. Schließlich wird durch die Anweisung y = Quotient. 8 punktweise potenziert, das heißt jede Koordinate des Vektor Quotient wird mit 8 potenziert, bevor das Resultat der Variablen y zugeordnet wird.

Die Matlab-Funktion vectorize vektorisiert einen String automatisch. Hierzu betrachten wir folgendes Beispiel. Sind die Vektoren Zaehler und Nenner wie folgt definiert:

```
Zaehler = [1 2 3];
Nenner = [4 5 6];
```

dann ist Zaehler/Nenner keine vektorielle Divsion, da vor dem Divisionszeichen / der Punkt . fehlt. Die Anweisung

```
>> vectorize('Zaehler/Nenner')
ans =
Zaehler./Nenner
```

erzeugt die gewünschte Syntax. Analog setzt der Befehl vectorize vor den Zeichen * und ^ einen Punkt und ermöglichst somit eine vektorisierte Operation.

Aufgabe 70 (Effiziente Programme) Vektorisieren Sie den String '($(1+x/24)/(1-x/12+x^2/384)$) '8' und zeigen Sie grafisch, dass f eine Approximation im Intervall [0,1] an die Exponentialfunktion e ist.

Lösung: Die folgenden Zeilen lösen die

Aufgabe.

```
y = vectorize('((1+x/24)/(1-x/12+x^2/384))^8');
x = linspace(0,1,20);
vs = vectorize('((1+x/24)/(1-x/12+x^2/384))^8');
y = eval(vs,x);
plot(x,exp(x),x,y,'ro')
legend('e','f')
```

Nicht alle Berechnungen sind jedoch vektorisierbar. In diesen Fällen muss man auf Schleifen zurückgreifen. Um diese Berechnungen jedoch schneller auszuführen, sollte man die Ausgabematrizen mit Nullen vorbesetzen (Preallokieren). Wir erläutern an einem Beispiel, was damit gemeint ist. Angenommen es sei die Matrix

$$\mathbf{A} = \left[\begin{array}{cc} 0.8 & 0.3 \\ 0.2 & 0.7 \end{array} \right]$$

gegeben und wir wollen die Eigenwerte der Matrizen A^k für $k=1,2,\ldots,10$ berechnen. Die Eigenwerte sollen spaltenweise in der Ausgabematrix E gespeichert werden. Das folgende Script realisiert dies:

```
E = zeros(2,10);
for k=1:10
   E(:,k) = eig(A^k);
end
```

Mit der Anweisung E = zeros(2,10); haben wir die Ausgabematrix E mit Nullen vorbesetzt. Hätten wir dies nicht getan, so müsste Matlab in jedem Schleifendurchlauf die Größe der Matrix E durch Hinzunahme einer weiteren Spalte verändern, was sich durch eine länge-

re Ausführungszeit bemerkbar machen würde. Darüber hinaus hat das Vorbesetzen der Ausgabematrizen den Vorteil, dass man sich bereits vorher über die Größe und Orientierung Gedanken machen muss, was zu disziplinärem Programmierstil erzieht.

Wir fassen noch einmal zusammen: Das Ersetzen einer Schleife durch eine Vektoroperation nennt man *Vektorisierung* und hat drei Vorteile:

- Geschwindigkeit. Viele eingebaute Matlab-Funktionen werden schneller ausgeführt, wenn man anstelle eines mehrfachen Aufrufs als Argument einen Vektor übergibt.
- Übersichtlichkeit. Es ist übersichtlicher, ein vektorisiertes Matlab-Script zu lesen, als das skalare Gegenstück.
- Ausbildung. Im wissenschaftlichen Rechnen ist man bei verschiedenen Rechnern interessiert, vektorisierte Algorithmen zu entwickeln und zu implementieren. Matlab unterstützt dies.

Somit gilt:

Vermeiden Sie Schleifen in Matlab, wann immer dies möglich ist.

Vektorisieren Sie ihre Rechnungen, wann immer dies möglich ist.

Aufgabe 71 (Programmierung) Schreiben Sie die folgenden MATLAB-Zeilen vektoriell.

```
for x = 1:10
    y = sqrt(x);
end
```

Lösung: Die vektorielle und effizientere Programmierung ist:

```
x = 1:10;
y = sqrt(x);
```

Aufgabe 72 (Programmierung) Plotten Sie die Funktion

```
f(x) = 2\sin(x) + 3\sin(2x) + 7\sin(3x)+ 5\sin(4x), \quad x \in \mathbb{R}
```

im Intervall [-10, 10].

Lösung: Wir nutzen die vektorielle Programmierfähigkeit von Matlab, sowie etwas Matrizenrechnung und können f so wie folgt effizient darstellen.

Der Graph der Funktion f ist in Abbildung 41 zu sehen. \odot ... \odot

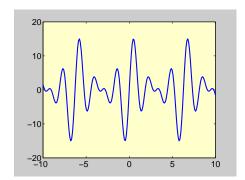


Abbildung 41: Graph von f

48. Lineare Algebra (Teil 1)

Diesen Abschnitt habe ich im Zusammenhang mit meinem Buch [4] geschrieben. Viele Beispiele sind daraus. Im Folgenden werden wir je nach Bedarf zwischen numerischen und symbolischen Funktionen hin und her wechseln, das heißt zwischen Grundfunktionen aus Matlab und zwischen Funktionen aus der *Symbolic Toolbox*, siehe auch Abschnitt 67.

48.1. Lineare Gleichungssysteme und Matrizen

Zu Matrizen siehe auch Abschnitt 36. Das lineare Gleichungssystem

$$x + y + 2z = 9$$
$$2x + 4y - 3z = 1$$
$$3x + 6y - 5z = 0$$

aus Beispiel 1.6 in [4] können wir in Matlab mit dem \-Operator (Backslash-Operator) wie folgt lösen.

```
>> A = [1 1 2; 2 4 -3; 3 6 -5];

>> b = [9; 1; 0];

>> x = A\b

ans =

1.0000

2.0000

3.0000
```

Alternativ können wir die Funktion rref verwenden. Diese transformiert eine gegebene Matrix in reduzierte Zeilenstufenform.

```
>> rref([A b])
ans =
```

```
1 0 0 1
0 1 0 2
0 0 1 3
```

Die Lösung kann nun direkt abgelesen werden. Die Funktion inv berechnet die Inverse einer Matrix. Daher finden wir die Lösung auch wie folgt (Satz 1.8)

```
>> x = inv(A)*b
x =
1.0000
2.0000
3.0000
```

Wir bestätigen Satz 1.9 in MATLAB.

```
>> syms a b c d

>> A = [a,b; c,d];

>> inv(A)

ans =

[ d/(a*d-b*c), -b/(a*d-b*c)]

[ -c/(a*d-b*c), a/(a*d-b*c)]
```

Die Funktion inv kann sowohl für numerische als auch für symbolische Rechnungen verwendet werden. Dies gilt auch für viele andere Funktionen.

Die Null- und Einheitsmatrizen jeder Größe können durch die Funktionen zeros und eye erzeugt werden. Die Funktion 1u berechnet die LU-Faktorisierung einer Matrix.

Aufgabe 73 (Lineare Systeme) Berechnen Sie die allgemeine Lösung des linearen Glei-

chungssystems

$$\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}.$$

$$A \qquad x \qquad b$$

Lösung: Die allgemeine Lösung ist die eindeutige Lösung x = (-1, 2, 2). Dies zeigen die folgenden Matlab-Zeilen.

Die Lösung ist eindeutig, sonst wäre die Matrix *A* singulär und Matlab hätte mit einer Fehlermeldung geantwortet. Hier noch alternativ die symbolische Lösung.

Aufgabe 74 (Lineare Systeme) Berechnen Sie die allgemeine Lösung des linearen Gleichungssystems

$$4x_1 - 8x_2 = 12$$
$$3x_1 - 6x_2 = 9$$
$$-2x_1 + 4x_2 = -6.$$

Lösung: Die allgemeine Lösung können wir aus

ablesen. Demnach ist: x = (3,0) + t(2,1), $t \in \mathbb{R}$ die allgemeine Lösung. Dieses Beispiel zeigt, dass auch überbestimmte Systeme unendlich viele Lösungen haben können. $\odot \dots \odot$

Aufgabe 75 (Lineare Systeme) Berechnen Sie die allgemeine Lösung des linearen Gleichungssystems

$$-x_2 + 3x_3 = 1$$
$$3x_1 + 6x_2 - 3x_3 = -2$$
$$6x_1 + 6x_2 + 3x_3 = 5.$$

Lösung: Die allgemeine Lösung ist die eindeutige Lösung x = (3, -2, -1/3), denn es gilt

```
>> A = [0 -1 3; 3 6 -3; 6 6 3];

>> b = [1; -2; 5];

>> sym(A)\b

ans =

[ 3]

[ -2]

[ -1/3]
```

Aufgabe 76 (Lineare Systeme) Berechnen Sie die allgemeine Lösung des linearen Glei-

chungssystems

$$x_1 + x_2 + 2x_3 = 8$$

 $-x_1 - 2x_2 + 3x_3 = 1$
 $3x_1 - 7x_2 + 4x_3 = 10$.

Lösung: Die allgemeine Lösung ist die eindeutige Lösung x = (3, 1, 2), denn es gilt

Aufgabe 77 (Lineare Systeme) Berechnen Sie die allgemeine Lösung von Ax = b mit

$$A = \begin{bmatrix} 4 & 1 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 & 4 \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} 4.1 \\ 2.4 \\ 4.2 \\ 2.4 \\ 4.1 \end{bmatrix},$$

wobei es ganz Ihnen überlassen ist, wie Sie die allgemeine Lösung herausbekommen.

Lösung: Die eindeutige Lösung ergibt sich aus den folgenden Anweisungen.

Aufgabe 78 (Beispiel 1.24 in [4]) Berechnen Sie die Inverse der Matrix

$$A = \left[\begin{array}{cc} 1 & -2 \\ 3 & 2 \end{array} \right]$$

Lösung: Es ist

Aufgabe 79 (Inverse) Bestätigen Sie Satz 1.9 aus aus [4].

Lösung: Es ist

was Satz 1.9 in Matlab bestätigt. ◎ ◎

Aufgabe 80 (Inverse) Finden Sie mit Matlab Beispiel 1.26 in [4], so erhält man die Inverse die Inverse der Matrix

$$\mathbf{S} = \left[\begin{array}{ccc} a & b & c \\ d & e & f \\ g & h & i \end{array} \right]$$

Unter welchen Bedingungen existiert diese? Bestätigen Sie mit diesem Ergebnis die Inverse in Beispiel 1.26 aus [4].

Lösung: Die Inverse erhalten wir durch

```
>> syms a b c d e f g h i
>> S = [a b c; d e f; g h i];
>> pretty(inv(S))
 [ei-fh]
  -----
              usw.
     %1
   di-fg
              usw.
 Γ
      %1
 Γ
   -dh+eg
              usw.
Γ
       %1
%1 := a e i - a f h - d b i +
     dch+gbf-gce
```

Die Inverse exisiert genau dann, wenn der Nenner aei - afh - dbi + dch + gbf - gce ungleich Null ist. Ein Vergleich mit

```
>> det(S)
ans =
a*e*i-a*f*h-d*b*i+d*c*h+g*b*f-g*c*e
```

zeigt, dass dies gerade die Determinante von S ist. Setzt man für a = 1, b = 1, ..., i = -5 nach Die Länge des Vektors v = (4, -3) ist

$$\left[\begin{array}{cccc}
2 & -17 & 11 \\
-1 & 11 & -7 \\
0 & 3 & -2
\end{array}\right].$$

Übrigends die Determinante der Matrix ist −1. ◎.......

Aufgabe 81 (Symmetrische Matrizen) Angenommen Sie wollen einen Algorithmus testen, der als Eingabe eine symmetrische Matrix benötigt. Finden Sie eine Möglichkeit eine beliebige symmetrische Matrix in Matlab zu erzeugen.

Lösung: Nach den folgenden Zeilen

```
n = 44; %-zum Beispiel 44.0rdnung
A = rand(n);
B = A + A';
```

steht in der Matrix B eine symmetrische Matrix aus $\mathbb{R}^{44\times44}$. \odot \odot

48.2. Vektoren in der Ebene und im Raum

Zu Vektoren siehe auch Abschnitt 34. Vektoroperationen lassen sich in MATLAB einfach ausführen. Die Summe der beiden Vektoren u =(2, 3, -1) und v = (3, -4, 2) ist (Beispiel 2.3 in [4])

```
>> u = [2;3;-1]; v = [3;-4;2];
>> u+v
ans =
     5
    -1
```

```
>> norm([4;-3])
ans =
5
```

Das Skalarprodukt der beiden Vektoren u = (0, 0, 1) und v = (0, -2, 2) haben wir in Beispiel 2.8 ausgerechnet; hier die Bestätigung mit der dot-Funktion

```
>> u = [0;0;1]; v = [0;-2;2];
>> dot(u,v)
ans =
2
```

Den Winkel aus Beispiel 2.9 berechnen wir wie folgt

Beachten Sie, Matlab rechnet im Bogenmaß. Die Zeile

```
>> 180*phi/pi
ans =
60.0000
```

liefert den Winkel im Gradmaß. Wir berechnen den Projektionsvektor aus Beispiel 2.12

```
>> p = (u'*a)/(norm(a)^2)*a

p =

2.7692

-0.6923

2.0769
```

Mit der cross-Funktion können wir das Kreuzprodukt berechnen. Wir bestätigen damit Beispiel 2.13

```
>> u = [1;2;-2]; v = [3;0;1];
>> cross(u,v)
ans =
2
-7
-6
```

Das dyadische Produkt ab^{T} von Vektor a = (1,2) und b = (4,1,4,3) berechnet sich wie folgt

```
>> a = [1;2]; b = [4;1;4;3];
>> a*b'
ans =
4 1 4 3
8 2 8 6
```

Aufgabe 82 (Skalarprodukt) Berechnen Sie jeweils das Skalarprodukt der angegebenen Vektoren:

- (a) $\mathbf{a} = (1,0); \mathbf{b} = (0,1)$
- (b) $\boldsymbol{a} = (1,0,0); \boldsymbol{b} = (0,1,0)$
- (c) $\mathbf{a} = (1, 1, 1); \mathbf{b} = (-2, -2, -2)$
- (d) a = (2, 2, 2); b = (3, 3, 3)

Lösung:

Das bedeutet, dass die beiden Vektoren senkrecht aufeinander stehen.

0

Das bedeutet, dass die beiden Vektoren senkrecht aufeinander stehen.

48.3. Analytische Geometrie von Geraden und Ebenen

Wir berechnen den Normalenvektor aus Beispiel 3.12 in [4]

```
>> cross([1 0 2],[0 -5 8])
ans =
10 -8 -5
```

Mehr Geometrie und Matlab finden Sie bei [20].

48.4. Reelle Vektorräume und Unterräume

Zur Berechnung einer orthonormalen Basis des Nullraumes und des Spaltenraumes stehen uns die Funktionen null und orth zur Verfügung. Die Funktion null liefert auch von symbolischen Matrizen eine Basis; diese ist im Allgemeinen aber nicht orthogonal. Um symbolisch eine Spaltenraumbasis zu erhalten, können wir colspace verwenden. Auch hier gilt, dass die-

se im Allgemeinen nicht orthogonal ist. Wir betrachten Zahlenbeispiele mit der Matrix *A* aus Beispiel 4.23.

```
>> A = sym([1,2; 3 6]);
>> null(A)
ans =
[ -2]
[ 1]
```

Auch Matlab hat t = 1 gewählt, um einen Basisvektor für den Nullraum von A zu bestimmen. Eine (numerische) Basis der Länge 1 erhält man mit

```
>> A = [1,2; 3 6];
>> null(A)
ans =
-0.8944
0.4472
```

Die beiden Anweisungen

```
>> A = sym([1,2; 3 6]);
>> colspace(A)
ans =
[ 1]
[ 3]
```

liefern eine Basis des Spaltenraumes von A. Eine Basis der Länge 1 erhält man mit orth.

```
>> A = [1,2; 3 6];
>> orth(A)
ans =
-0.3162
-0.9487
```

Wir bestätigen Satz 4.12.

```
>> null(A)
ans =
```

```
[ -2]
[ 1]
>> Z = rref(A)
Z =
[ 1, 2]
[ 0, 0]
>> null(Z)
ans =
[ -2]
[ 1]
```

und

```
>> colspace(A')
ans =
[ 1]
[ 2]
>> colspace(rref(A)')
ans =
[ 1]
[ 2]
```

Den Rang einer Matrix können wir mit der Funktion rank bestimmen. Wir berechnen den Rang der Matrix *A* aus Beispiel 4.26.

```
>> A = [1 2 4 0 1; 0 1 2 0 0;...
0 0 0 1 0; 0 0 0 0 0];
>> rank(A)
ans =
3
```

In Beispiel 4.28 haben wir alle vier Fundamentalräume an Hand der Matrix *A* diskutiert. Wir bestätigen nun die dort gefundenen Resultate, indem wir zu jedem Fundamentalraum eine Basis berechnen.

```
>> A = sym([1,2; 3 6]);
>> null(A)
```

```
ans =
[ -2]
  1]
>> colspace(A)
ans =
[ 1]
[ 3]
>> null(A')
ans =
[ -3]
1]
>> colspace(A')
ans =
[ 1]
[ 2]
```

Basen der Länge 1 erhalten wir wie folgt

```
>> A = [1,2; 3 6];
>> null(A)
ans =
   -0.8944
    0.4472
>> orth(A)
ans =
   -0.3162
   -0.9487
>> null(A')
ans =
   -0.9487
    0.3162
>> orth(A')
ans =
   -0.4472
   -0.8944
```

Der Spaltenraum von A und der Nullraum von A^{T} sind orthogonale Komplemente im \mathbb{R}^m . Hier eine Bestätigung

```
>> A = [1,2; 3 6];
>> orth(A)'*null(A')
ans =
-1.6653e-016
```

bzw. analog im \mathbb{R}^n , nun symbolisch.

```
>> colspace(A')'*null(A)
ans =
0
```

Mit dem Backslash-Operator \ können wir lineare Ausgleichsaufgaben lösen. Wir bestätigen damit die Lösung aus Beispiel 4.34.

```
>> A = [1,0; 1 1; 1 2];
>> b = [6;0;0];
>> A\b
ans =
5.0000
-3.0000
```

Aufgabe 83 (Lineare Unabhängigkeit) Sind die Vektoren a = (1, -2, 3), b = (5, 6, -1) und c = (3, 2, 1) linear unabhängig (Beispiel 4.16 in [4])?

Lösung: Die folgenden Matlab-Zeilen zeigen auf verschiedene Weisen, dass die drei Vektoren linear abhängig sind.

```
>> a = [1,-2,3];

>> b = [5,6,-1];

>> c = [3,2,1];

>> A = sym([a',b',c']);

>> det(A), rank(A)

ans =

0

ans =

2
```

```
>> eig(A)
ans =
 0
 4
 4
>> svd(A)
ans =
 (45+3*97^(1/2))^(1/2)
 (45-3*97^(1/2))^(1/2)
>> rref(A)
ans =
Γ
          0.1/21
Γ
    0.
          1, 1/2]
    0,
          0,
               0]
>> null(A)
ans =
  1
  1
 -2
>> colspace(A)
ans =
1,
       0]
   0,
       1]
  [1, -1]
```

Aufgabe 84 (Basen) Bestimmen Sie von der Matrix

......

$$\mathbf{A} = \left[\begin{array}{cc} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{array} \right]$$

jeweils eine Basis der vier Fundamentalräume. *Lösung:* Hier eine symbolische Lösung. Zuerst Basen des Nullraumes und des Spaltenraumes, dann Basen des transponierten Nullraumes und des Zeilenraumes.

```
>> null(A), colspace(A)
ans =
[ empty sym ]
ans =
   1,
       0]
       1]
0,
[-1,
       2]
>> null(A'), colspace(A')
ans =
  1]
[ -2]
   1]
ans =
[ 1, 0]
[ 0, 1]
```

Für eine numerische Lösung benützt man die Funktionen null und orth; diese liefern sogar orthonormale Basen.

Aufgabe 85 (LGS und Rang) Wie können Sie (in Matlab) entscheiden, ob ein lineares Gleichungssystem eine Lösung hat?

Aufgabe 86 (LGS und Rang) Wenn ein lineares Gleichungssystem eine Lösung hat, wie können Sie dann (in Matlab) entscheiden, ob es genau eine oder mehrere gibt?

Aufgabe 87 (Ausgleich) Berechnen Sie die Näherungslösung (im Sinne der linearen Ausgleichsrechnung) von Ax = b und den orthogonalen Projektionsvektor p von b auf den

Spaltenraum von A des Systems

$$\begin{bmatrix} 1 & 1 \\ -1 & 1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 7 \\ 0 \\ -7 \end{bmatrix}$$

$$A \qquad x \qquad b$$

Lösung: Die folgende Zeilen geben die Lösungen.

Aufgabe 88 (Ausgleich) Die Datenpunkte

t_i	b_i
0	3.825
0.2	4.528
0.4	4.746
0.6	4.873
0.8	4.865
1.0	4.813

sollen im Sinne der linearen Ausgleichsrechnung an eine Funktion der Form

$$f(t, \boldsymbol{x}) = x_1 + x_2 t + x_3 \cos(t)$$

angepasst werden. Zeichnen Sie die Datenpunkte und die Ausgleichsfunktion, nachdem Sie die Lösung berechnet haben. Lösung:

```
t = [0 \ 0.2 \ 0.4 \ 0.6 \ 0.8 \ 1]';
b = [3.825 \ 4.528 \ 4.746 \ 4.873...
     4.865 4.813]';
A = [ones(length(t),1),t,cos(t)];
x = A \setminus b
x =
   -1.0428
    3.1516
    4.9265
plot(t,b,'o','MarkerSize',10)
tt = linspace(min(t), max(t));
f = x(1)+x(2)*tt+x(3)*cos(tt);
hold on
plot(tt,f)
```

Die Abbildung 42 zeigt die grafische Darstel-

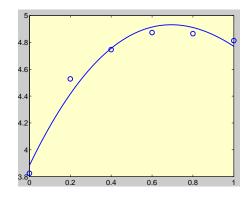


Abbildung 42: Zu Aufgabe 88

lung. \odot \odot

48.5. Determinanten

trix steht die Funktion det zur Verfügung, so- geschieht über die LU-Faktoren.

wohl numerisch als auch symbolisch. Wir berechnen die Determinante von

$$\boldsymbol{A} = \left[\begin{array}{cc} a & b \\ c & d \end{array} \right].$$

```
>> syms a b c d
>> A = [a b; c d];
>> det(A)
ans =
a*d-b*c
```

Die folgende Zeile bestätigt das Ergebnis aus Beispiel 5.10.

Aufgabe 89 (Determinanten) Berechnen Sie die Determinante von

$$\mathbf{A} = \left[\begin{array}{rrr} 1 & 1 & 2 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{array} \right]$$

Lösung: Hier die symbolische Lösung.

```
\Rightarrow A = sym([1 1 2; 2 4 -3; 3 6 -5]);
>> det(A)
ans =
-1
```

Zur Berechnung der Determinante einer Ma- Die numerische Berechnung der Determinante

48.6. Eigenwerte und Eigenvektoren

Die Funktion eig hift uns beim Lösen einer Eigenaufgabe. Es können damit sowohl Eigenwerte als auch Eigenvektoren berechnet werden, sowohl numerisch als auch symbolisch. Wir bestätigen Beispiel 6.2.

```
>> A = sym([1,1; -2,4]);

>> [X,D] = eig(sym(A))

X =

[ 1, 1]

[ 1, 2]

D =

[ 2, 0]

[ 0, 3]
```

Das charakteristische Polynom einer Matrix erhalten wir mit der Funktion poly; siehe Beispiel 6.2.

```
>> A = sym([1 1; -2 4]);
>> poly(A)
ans =
x^2-5*x+6
```

Eine symmetrische Matrix ist immer diagonalisierbar. Wir überprüfen Beispiel 6.12.

```
>> A = [4,2,2; 2,4,2; 2,2,4];

>> [X,D] = eig(sym(A))

X =

[ 0, 1, 1]

[ -1, -1, 1]

[ 1, 0, 1]

D =

[ 2, 0, 0]

[ 0, 2, 0]

[ 0, 0, 8]
```

Wir machen die Probe.

```
>> X*D*inv(X)
ans =
[ 4, 2, 2]
[ 2, 4, 2]
[ 2, 2, 4]
```

Für eine symbolische Matrix versucht die Funktion eig ein exaktes Eigenwertsystem zu berechnen. Aus der Galois-Theorie wissen wir, dass dies jedoch nicht immer für Matrizen der Ordnung 5 oder größer möglich ist. Es sei

$$\mathbf{A} = \begin{bmatrix} -6 & 12 & 4 \\ 8 & -21 & -8 \\ -29 & 72 & 27 \end{bmatrix}$$

gegeben. Wir wollen Matlab verwenden, um folgende Frage zu beantworten: Ist A reell diagonalisierbar, das heißt, können wir eine Matrix X finden, so dass $X^{-1}AX$ diagonalisierbar ist? Die Eigenwerte von A finden wir mit eig(sym(A)). Matlab antwortet mit:

```
ans =
[ 3]
[ -2]
[ -1]
```

Da die Eigenwerte von A alle reell und voneinander verschieden sind, kann A diagonalisiert werden, das heißt, es gibt eine Matrix X mit

$$\mathbf{X}^{-1}\mathbf{A}\mathbf{X} = \left[\begin{array}{rrr} -1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 3 \end{array} \right].$$

Wie findet man *X*? Die Spalten von *X* können als Eigenvektoren zu den Eigenwerten -1,-2 und 3 gewählt werden. Gibt man

ein, so gibt Matlaß in der Diagonalmatrix D die Eigenwerte zurück und die Spalten von X sind dazugehörige Eigenvektoren:

```
X =
  1,
Г
       [0, -4]
  0, 1, -2]
  1, -3,
            1]
D =
            0]
[-2,
       0,
   0,
       3,
            0]
[0, -1]
```

Schließlich kann man mit inv(X)*A*X überprüfen, ob $X^{-1}AX$ die gewünschte Diagonalmatrix ist. Wir erhalten das gewünschte Ergebnis:

Aufgabe 90 (Eigensysteme) Gegeben ist die 4×4 Matrix

$$\mathbf{A} = \begin{bmatrix} 12 & 48 & 68 & 88 \\ -19 & -54 & -57 & -68 \\ 22 & 52 & 66 & 96 \\ -11 & -26 & -41 & -64 \end{bmatrix}.$$

Zeigen Sie mit Matlab, dass die Eigenwerte von A reell und voneinander verschieden sind. Finden Sie eine Matrix X, so dass $X^{-1}AX$ diagonal ist.

Lösung:

```
52
         22
                 66 96;
        -11 - 26 - 41 - 64;
\gg [X,D] = eig(sym(A))
X =
1, -12/11,
                         0,
                                 -2]
2]
      -2,
                1,
                         1,
                                 -2]
1, -14/11,
                        -2,
7/11,
                         1,
                                  1]
D =
 -16,
               0,
                    0]
         0,
0.
        -4.
               0,
                    0]
0,
              -8,
                    0]
    0.
0, -12
>> inv(X)*A*X
ans =
[-16,
                    0]
               0,
0.
                    0]
        -4.
0.
         0,
              -8,
                    0]
               0, -12
```

Die Anweisung J = jordan(A) berechnet die Jordan-Matrix von A und [V,J] =

jordan(A) zusätzlich die Ähnlichkeitstransformation V. Die Spalten von V sind die verallgemeinerten Eigenvektoren von A.

```
>> A = sym([3/2 1;-1/4 1/2]);
>> [V,J] = jordan(A)
V =
[    1/2,    1]
[ -1/4,    0]
J =
[    1,   1]
[    0,   1]
```

In Matlab gibt es die Funktion eigshow, die das Eigenwertproblem für eine (2, 2)-Matrix

visualisiert. Beim Aufruf sieht man den Einheitsvektor x = (1,0) und mit der Maus kann man diesen auf dem Einheitskreis bewegen. Zur gleichen Zeit wird Ax angezeigt und bewegt. Manchmal ist Ax vor x, manchmal aber auch hinter x. Ax kann zu x auch parallel sein. In diesem Augenblick gilt $Ax = \lambda x$ und x ist ein Eigenvektor. Der Eigenwert λ bestimmt sich aus der Länge und Richtung von Ax. Die zur Verfügung stehenden Matrizen illustrieren folgende Möglichkeiten:

- 1. Es gibt keine (reellen) Eigenvektoren. x und Ax sind nie parallel. Die Eigenwerte und Eigenvektoren sind komplex. Es gibt keinen Eigenraum von \mathbb{R}^2 .
- 2. Es gibt nur eine Gerade, auf der die Eigenvektoren liegen. Dies ist der einzige Eigenraum von \mathbb{R}^2 .
- 3. Es gibt Eigenvektoren mit zwei unabhängigen Richtungen. Es gibt genau zwei Eigenräume von \mathbb{R}^2 . Dies ist typisch, der Standardfall.
- Jeder Vektor ist Eigenvektor oder anders gesagt: Jeder eindimensionale Unterraum von ℝ² ist Eigenraum.

Auch die Singulärwertzerlegung kann im Fall m = n = 2 mit der Funktion eigshow visualisiert werden. Für weitere Informationen siehe [6].

Viele in der Praxis auftretenden Eigenwertprobleme haben die Form eines *verallgemeinerten (allgemeinen) Eigenwertproblems*: Gegeben sind zwei quadratische Matrix \boldsymbol{A} und \boldsymbol{B} . Gesucht sind Zahlen λ und Vektoren \boldsymbol{x} , sodass gilt

$$Ax = \lambda Bx$$
.

Für $B \neq E$ ist diese Aufgabenstellung eine Verallgemeinerung der herkömmlichen Eigenwertaufgabe; für B = E reduziert sich das verallgemeinerte Eigenwertproblem auf die Standardeigenwertaufgabe. Verallgemeinerte Eigenwertprobleme treten zum Beispiel in mechanischen Schwingungssystemen auf; dort ist A die Steifigkeitsmatrix und B die Massenmatrix.

Ist *A* oder *B* eine reguläre Matrix, dann kann man das verallgemeinerte Problem auf ein gewöhnliches reduzieren, entweder so

$$(\mathbf{B}^{-1}\mathbf{A})\mathbf{x} = \lambda \mathbf{x}.$$

oder so

$$(\boldsymbol{A}^{-1}\boldsymbol{B})\boldsymbol{x} = \frac{1}{\lambda}\boldsymbol{x}.$$

In Matlab kann auch dieses verallgemeinerte Problem mit der Funktion eig angegangen werden. Die Funktion eigs berechnet ein paar Eigenwerte und Eigenvektoren, wenn gewünscht.

Aufgabe 91 (Eigensysteme) Bestimmen Sie eine orthogonale Eigenvektorenmatrix Q, die die Matrix

$$A = \left[\begin{array}{cc} 3 & 1 \\ 1 & 3 \end{array} \right]$$

diagonalisiert.

Lösung:

Die Matrix

$$Q = \begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

diagonalisiert A. Man erhält sie mit der Funktion eig.

```
>> A = [3 1; 1 3];

>> [Q,D] = eig(A)

Q =

-0.7071  0.7071

0.7071  0.7071

D =

2  0

0  4
```

Probe:

```
>> inv(Q)*A*Q
ans =
2.0000 0
0 4.0000
```

Die symbolische eig-Funktion liefert (nur) eine orthogonale Basis aus Eigenvektoren. \odot . \odot

48.7. Lineare Abbildungen und Matrizen

Wir visualisieren lineare Abbildungen im \mathbb{R}^2 . Diese können durch (2,2)-Matrizen beschrieben werden. Wir zeichnen ein Haus und betrachten das "neue" Haus nach linearen Abbildungen. Die folgende Matlab-Funktion zeichnet ein Haus in der Ebene, indem sie Datenpunkte, die in der Matrix H angegeben werden müssen, miteinander verbindet.

```
function plotHaus(H)
%-Zeichnet ein Haus.
x = H(1,:)'; y = H(2,:)';
plot(x,y,'o',x,y,'r-')
axis([-10 10 -10 10]); axis square;
```

Die beiden Anweisungen

erzeugen das Bild 43.

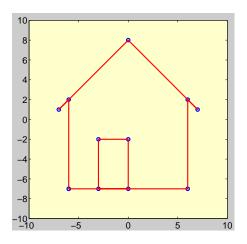


Abbildung 43: Durch die Datenpunkte in H erzeugte Haus

Um nun lineare Abbildungen zu "sehen", müssen wir eine lineare Abbildung als Matrix A definieren und diese mit H multiplizieren. Als Beispiel betrachten wir eine Drehung um 60° und eine Spiegelung an der x-Achse. Im ersten Fall ist

$$A = \begin{bmatrix} \cos 60^{\circ} & -\sin 60^{\circ} \\ \sin 60^{\circ} & \cos 60^{\circ} \end{bmatrix}$$

und im Zweiten

$$\boldsymbol{A} = \left[\begin{array}{cc} 1 & 0 \\ 0 & -1 \end{array} \right].$$

Nach zweimaligem Aufruf von plotHaus(A*H) (zuvor entsprechend A

definieren!) erhalten wir die Bilder 44 und 45.

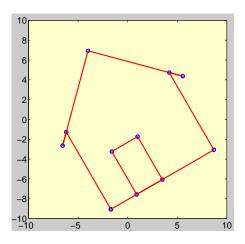


Abbildung 44: Nach Drehung

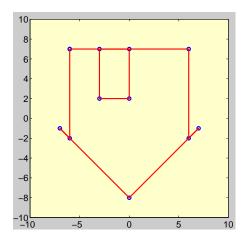


Abbildung 45: Nach Spiegelung

48.8. Matlab-Funktionen für die Lineare Algebra im Überblick

Zusammenfassend habe ich in den Tabellen 24, 25 und 26 die wichtigsten Funktionen aus Mat-LAB zur Linearen Algebra zusammengestellt. Beachten Sie, dass zum Beispiel die Funktionen eig, inv usw. vom Overloading Gebrauch machen. Overloading ist ein Mechanismus, der es erlaubt verschiedene Typen von Funktionsargumenten zu verwenden. So kann A beim Aufruf von eig (A) eine symbolische oder eine numerische Matrix sein.

Symbolische Funktionen		
colspace	Basis für Spaltenraum	
det	Determinante	
diag	Diagonalmatrix	
eig	Eigenwerte und -vektoren	
expm	Exponentialfunktion	
inv	Inverse	
jordan	Jordan-Form	
null	Basis für Nullraum	
poly	Charakteristisches Polynom	
rank	Rang	
rref	Reduzierte Zeilenstufenform	
svd	Singulärwertzerlegung	
tril	Untere Dreiecksmatrix	
triu	Obere Dreiecksmatrix	

Tabelle 24: Lineare Algebra

Numerische Funktionen	
det	Determinante
diag	Diagonalmatrix
eig	Eigenwerte und -vektoren
expm	Exponentialfunktion
inv	Inverse
null	Orth. Basis für Nullraum
orth	Orth. Basis für Spaltenraum
poly	Charakteristisches Polynom
rank	Rang
rref	Reduzierte Zeilenstufenform
svd	Singulärwertzerlegung
tril	Untere Dreiecksmatrix
triu	Obere Dreiecksmatrix

Tabelle 25: Lineare Algebra

Weitere Funktionen		
cross	Kreuzprodukt	
dot	Skalarpordukt	
eye	Einheitsmatrix	
lu	LU-Faktorisierung	
qr	QR-Faktorisierung	
zeros	Nullmatrix	

Tabelle 26: Linearen Algebra

49. Lineare Algebra (Teil 2)

49.1. Lineare Gleichungssysteme (2)

In vielen Anwendungen muss man lineare Gleichungen lösen. Daher ist es in MATLAB besonders einfach, solche zu lösen. Hierzu dient der \-Operator (Backslash-Operator), siehe doc slash (help slash). Auch die Funktion linsolve löst lineare Gleichungssysteme. Unter Verwendung dieser Funktion ist es möglich, lineare Systeme effizienter zu lösen, wenn die Struktur der Koeffizientenmatrix (Systemmatrix) bekannt ist, siehe doc linsolve (help linsolve).

49.1.1. Quadratische Systeme

Ist A eine reguläre (quadratische) Matrix, so gibt es genau eine Lösung des linearen Systems Ax = b und zwar für jede rechte Seite b. Zum Beispiel ist die Matrix

$$A = \left[\begin{array}{cc} 1 & -2 \\ 3 & 2 \end{array} \right]$$

regulär und daher gibt es für b = (1, 11) genau eine Lösung; diese ist x = (3, 1). In Matlab löst man dies in einem Einzeiler, nachdem man A und b eingegeben hat.

Ist die Koeffizientenmatrix A singulär, so erhält man eine Fehlermeldung, auch dann wenn

b im Spaltenraum von A liegt und es nicht nur 49.1.2. Überbestimmte Systeme eine, sondern unendlich viele Lösungen gibt.

Aufgabe 92 (Lineare Systeme) Berechnen Sie die allgemeine Lösung des linearen Gleichungssystems

$$\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}.$$

$$A \qquad x \qquad b$$

Lösung: Die allgemeine Lösung ist die eindeutige Lösung x = (-1, 2, 2). Dies zeigen die folgenden Matlab-Zeilen.

Die Lösung ist eindeutig, sonst wäre die Matrix A singulär und Matlab hätte mit einer Fehlermeldung geantwortet. Hier noch alternativ die symbolische Lösung.

Gibt es mehr Gleichungen als Unbekannte, so nennt man Ax = b überbestimmt. Das System Ax = b hat in der Regel keine Lösung, aber das Ersatzproblem $Ax \cong b$ (die lineare Ausgleichsaufgabe) ist stets lösbar. Entweder hat das Ersatzproblem genau eine oder aber unendlich viele Lösungen, je nachdem ob die Spalten von A linear unabhängig sind oder nicht. Im Fall, dass es unendlich viele Lösungen gibt erhält man durch A\b eine Basislösung. Mit pinv(A)*b erhält man den Lösungsvektor kleinster Länge.

Aufgabe 93 (Ausgleich) Berechnen Sie die Lösung des linearen Ausgleichsproblems $Ax \cong b$ mit

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \quad \text{und} \quad \mathbf{b} = \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix}.$$

Lösung: Die eindeutige Lösung ist x (5, -3). Diese kann auf die folgenden drei Arten berechnet werden.

```
>> A = [1 0; 1 1; 1 2];
>> b = [6; 0; 0];
>> x = A \setminus b
x =
    5.0000
   -3.0000
>> x = pinv(A)*b
x =
\gg x = inv(A'*A)*A'*b
```

-3

©.....©

Aufgabe 94 (Lineare Systeme) Berechnen Sie in Matlab $x = A \setminus b$ und x = pinv(A)*b mit

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ -1 & -2 \end{bmatrix} \quad \text{und} \quad \mathbf{b} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}.$$

Interpretieren Sie die Ergebnisse.

Lösung: Es ist

und

Beides sind spezielle Lösungen der allgemeinen Lösung $\mathbf{x} = (x_1, x_2) = (1 - 2t, 2), t \in \mathbb{R}$ der linearen Ausgleichsaufgabe. $\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$ berechnet eine Basislösung (t = 0.5) und $\mathbf{x} = \mathbf{pinv}(\mathbf{A}) * \mathbf{b}$ die Lösung kleinster Länge (t = 0.4). Da $\mathbf{inv}(\mathbf{A}' * \mathbf{A})$ nicht existiert, kann über diesen Weg keine Lösung berechnet werden. \odot

49.1.3. Unterbestimmte Systeme

Ein lineares System Ax = b heißt unterbestimmt, wenn weniger Gleichungen als Unbekannte vorliegen; in der Regel hat Ax = b dann unendlich viele Lösungen. In diesem Fall wird

durch A\b die Lösung kleinster Länge berechnet, das heißt von allen Lösungen x wird diejenige ermittelt, wo die Länge von x am kleinsten ist. Hat das System Ax = b keine Lösung, so wird eine Basislösung des Ersatzproblems $Ax \cong b$ berechnet.

Aufgabe 95 (Lineare Systeme) Berechnen Sie in Matlab $x = A \setminus b x = pinv(A) *b und x = A'*inv(A*A')*b$

$$A = \begin{bmatrix} 1 & 1 \end{bmatrix}$$
 und $b = \begin{bmatrix} 6 \end{bmatrix}$.

Interpretieren Sie die Ergebnisse.

Lösung: Es ist

und

Dies sind spezielle Lösungen der allgemeine Lösung $\mathbf{x} = (x_1, x_2) = (6 - t, t), t \in \mathbb{R}$ des linearen Gleichungssystems $x_1 + x_2 = 6$. $\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$ berechnet eine Basislösung (t = 0) und $\mathbf{x} = \mathbf{pinv}(\mathbf{A}) * \mathbf{b}$ bzw. $\mathbf{x} = \mathbf{A}' * \mathbf{inv}(\mathbf{A} * \mathbf{A}') * \mathbf{b}$ die Lösung kleinster Länge (t = 3). $\odot \dots \odot$

49.2. Lineare Gleichungssysteme (3)

Lineare Gleichungssysteme lassen sich symbolisch in Matlab mit den Funktionen solve und \ lösen. Darüberhinaus gibt es noch die Maple-Funktionen gausselim und gaussjord, die mit der maple-Funktion angesprochen werden können. Für das maple-Kommando verweise ich sie auf Abschnitt 67.2.

Der Prozess einer Matrix auf reduzierte Zeilenstufenform zu transformieren wird als Gauss-Jordan-Verfahren bezeichnet. Daher haben auch die beiden Funktionen gaussjord und rref die gleiche Wirkung. Das folgende Beispiel zeigt dies:

```
>> maple('gaussjord',...
sym([1 1 2 9; 2 4 -3 1; 3 6 -5 0]))
ans =
[ 1, 0, 0, 1]
[ 0, 1, 0, 2]
[ 0, 0, 1, 3]
```

Die gausselim-Funktion transformiert die erweiterte Matrix auf Zeilenstufenform, wobei die führenden Elemente nicht notwendig gleich 1 sind.

```
>> maple('gausselim',...
sym([1 1 2 9; 2 4 -3 1; 3 6 -5 0]))
ans =
[ 1, 1, 2, 9]
[ 0, 2, -7, -17]
[ 0, 0, -1/2, -3/2]
```

Die backsub-Funktion erlaubt es dann die Lösung durch Rückwärtseinsetzen zu gewinnen:

```
>> maple('backsub',ans)
```

Wir betrachten das folgende unterbestimmte lineare Gleichungssystem

$$-2x_3 + 7x_5 = 12$$
$$2x_1 + 4x_2 - 10x_3 + 6x_4 + 12x_5 = 28$$
$$2x_1 + 4x_2 - 5x_3 + 6x_4 - 5x_5 = -1$$

mit der erweierten Matrix

$$\begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -2 & 0 & 7 & 12 \\ 2 & 4 & -10 & 6 & 12 & 28 \\ 2 & 4 & -5 & 6 & -5 & -1 \end{bmatrix}.$$

Dieses lineare Gleichungssystem hat unendlich viele Lösungen. Mit den Parametern $t \in \mathbb{R}$ und $s \in \mathbb{R}$ können die Lösungen wie folgt parametrisiert angegeben werden:

$$x_1 = 7 - 2x_2 - 3x_4$$

$$x_2 = t$$

$$x_3 = 1$$

$$x_4 = s$$

$$x_5 = 2$$

Mit rref können wir die Lösungen bestätigen:

```
>> Ab = sym([0 0 -2 0 7 12;...
2 4 -10 6 12 28; 2 4 -5 6 -5 -1])
Ab =
0,
         0, -2,
                   0,
                        7,
    2,
         4, -10,
                   6, 12,
2,
         4,
            -5,
                   6, -5,
>> rref(Ab)
ans =
[ 1, 2, 0, 3, 0, 7]
[ 0, 0, 1, 0, 0, 1]
[ 0, 0, 0, 0, 1, 2]
```

Die Lösungen erhalten wir auch wie folgt

```
>> maple('gausselim',Ab)
ans =
                        12,
2,
         4, -10,
                    6,
                              28]
             -2,
                    0,
                         7,
                              12]
0,
                    0, 1/2,
                               1]
>> maple('backsub',ans)
ans =
[ 7-2*_t2-3*_t1,_t2,1,_t1,2]
```

Dagegen liefern die Anweisungen

```
>> A = Ab(:,1:5)
A =
Γ
    0,
          0, -2,
                     0,
                          7]
2.
          4, -10,
                     6,
                         127
          4, -5,
                     6,
    2,
                         -5]
>> b = Ab(:,6)
b =
[ 12]
[ 28]
[-1]
>> A\b
ans =
[ 7]
[ 0]
[ 1]
[ 0]
[ 2]
```

nur die Lösung x = (7, 0, 1, 0, 2). Die Parameter t und s sind hier gleich Null gewählt.

Lösen wir das Beispiel numerisch, so erhalten wir die spezielle Lösung x = (0, 0, 1, 7/3, 2), wie die folgende Zeile zeigt:

```
>> double(A)\double(b)
ans =
0
```

```
0
1.0000
2.3333
2.0000
```

Aufgabe 96 (Lineare Systeme) Berechnen Sie die allgemeine Lösung des linearen Gleichungssystems

$$4x_1 - 8x_2 = 12$$
$$3x_1 - 6x_2 = 9$$
$$-2x_1 + 4x_2 = -6.$$

mit Hilfe der Funktionen gausselim und backsub, siehe Aufgabe 74.

Lösung: Die allgemeine Lösung erhalten wir durch

```
>> A = sym([4 -8; 3 -6; -2 4]);
>> b = sym([12; 9; -6]);
>> maple('gausselim', [A b])
ans =
[ 4, -8, 12]
[ 0, 0, 0]
[ 0, 0, 0]
>> maple('backsub', ans)
ans =
[ 3+2*_t1, __t1]
```

Demnach ist: x = (3,0) + t(2,1), $t \in \mathbb{R}$ die allgemeine Lösung. Mit

```
>> A\b
ans =
3
0
```

erhalten wir die spezielle Lösung (3,0). Hier ist $t = 0. \odot \dots \odot$

Wir betrachten nun ein lineares Gleichungssys-

tem, das keine Lösung hat. Das folgende System ist überbestimmt. Überbestimmte System haben mehr Gleichungen als Variablen und daher in der Regel (generisch) keine Lösung. Tatsächlich ist das System

$$2x_1 - 3x_2 = -2$$
$$2x_1 + x_2 = 1$$
$$3x_1 + 2x_2 = 1$$

inkonsistent. Dies können wir mit der rref-Funktion sofort überprüfen:

```
>> Ab = sym([2 -3 -2; 2 1 1; 3 2 1])
Ab =
[ 2, -3, -2]
[ 2, 1, 1]
[ 3, 2, 1]
>> rref(Ab)
ans =
[ 1, 0, 0]
[ 0, 1, 0]
[ 0, 0, 1]
```

Die letzte Zeile der erweiterten Matrix besagt

$$0x_1 + 0x_2 = 1.$$

Wir testen die anderen Funktionen, die uns zur Lösung linearer Gleichung symbolisch oder numerisch in Matlab zur Verfügung stehen. Das Gauss-Verfahren liefert

```
>> maple('gausselim',Ab)
ans =
[ 2, -3, -2]
[ 0, 4, 3]
[ 0, 0, -7/8]
```

Auch hier erkennt man an der letzten Zeile der Matrix die Unlösbarkeit des linearen Systems,

denn die Gleichung $0x_1+0x_2 = -7/8$ hat keine Lösung. Das Rückwärtseinsetzen bestätigt die Beobachtung

```
>> maple('backsub',ans)
??? Error using ==> sym/maple
Error, (in backsub) inconsistent
system
```

Was liefert A\b?

```
>> A\b
Warning: System is inconsistent.
Solution does not exist.
> In C:\MATLAB6p1\toolbox\symbolic\
@sym\mldivide.m at line 29
ans =
Inf
```

Wie zu erwarten: das System ist inkonsistent. Auch die Funktion solve sagt, dass die Lösungsmenge leer ist:

Numerisch dagegen erhalten wir die Lösung 0.6429:

```
>> double(A)\double(b)
ans =
```

0.6429

Dies ist die eindeutige Lösung des dazugehörigen linearen Ausgleichsproblems. Für weitere Einzelheiten hierzu verweisen wir auf [7].

Die Tabelle 27 zeigt zusammenfassend die

Matlab-Syntax	Bedeutung
\	$L\ddot{o}st Ax = b$
rref	Red. Zeilenstufenform
solve	Löst Gleichungen
gausselim	Gauss-Verfahren
gaussjord	Gauss-Jordan-Verf.

Tabelle 27: Lineare Gleichungssysteme symbolisch lösen

Funktionen, die man einsetzen kann, um lineare Gleichungen symbolisch zu lösen. Beachten Sie, dass es das Gleiche bedeutet, eine Matrix in reduzierte Zeilenstufenform zu bringen wie das Gauss-Jordan-Verfahren durchzuführen. Die Funktion rref bzw. das Kommando \ stehen auch für numerische Berechnungen zur Verfügung.

49.3. Lineare Gleichungssysteme (4)

Über lineare Gleichungssysteme haben wir in den vorhergehenden Abschnitten schon viel gesagt. Hier soll noch ergänzt werden, dass der \-Operator (Backslash-Operator) auch verwendet werden kann, um Matrizengleichungen AX = B zu lösen. Hier ist die Rechte Seite B eine Matrix mit p Spalten. In diesem Fall löst MATLAB AX(:, j) = B(:, j) für j = 1: p.

49.4. Inverse

Die Inverse einer (n,n)-Matrix ist die Matrix X, die den Gleichungen $AX = XA = E_n$ genügt. Hierbei ist E_n die (n,n)-Einheitsmatrix, die in Matlab mit eye(n) erzeugt werden kann. Eine Matrix, die keine Inverse hat, heißt singulär. Eine singuläre Matrix kann auf mehrere Arten charakterisiert werden: Die Determinante ist Null oder es gibt einen Vektor $v \neq o$ mit Av = o.

Die Inverse einer quadratischen Matrix kann mit der Funktion inv (auch symbolisch) berechnet werden. Zum Beispiel

>> A = pascal(3), X = inv(A)				
A =				
	1	1	1	
	1	2	3	
	1	3	6	
X =				
	3	-3	1	
	-3	5	-2	
	1	-2	1	
>> norm(A*X-eye(3))				
ans	=			
	0			

Die Inverse wird mit Hilfe der LU-Faktorisierung mit partieller Pivotisierung berechnet. Gleichzeit wird mit rond der Kehrwert der Konditionszahl berechnet. Wird festgestellt, dass A (exakt) singulär oder die rond kleiner als eps ist, so wird eine Warnung ausgegeben.

Beachten Sie: Die Inverse einer Matrix muss explizit in der Praxis nur sehr selten berechnet werden. Zum Beispiel ist das Berechnen eines quadratischen lineare Gleichungssystems Ax = b mit A\b zwei bis dreimal schneller als durch inv(A)*b. Gewöhnlich ist es möglich, das Berechnen der Inversen einer Matrix in Termen eines linearen Gleichungssystems auszudrücken, so dass die explizite Inversion vermieden werden kann.

50. Lineare Algebra (Teil 3)

Matlab wurde ursprünglich entwickelt, um numerische Berechnungen in der Linearen Algebra durchzuführen. Daher ist es nicht überraschend, dass viele Funktionen zum Lösen linearer Gleichungssysteme und Eigenwertaufgaben zur Verfügung stehen. Sehr viele dieser Funktionen basieren auf der Fortran-Bibliothek Lapack [1].

Lange Zeit waren reelle und komplexe Matrizen der einzige Datentyp in Matlab und die meisten Funktionen können auch mit reellen und komplexen Matrizen aufgerufen und bearbeitet werden. Das Teilgebiet, das sich mit numerischen Methoden in der Linearen Algebra befasst, heißt *Numerische Lineare Algebra*. Jedes Buch über *Numerische Mathematik* behandelt auch dieses Thema, siehe zum Beispiel [7], [12] und der darin zitierten Literatur.

50.1. Normen

Eine Norm ist ein skalares Maß für die Größe eines Vektors oder einer Matrix. Die p-Norm eines n-Vektors x ist definiert durch

$$||x||_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}, \qquad 1 \le p < \infty.$$

Für $p = \infty$ ist

$$||x||_{\infty} = \max_{1 \le i \le n} |x_i|.$$

Die Funktion norm kann verwendet werden, um jede p-Norm auszurechnen. Der Aufruf ist norm(x,p), wobei p=2 standardmäßig eingestellt ist. Im Fall $p=-\inf$ wird $min_i|x_i|$ berechnet.

```
>> x = 1:4;
>> [norm(x,1) norm(x,2) ...
norm(x,inf) norm(x,-inf)]
ans =
    10.0000    5.4772    4.0000
    1.0000
```

Die Tabelle 28 zeigt die Vektornormen über-

MATLAB	Bedeutung
norm(x) = norm(x, 2)	$ x _2$
norm(x,1)	$ x _{1}$
<pre>norm(x,inf)</pre>	$ x _{\infty}$
<pre>norm(x,-inf)</pre>	$ x _{-\infty}$
norm(x,p)	$ x _p$ für $p \ge 1$

Tabelle 28: Vektornormen

sichtlich.

Aufgabe 97 (Vektornormen) Berechnen Sie für den Vektor $\mathbf{x} = (1, -2, 3)$ die Normen $\|\mathbf{x}\|_p$ für $p = 1, 1.5, 2, \dots 9.5$ und für $p = \infty$.

Lösung: Der Aufruf Vektornormen

р	norm(x,p)	
1.0	6.0000	
1.5	4.3346	
2.0	3.7417	
2.5	3.4586	
'		

3.0	3.3019
3.5	3.2072
4.0	3.1463
4.5	3.1056
5.0	3.0774
5.5	3.0574
6.0	3.0430
6.5	3.0325
7.0	3.0247
7.5	3.0188
8.0	3.0144
8.5	3.0111
9.0	3.0086
9.5	3.0067
Inf	3.0000

berechnet die Normen. Die Funktion Vektornormen ist wie folgt realisiert.

```
function Vektornormen
%-Vektor x
x = [1 -2 3];
%
disp(' p norm(x,p)');
disp('----');
for p = [1:0.5:9.5 inf]
disp(sprintf('%2.1f %15.4f ',...
p,norm(x,p)))
end
```

Aufgabe 98 (Vektornormen) Berechnen Sie die 1, 2 und ∞ -Norm der Vektoren $v_1 = (2, 1)$, $v_2 = (1, 0)$ und $v_3 = (1, -1)$.

Lösung: Wir erhalten:

```
>> v1= [2,1];

[norm(v1,1) norm(v1,2) norm(v1,inf)]

ans =

3.0000 2.2361 2.0000
```

Die *p*-Norm einer Matrix ist definiert durch

$$||A||_p = \max_{x \neq o} \frac{||Ax||_p}{||x||_p}.$$

Die 1- und ∞ -Norm einer (m, n)-Matrix A sind (Maximale Spaltensumme)

$$||A||_1 = \max_{1 \le j \le n} \sum_{i=1}^m |a_{ij}|$$

und (Maximale Zeilensumme)

$$||A||_{\infty} = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|.$$

Die 2-Norm von A kann berechnet werden als der größte singuläre Wert von A, also $\max(\text{svd}(A))$. Matrizennormen werden ebenfalls mit der Funktion norm berechnet. Für Matrizen funktioniert der Aufruf norm(A,p) mit p=1,2, inf und p='fro', wobei

$$||A||_F = \left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2\right)^{1/2}$$

die Frobenius-Norm ist. Beachten Sie: Die Funktion norm ist ein Beispiel einer Funktion mit einem Argument, das einen unterschiedlichen Datentyp haben kann; hier double und char (String). Hier noch ein Beispiel:

Ist das Berechnen der Matrixnorm für p=2 zu aufwendig, so kann mit der Funktion normest ein Näherungswert berechnet werden. Die Tabelle 29 zeigt die Matrizennormen übersicht-

MATLAB	Bedeutung
norm(A)=norm(A,2)	A_2
norm(A,1)	\boldsymbol{A}_1
<pre>norm(A,inf)</pre>	$oldsymbol{A}_{\infty}$
<pre>norm(A,'fro')</pre>	$oldsymbol{A}_F$

Tabelle 29: Matrizennormen

lich.

Aufgabe 99 (Matrizennormen) Berechnen Sie die 1-, 2-, ∞- und Frobenius-Norm für die Matrizen

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}$$

und

$$C = \left[\begin{array}{cc} 1 & 4 \\ 1 & 2 \end{array} \right].$$

Lösung: Es ist

```
>> A = [0 1; -1 0];

>> [norm(A,1) norm(A,2) ...

norm(A,inf) norm(A,'fro')]

ans =

1.0000 1.0000 1.0000
```

```
1.4142
>> B = [1 2; 2 2];
>> [norm(B,1) norm(B,2) ...
norm(B,inf) norm(B,'fro')]
ans =
    4.0000
               3.5616
                         4.0000
    3.6056
>> C = [1 4; 1 2];
>> [norm(C,1) norm(C,2) ...
norm(C,inf) norm(C,'fro')]
ans =
    6.0000
               4.6708
                         5.0000
    4.6904
```

50.2. Konditionszahlen

Für eine reguläre Matrix A ist

$$\kappa(A) = \frac{\|A\|}{\|A^{-1}\|} \ge 1$$

die Konditionszahl bezüglich der Inversion. Die Zahl misst die Sensitivität der Lösung eines linearen Gleichungssystems Ax = b bezüglich Störungen in A und b. Man sagt, dass A gut oder schlecht konditioniert ist, wenn $\kappa(A)$ klein bzw. groß ist, wobei "klein" und "groß" vom Kontext abhängig sind. Die Kondition kann mit der Funktion cond berechnet werden, wobei die Normen p=1,2,inf,'fro' unterstützt werden. Für p=2 sind auch rechteckige Matrizen erlaubt, wobei dann die Konditionszahl durch

$$\kappa_2(\mathbf{A}) = \frac{||\mathbf{A}||_2}{||\mathbf{A}^+||_2}$$

definiert ist, wobei A^+ die Pseudoinverse von A ist.

Die Konditionszahl auszurechnen ist aufwendig. Daher gibt es in MATLAB zwei Funktionen, die die Konditionszahl näherungsweise berechnen: rcond und condest, siehe doc rcond und doc condest.

50.3. LU-Faktorisierung

Eine LU-Faktorisierung einer quadratischen Matrix A ist eine Faktorisierung der Form A =LU, wobei L eine untere Dreicksmatrix mit Einsdiagonale und *U* eine obere Dreicksmatrix sind. Nicht jede Matrix kann in dieser Form faktorisiert werden, lässt man aber Zeilenvertauschungen zu, so ist dies immer möglich und solch eine Faktorisierung existiert immer. Die Funktion 1u berechnet eine LU-Faktorisierung mit teilweiser Pivotisierung PA = LU, wobei P eine Permutationsmatrix ist. Der Aufruf [L,U,P] = lu(A) gibt die Dreiecksmatrizen und die Permutationsmatrix zurück. Mit zwei Ausgabeargumenten, [L,U] = lu(A) wird Uund $L = P^{T}L$ zurückgegeben, das heißt List eine Dreiecksmatrix bis auf Zeilenvertauschungen. Wir geben ein Beispiel.

```
>> A = [1 \ 2 \ -1; \ -2 \ -5 \ 3; \ -1 \ -3 \ 0];
[L,U,P] = lu(A)
L =
     1.0000
                                   0
   -0.5000
                 1.0000
                 1.0000
    0.5000
                             1.0000
U =
   -2.0000
               -5.0000
                             3.0000
                -0.5000
                             0.5000
           0
                            -2.0000
P =
             1
                    0
```

```
1
                   0
      0
             0
                    1
\gg [L,U] = lu(A)
   -0.5000
                1.0000
                                 0
     1.0000
     0.5000
                1.0000
                            1.0000
U =
   -2.0000
               -5.0000
                            3.0000
               -0.5000
                            0.5000
                          -2.0000
```

Die 1u Funktion arbeitet auch für rechteckige Matrizen. Ist A eine (m, n)-Matrix, dann ist L eine (m, n)- und U eine (n, n)-Matrix, falls $m \ge n$ ist und L eine (n, n)- und U eine (m, n)-Matrix, wenn m < n gilt.

Das Lösen eines quadratischen linearen Gleichungssystems Ax = b durch x=A ist gleichbedeutend damit, von A eine LU-Faktorisierung zu bilden, und dann mit den Faktoren zu lösen:

```
[L,U] = lu(A);
x = U\(L\b);
```

Hier ist ein Beispiel.

```
>> A = [1 1 2; 2 4 -3; 3 6 -5];
>> b = [9;1;0];
>> x = A\b
x =
    1.0000
    2.0000
    3.0000
>> [L,U] = lu(A);
>> x = U\(L\b)
x =
    1.0000
    2.0000
```

3.0000

Der Vorteil dieser Vorgehensweise ist, dass man sich Rechnungen einspart, wenn ein Gleichungssystem mit mehreren rechten Seiten oder eine Matrix gleichung vorliegt, weil dann die Faktorisierung nur einmal gemacht werden muss.

50.4. Cholesky-Faktorisierung

Jede positiv definite Hermitesche Matrix hat eine Cholesky-Zerlegung $A = R^*R$, wobei R eine obere Dreiecksmatrix mit reeller positiver Diagonale ist. Die Cholseky-Faktorisierung wird mit chol berechnet: R=chol(A). Hier ein Beispiel.

```
\gg A = pascal(3)
A =
      1
             1
                     1
      1
             2
                     3
             3
      1
                     6
>> R = chol(A)
R =
      1
             1
                     1
      0
             1
                     2
      0
                     1
```

Und die Probe:

>> R'*R			
ans =			
1	1	1	
1	2	3	
1	3	6	

Beachten Sie, dass chol nur die obere Dreiecksmatrix einschließlich der Diagonalen von

A zur Faktorisierung verwendet. Ist die Matrix A nicht positiv definit, so erhält man eine Fehlermeldung. Tatsächlich ist die Funktion chol gegeignet um zu überprüfen, ob eine Matrix positiv definit ist. Ist nach dem Aufruf [R,p]=chol(A) p gleich Null, so ist A positiv definit. Siehe doc chol für weitere Detail zum Ausgabeargument p.

50.5. QR-Faktorisierung

Eine QR-Faktorisierung einer (m,n)-Matrix A ist eine Faktorisierung der Form A = QR, wobei Q eine (m,m)-unitäre und R eine (m,n)- obere Dreicksmatrix ist. Diese Faktorisierung wird zum Lösen linearer Ausgleichsaufgaben und zur Konstruktion einer orthonormalen Basisfür den Spaltenraum von A eingesetzt. Die Anweisung [Q,R] = qr(A) berechnet die Faktorisierung. Ist m > n, so gibt die Anweisung [Q,R] = qr(A,0) "wirtschaftlichere" Matrizen zurück. Man spricht auch von reduzierter QR-Zerlegung. Die Matrix Q hat nur n Spalten und die Matrix R hat n Zeilen und n Spalten. Wir geben ein Beispiel.

```
>> A = [1 0 0; 1 1 0; 1 1 1];
\gg [Q,R] = qr(A)
Q =
   -0.5774
               0.8165
                         -0.0000
   -0.5774
              -0.4082
                         -0.7071
   -0.5774
              -0.4082
                          0.7071
R =
   -1.7321
                         -0.5774
              -1.1547
              -0.8165
         0
                         -0.4082
                          0.7071
```

Hier noch die Probe:

Aufgabe 100 (QR-Faktorisierung) Berechnen Sie von der Matrix

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}$$

die QR- und die reduzierte QR-Faktorisierung. Lösung: Es ist

```
\gg [Q,R] = qr(A)
Q =
               0.6708
   -0.5000
                          0.5000
   -0.5000
               0.2236
                          -0.5000
   -0.5000
              -0.2236
                         -0.5000
   -0.5000
              -0.6708
                          0.5000
    0.2236
   -0.6708
    0.6708
   -0.2236
R =
   -2.0000
              -1.0000
                         -3.0000
              -2.2361
                         -2.2361
          0
          0
                     0
                          2.0000
          0
                     0
                                0
```

und

In der orthogonalen Matrix Q fehlt nun die letzte Spalte und in der Dreiecksmatrix R die letzte Nullzeile. \odot \odot

50.6. Singulärwertzerlegung

Die Singulärwertzerlegung (Singular Value Decomposition, kurz: SVD) einer (m, n)-Matrix A hat die Form $A = USV^*$, wobei U eine unitäre (m, m)-Matrix, S eine reelle (m, n)-Diagonalmatrix und V eine unitäre (n, n)-Matrix ist. Die Diagonalelemente von Ssind die singulären Werte s_i mit $s_1 \ge s_2 \ge$ $\cdots \geq s_{\min\{m,n\}} \geq 0$. Die Anweisung [U,S,V] = svd(A) berechnet eine Singulärwertzerlegung. Gibt man nur eine Ausgabevariable an, so ist dies der Vektor, der aus den singulären Wert besteht. Die Anweisung [U,S,V] = svd(A,0) liefert eine reduzierte Singulärwertzerlegung (economy size), wenn m > nist. Dann ist U eine (m, n)-Matrix mit orthonormalen Spalten und S hat die Größe (n, n). Die Anweisung [U,S,V] = svd(A, 'econ') liefert das gleiche Ergebnis wie [U,S,V] = svd(A,0), wenn $m \ge n$ ist, ist aber m < n, so hat V die Größe (n, m) mit orthonormalen Spalten, und S ist quadratisch mit Ordnung m. Hier ist ein Beispiel [7]:

Die Funktionen rank, null und orth berechnen den Rang, eine orthonormale Basis für den Nullraum und eine orthonormale Basis für den Spaltenraum der Argumentmatrix. Alle drei Funktionen basieren auf der Singulärwertzerlegung, wobei Ein Toleranzwert proportional zu eps verwendet wird, um zu entscheiden, wann ein singulärer Wert als Null betrachtet werden kann. Wir geben ein Beispiel.

```
>> A = [1 -1 1; 1 0 0; 1 1 1; ...
        1 2 4];
\gg [U,S,V] = svd(A)
U =
   -0.1517
               0.8959
                         -0.3525
   -0.0612
               0.4013
                          0.6207
   -0.3215
               0.0406
                          0.6671
   -0.9327
              -0.1861
                         -0.2134
    0.2236
   -0.6708
    0.6708
   -0.2236
S =
    4.8956
                    0
                               0
         0
               1.6942
                               0
```

```
1.0784
         0
                    0
         0
                    0
V =
               0.6798
                          0.6693
   -0.2997
   -0.4157
              -0.7245
                          0.5498
   -0.8587
               0.1135
                         -0.4997
>> rank(A)
ans =
     3
>> null(A)
ans =
   Empty matrix: 3-by-0
>> orth(A)
ans =
   -0.1517
               0.8959
                         -0.3525
   -0.0612
               0.4013
                          0.6207
   -0.3215
               0.0406
                          0.6671
   -0.9327
              -0.1861
                         -0.2134
```

Aufgabe 101 (SVD [5]) Berechnen Sie eine Singulärwertzerlegung der Matrix

$$\boldsymbol{A} = \left[\begin{array}{cc} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{array} \right].$$

Lösung: Es ist

```
-0.7071 -0.7071
```

Man erkennt, dass die Matrix A den Rang 1 hat; die singulären Werte sind $s_{11} = 2$ und $s_{22} = 0. \odot \dots \odot$

Die verallgemeinerte Singulärwertzerlegung (Generalized Singular Value Decomposition, kurz: GSVD) einer (m, p)-Matrix A und einer (n, p)-Matrix B hat die Form $A = UCX^*$, $B = VSX^*$, $C^*C + S^*S = E$, wobei U und V unitär sind, X regulär, und C und D sind reelle Diagonalmatrizen mit nichtnegativen Diagonalelementen. Die Zahlen c_{ii}/s_{ii} sind die verallgemeinerten singulären Werte. Diese Zerlegung wird in Matlab wie folgt erreicht: [U,V,X,C.S] = gsvd(A,B). Siehe doc gsvd (help gsvd) für weitere Details.

50.7. Pseudoinverse

Die Pseudoinverse ist die Verallgemeinerung der Inversen für Matrizen A, die singulär oder nicht quadratisch sind ([5]). Die mathematische Notation ist A^+ . Sie wird in Matlab mit pinv berechnet. Die Pseudoinverse wird mit Hilfe der Singulärwertzerlegung berechnet. Beispiele ([5]):

```
>> A = [2 0; 0 0];

>> pinv(A)

ans =

    0.5000     0

    0     0

>> B = [2; 4];

>> pinv(B)

ans =

    0.1000     0.2000

>> C = [2 0; 0 3; 0 0];
```

```
>> pinv(C)
ans =
0.5000 0 0
0 0.3333 0
```

50.8. Eigensysteme

Die effiziente numerische Berechnung von Eigensystemen (Eigenwerte und Eigenvektoren) ist ein komplexer Akt. Die Matlab-Funktion eig vereinfacht diesen Lösungsprozess, indem sie die Anzahl der Eingabematrizen, sowie deren Struktur und die Ausgabe berücksichtigt. Sie unterscheidet intern zwischen 16 verschiedenen Algorithmen:

- Standardproblem (eig(A)) oder verallgemeinertes Problem (eig(A,B)),
- reelle oder komplexe Matrizen A und B,
- symmetrische/Hermitesche Matrizen A und B mit B positiv definit oder nicht,
- sind Eigenvektoren gewünscht oder nicht. Beispiele findet man in Abschnitt 48.6.

50.9. Iterative Methoden

In diesem Abschnitt besprechen wir iterative Methoden. Solche Verfahren werden hauptsächlich bei großen und möglicherweise dünn besetzten (sparse) Problem eingesetzt; also immer dann, wenn direkte Verfahren zu ineffizient oder gar unmöglich anwendbar sind. Dünn besetzte Matrizen behandeln wir in Abschnitt 72. Es gibt in MATLAB iterative Methoden für lineare Gleichungssysteme und iterative Verfahren für Eigensysteme.

50.9.1. Iterative Methoden für lineare Gleichungssysteme

Es stehen mehrere Funktionen zur Verfügung, um quadratische lineare Gleichungssysteme Ax = b iterativ zu lösen, siehe Tabelle 30.

Funktion	Methode
bicg	BiCG
bicgstab	BiCG stabilized
cgs	CG squared
gmres	GMRES
lsqr	CG für Normalleichungen
minres	Residuienm.
pcg	CG mit Vorkonditionierung
qmr	quasi-minimale Residuienm.
symmlq	Symmetrische LQ

Tabelle 30: Iterative Methoden

Bis auf die Funktion minres, symmlq und pcg sind alle Funktionen auf beliebige Matrizen A anwendbar. Für minres und symmlq muss A Hermitesch und für pcg muss A Herмітеsch und positiv definit sein. Alle Methoden verwenden Matrix-Vektor Produkte Ax bzw. A^*x und benötigen die Matrixelemente von A nicht explizit. Bis auf gemres (siehe unten) haben alle Funktionen den gleichen Funktionsaufruf. Die einfachste Form ist x = solver(B,b), wobei solver eine Funktion aus Tabelle 30 ist. Alternativ kann man auf x = solver(B,b,tol) mit einem Konvergenztoleranzwert tol, der standardmä-Big auf 10^{-6} gesetzt ist. Konvergenz tritt ein, wenn norm(b-A*x) <= tol*norm(b) erfüllt ist. Das Argument A kann eine voll oder dünn besetzt (sparse) Matrix sein, oder eine Funktion, die x als Eingabe und A*x als Ausgabe hat. Iterative Methoden benötigen gewöhnlich eine Vorkonditionierung, wenn sie effizient sein sollen. Einen guten Vorkonditionierer auszuwählen, ist im Allgemeinen nicht leicht und erfordert meist Kenntnisse von der Anwendung, die dahinter steckt. Die Funktionen luinc und cholinc berechnen unvollständige Faktorisierungen, die eine Möglichkeit darstellen, Vorkonditionierer zu konstruieren; siehe doc luinc, doc cholinc und doc bicg. Für Hintergrundinformationen zu iterativen Methoden der Linearen Algebra siehe [8], [10], [11], [18] und [22].

Um den Umgang mit iterativen Problemlösern zu zeigen, betrachten wir ein Beispiel, wo die Funktion pcg Verwendung findet. Die Funktion pcg realisiert eine konjugierte Gradientenmethode mit Vorkonditionierung. Für A verwenden wir eine positiv definite symmetrische Matrix, die Wathen-Matrix aus der Highamschen Matrizensammlung. Es handelt sich um eine zufällige dünn besetzte finite Elementematrix, siehe doc gallery.

```
>> A = gallery('wathen',12,12);
>> n = length(A)
n =
     481
>> b = ones(n,1);
>> x = pcg(A,b);
pcg stopped at iteration 20 without converging to the desired tolerance 1e-006 because the maximum number of iterations was reached.
The iterate returned (number 19) has relative residual 0.12
>> x = pcg(A,b,1e-6,100);
```

pcg converged at iteration 92 to a solution with relative residual 9.8e-007

Im ersten Aufruf haben wir nur die Systemmatrix A und die rechte Seite b eingegeben und gesehen, dass die konjugierte Gradientenmethode nicht konvergiert mit standardmäßig eingestellten 20 Iterationen und dem standard Abbruchkriterium von 10^{-6} . Ein erneuerte Versuch führt nach 92 Iterationen zum Erfolg, wobei wir die Iterationsgrenze auf 100 gesetzt haben. Der Toleanzwert 10^{-6} wurde erfüllt. Für diese Matrix kann man zeigen, dass M = diag(diag(A)) ein guter Vorkonditionierer ist. Übergeben wir der Funktion pcg diese Matrix als fünftes Argument, so erreichen wir eine effiziente Reduktion in der Anzahl der Iterationen:

```
>> [x,flag,relres,iter] = pcg(A,...
    b,1e-6,100,diag(diag(A)));
>> flag, relres, iter
flag =
    0
relres =
    8.2661e-007
iter =
    28
```

Beachten Sie, dass keine Bildschirmausgabe ausgegeben wird, wenn wenn man mehr als ein Ausgabeargument im Funktionsaufruf angibt. Der Wert 0 der Variablen flag gibt an, dass das Verfahren konvergiert ist mit einem relativen Residuum relres = norm(b-A*x)/nomrm(b) nach iter Iterationen.

Die anderen Funktionen der Tabelle 30 werden (bis auf gmres) genauso aufgerufen und gehandelt.

Mit doc sparfun (help sparfun) bekommen Sie eine komplette Liste über alle Funktionen, die bezüglich Sparserechnungen interessant sind, so auch die Funktionen für iterative Methoden von linearen Gleichungssystemen.

50.9.2. Iterative Methoden für Eigensysteme

Die Funktion eigs berechnet auswählbare Eigenwerte und Eigenvektoren des Standardeigenwertproblems $Ax = \lambda x$ oder der verallgemeinerten Eigenwertaufgabe $Ax = \lambda Bx$, wobei B eine reelle symmetrische positiv definite Matrix ist. Im Gegensatz dazu berechnet die Funktion eig immer das gesamte Eigensystem, also alle Eigenwerte und Eigenvektoren. Wie die iterativen Löser von linearen Gleichungssystemen, so benötigt eigs nur Matrix-Vektor Produkte, das heißt A kann explizit oder als eine Funktion, die Matrix-Vektor ausführt, angegeben werden. In der einfachsten Form kann eigs wie eig aufgerufen werden: [V,D] = eigs(A). Dann werden die sechs größten Eigenwerte und dazugehörige Eigenvektoren berechnet. Siehe doc eigs für weitere Details. Diese Funktion ist eine Schnittstelle zum ARPACK Paket, siehe [13]. Als Beispiel berechnen wir nun die fünf größten Eigenwerte einer dünn besetzten symmetrischen Matrix. Zum Vergleich verwenden wir die Funktion eig, welche erwartet, dass die Matrix im Speichermodus full vorliegt.

```
>> A = delsq(numgrid('N',40));
>> n = length(A)
n =
        1444
>> nnz(A)/n^2 %Anteil nicht Null.
ans =
    0.0034
>> tic, e_alle = eig(full(A)); toc
Elapsed time is 10.600102 seconds.
>> e_alle(n:-1:n-4)
ans =
    7.9870
    7.9676
    7.9676
    7.9482
    7.9354
>> options.disp = 0; %Keine Ausgabe.
>> tic, e = eigs(A,5,'LA',options);
>> toc %LA = Largest Algebraic
Elapsed time is 1.676173 seconds.
>> e
e =
    7.9870
    7.9676
    7.9676
    7.9482
    7.9354
```

Mit den Funktion tic und toc kann man Zeitmessungen durchführen. Es ist klar, dass eigs viel schneller ist als eig und auch weniger Speicherplatz benötigt.

50.9.3. Iterative Methoden für Singulärwertsysteme

Die Funktion, um ein paar singuläre Werte und singuläre Vektoren einer Matrix $A \in \mathbb{C}^{m \times n}$ iterativ zu berechnen, heißt: sdvs. Die Funktion svds verwendet dabei die Funktion eigs, indem eigs auf die Hermitesche Matrix

$$\left[\begin{array}{cc} \boldsymbol{O}_m & \boldsymbol{A} \\ \boldsymbol{A}^* & \boldsymbol{O}_n \end{array}\right]$$

angewendet wird.

50.10. Funktionen einer Matrix

Mit der Funktion expm kann man für eine quadratische Matrix A die Matrixexponentialfunktion e^A , die durch

$$e^{A} = E + A + \frac{1}{2!}A^{2} + \frac{1}{3!}A^{3} + \cdots$$

definiert ist, berechnen. Wir geben ein Beispiel, siehe Beispiel 3.5 in [5].

Weitere Funktionen für Matrixfunktionen finden Sie mit help matfun.

Aufgabe 102 (Matrixfunktionen) In dieser Aufgabe geht es um e^A , wobei A eine quadratische Matrix ist.

(a) Berechnen Sie e^A numerisch und symbolisch für

$$\boldsymbol{A} = \left[\begin{array}{cc} 0 & -1 \\ 1 & 0 \end{array} \right].$$

Vergleichen Sie die Resultate mit exp(A) bzw. exp(sym(A)). Erklären Sie den Unterschied!

- (b) Berechnen Sie nun symbolisch e^{At} , $t \in \mathbb{R}$, und deuten Sie das Ergebnis.
- (c) Ist X eine quadratische Matrix, dann gilt

$$\frac{de^{Xt}}{dt} = Xe^{Xt} = e^{Xt}X$$

und

$$\int e^{Xt} X dt = e^{Xt}$$

Bestätigen Sie diese Aussagen symbolisch.

Lösung:

(a) Es ist

Dagegen ist aber

und

In den letzten beiden Fällen werden von den Matrixelementen die Exponentialwerte ausgerechnet.

(b) Es ist mit A wie in (a) und

```
>> syms t
```

dann

(c) Nach der Deklaration

kann man die Aussagen durch

und

bestätigen.

Aufgabe 103 (Matrixfunktionen) Berechnen Sie die Matrix e^{At} , $t \in \mathbb{R}$ für folgenden Matrizen:

(a)
$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

(b)
$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

(c)
$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

(d)
$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

(e)
$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Lösung: Man erhält:

(a)
$$e^{\mathbf{A}t} = \begin{bmatrix} e^t & 0 \\ 0 & e^{2t} \end{bmatrix}$$

(b)
$$e^{At} = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}$$

(c)
$$e^{At} = \begin{bmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{bmatrix}$$

(d)
$$e^{At} = \begin{bmatrix} 1 & t & t^2/2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix}$$

(e)
$$e^{At} = \begin{bmatrix} \cosh t & \sinh t \\ \sinh t & \cosh t \end{bmatrix}$$

51. Mehr zu Funktionen

Funktionen spielen in Matlab eine sehr große Rolle. Seit Matlab 7 ergaben sich diesbezüglich einige Neuerungen. Wir wollen deshalb in diesem Abschnitt mehr zu Matlab-Funktionen sagen. Eine erste Einführung findet man in Abschnitt 44.

51.1. Function-Handles

In viele Anwendungen muss man einer Funktion als Argument eine andere Funktion übergeben. In Abschnitt 39 haben wir Beispiele betrachtet. Gewöhnlich macht man dies mit einem Function-Handle. Ein Function-Handle ist

ein Matlab-Datentyp, der alle notwendigen Informationen enthält, um eine Funktion auszuwerten. Ein Function-Handle wird erzeugt, indem man das @-Zeichen vor einen Funktionsnamen stellt.

Wir zeigen die Funktionsweise mit der Funktion ezplot, die den Graph einer Mathematischen Funktion f standardmäßig über dem Intervall $[-2\pi, 2\pi]$ zeichnet. Ist nun zum Beispiel fun eine m-Function, so geschrieben wie es die Funktion ezplot verlangt, dann funktioniert

fun kann aber auch eine Matlab-Funktion sein, wie zum Beispiel sin, cos oder exp. Der Aufruf

zeichnet die Kosinusfunktion im Intervall $[-2\pi, 2\pi]$.

Function-Handles wurden in Matlaß 6 eingeführt und ersetzen die frühere Syntax, nach der ein Funktionsname als eine Zeichenkette (String) übergeben werden sollte, also zum Beispiel ezplot ('cos').

Die Idee des nächsten Beispiels ist es, den Differenzenquotient

$$\frac{f(x+h)-f(x)}{h}$$

in einer Matlab-Funktion zu realisieren, um diesen dann für "kleines" h als Näherung für den Differenzialquotient f'(x) zu berechnen. Die m-Function

```
function y = Differenzenq(f,x,h)
if nargin < 3</pre>
```

```
h = sqrt(eps);
end
y = (f(x+h) - f(x))/h;
```

ist eine Implementierung des Differenzenquotienten von f an der Stelle x. Geben wir nun

```
>> Differenzenq(@sqrt,0.3)
ans =
    0.9129
```

ein, so erhalten wir 0.9129 als Näherung für f'(0.3), wobei hier $f(x) = \sqrt{x}$ ist. Standardmäßig haben wir $h \approx 1.5 \cdot 10^{-8}$ gewählt. Soll h größer oder kleiner sein, so kann dies im dritten Argument eingestellt werden. Dieses Beispiel zeigt, die Funktionsweise eines Function-Handles. Wir übergeben den Function-Handle als Argument und diese wird an den entsprechenden Stellen ausgewertet. Diese direkte Auswertung ist neu in Matlab 7. In früheren Versionen musste hier mit der Funktion feval gearbeitet werden.

Weitere Informationen finden Sie mit doc function handle.

51.2. Anonymous Functions

Anonymous Functions wurde in MATLAB 7 eingeführt. Anonymous Functions stellen einen Weg dar, eine "one line"-Funktion zu erzeugen, ohne einen *m*-File schreiben zu müssen. In Abschnitt 39 haben wir dies bereits zum ersten Mal gezeigt.

```
>> f = @(x) exp(x)-2
>> f(1)
ans =
```

```
0.7183
```

Hier ist f ein Function-Handle zu der Anonymous Function. Dem @-Zeichen, welches ein Function-Handle konstruiert, folgt in runden Klammern eine Liste von Eingabeargumenten und dannach folgt eine einfache Matlab-Anweisung. Als Function-Handle kann die Anonymous Function dann einer anderen Funktion übergeben werden.

Der Ausdruck einer Anonymous Function kann auch Variable haben, die nicht in der Argumentenliste vorkommen. So definiert die Anweisung

```
>> a = 2; b = 3;
>> g = @(x,y) a*x^2+b*exp(x);
```

den Mathematischen Funktionsterm $g(x, y) = ax^2 + be^x$ mit a = 2 und b = 3.

Die Anweisungen und Ausgaben

```
>> g(3,4)

ans =

78.2566

>> a = 10;

>> g(3,4)

ans =

78.2566
```

zeigen, dass die Variablenwerte so lange gültig bleiben, bis man die Anonymous Function neu konstruiert.

Aufgabe104(Anonymous Functions) DieDichtefunktionf einer normalverteiltenZufallsvariablenist die parameterabhängige

Funktion

$$f(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R}$$

wobei μ und $\sigma > 0$ reelle Parameter (Erwartungwert und Standardabweichung) sind. Diese Funktion ist in der *Statistics-Toolbox* unter dem Namen normpdf implementiert. Zeichnen Sie die Graphen der Funktionen $f(\cdot, 0, 1)$ und $f(\cdot, 0, 2)$ im Intervall [-4, 4].

Lösung: Die Anweisungen

```
mu = 0; sigma = 1;
f = @(x) normpdf(x,mu,sigma);
fplot(f,[-4,4]), hold on,
sigma = 2;
f = @(x) normpdf(x,mu,sigma);
fplot(f,[-4,4]),
```

erzeugen die Abbildung 46. ©

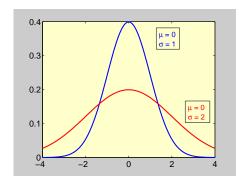


Abbildung 46: Normalverteilungen

51.3. Subfunctions

Eine Function m-File kann weitere Funktionen beinhalten, die man subfunctions (Unterfunktionen) nennt. Diese können in einem m-File in irgendeiner Reihenfolge nach der Hauptfunktion (main function, primary function) stehen. Subfunctions sind nur der Hauptfunktion und den anderen im m-File befindlichen Subfunctions sichtbar.

Wie Subfunctions verwendet werden können, zeigen wir in den folgenden Abschnitten, siehe zum Beispiel 62.2.

51.4. Nested Functions

Man kann eine und mehr Funktionen in einer Funktion schachteln. Es muss darauf geachtet werden, dass die end-Anweisung am Ende der eingebauten und der Hauptfunktion steht. Es gehört zum guten Stil, die eingebaute Funktion einzurücken.

Nested Functions haben zwei Haupteigenschaften:

- Eine Nested Function hat Zugriff zum Workspace aller Funktionen in welche sie eingebaut ist.
- Ein Function-Handle einer Nested Function speichert alle Informationen, um zu dieser Zugang zu haben, und darüberhinaus alle Variablen der Funktionen, die die Nested Funktion beinhalten.

Ein Beispiel einer Nested Function zeigt das folgende Listing:

```
function RationalBsp(x)
a = 1; b = 2; c = 1; d = -1;
Differenzenq(@Rational,x)
function r = Rational(x)
r = (a+b*x)/(c+d*x);
end
end
end
```

Das Beispiel zeigt, wie eine parameterabhängige Funktion einer anderen Funktion übergeben werden kann. Die Funktion Rational ist parameterabhängig und wird der Funktion Differenzenq übergeben. Die Variablen a, b, c und d des Workspace der Hauptfunktion RationalBsp sind der Funktion Rational bekannt und die Werte werden der Funktion Differenzenq ordungsgemäß weitergereicht:

```
>> RationalBsp(2)
ans =
    3.0000
```

Das Ergebnis ist korrekt, denn für den gebrochen rationalen Funktionsterm f(x) = (1 + 2x)/(1-x) gilt $f'(x) = 3/(1-x)^2$, also f'(2) = 3

Nested Functions sind neu seit MATLAB 7.

51.5. Beispielhafte Funktionen

Der wahrscheinlich beste Weg gute Matlab-Funktionen zu schreiben, besteht darin, sich Beispiele von vorbildlich geschriebenen Funktionen anzusehen und sich daran zu orientieren. Solche Funktionen werden natürlich mit Matlab mitgeliefert. Schauen Sie sich diese an! Mit type <Funktionsname> bekommen Sie ein Listing und mit edit <Funktionsname> können Sie die Funktion in den Matlab-Editor laden. Hier sind ein paar Beispiele:

• cov: Verwendung von varargin

• var: Überprüfung der Argumente

• hadamard: Aufbau einer Matrix

• why: Subfunctions

• fminbnd: Schleifenkonstruktionen

• quad, quad1: Rekursive Funktionen

• gsvd: Subfunctions

• ode45: Verwendung von varargin und varargout

52. Lineare Ausgleichsaufgaben

Lineare Ausgleichsaufgaben können mit Mat-LAB auf mehrere Arten gelöst werden. Zum Einem ist da der \-Operator (Backslash-Operator), siehe Abschnitt 48.4 und Abschnitt 49.1.2.

Lineare Ausgleichsprobleme mit polynomialer Modellfunktion können alternativ mit der Funktion polyfit und interaktiv aus jeder *Figure* unter *Tools Basic Fitting* gelöst werden. Siehe auch Abschnitt 55.

Nichtlineare Ausgleichsaufgaben werden in Abschnitt 70 besprochen.

53. Polynome

Polynome haben in der Mathematik eine große Bedeutung. Im wissenschaftlichen Rechnen benutzt man sie, um kompliziertere Funktionen durch einfachere anzunähern. Polynome lassen sich einfach differenzieren und integrieren, und man weiß, wie man numerische Näherungen für die Nullstellen findet. Numerische Schwierigkeiten können auftreten, wenn man mit Polynomen höherer Ordnung arbeitet.

In Matlab gibt es eine Reihe von Funktionen, die es ermöglichen, effizient mit Polynomen zu

arbeiten. Zum symbolischen Rechnen mit Polynomen siehe Abschnitt 67.13.

53.1. Darstellung von Polynomen

Polynome werden in Matlab durch einen Zeilenvektor repräsentiert, wobei die Koordinaten die Koeffizienten des Polynoms darstellen. Die Reihenfolge ist absteigend festgelegt, das heißt die erste Koordinate des Vektors ist der Koeffizient des Monoms höchster Ordnung und die letzte Kooordinate ist der Koeffizient des konstanten Terms des Polynoms. Zum Beispiel repräsentiert der Zeilenvektor [1 -8 2 1 12] das Polynom $p_1(x) = x^4 - 8x^3 + 2x^2 + x - 12$; [2 0 1] stellt das Polynom $p_2(x) = 2x^2 + 1$ dar. Beachten Sie, dass Nullkoeffizienten mitgeführt werden müssen.

53.2. Nullstellen von Polynomen

Ist ein Polynom als Zeilenvektor dargestellt, so erlaubt die Funktion roots, die Nullstellen des Polynoms zu berechnen, das heißt diejenigen Werte zu bestimmen, für die das Polynom den Wert Null annimmt. roots (p2) berechnet die Nullstellen des Polynoms $2x^2 + 1$, gibt sie als Spaltenvektor zurück und weist diese der Variablen r zu.

```
>> r = roots(p2)
r =
0 + 0.7071i
0 - 0.7071i
```

Kennt man die Nullstellen eines Polynoms, nicht aber das Polynom selbst, so kann man mit der Funktion poly das Polynom konstruieren. Da ein Polynom aber durch seine Nullstellen nur bis auf ein Vielfaches eindeutig bestimmt ist, wählt Matlab das Polynom mit Koeffizient 1 im höchsten Monom.

poly(r) berechnet aus den Nullstellen in r das Polynom $x^2 + 1/2$.

```
>> poly(r)
ans =
1.0000 0 0.5000
```

Aufgabe 105 (Polynomfunktionen) Bestimmen Sie die reellen Nullstellen folgender Polynome. Zeichnen Sie diese Polynome dann in einem geeigneten Intervall, um diese Nullstellen geometrisch überprüfen zu können ($x \in \mathbb{R}$).

(a)
$$g_1(x) = x^3 - 5x^2 + 2x + 8$$

(b)
$$g_2(x) = x^2 + 4x + 4$$

(c)
$$g_3(x) = x^5 - 3x^4 + 4x^3 - 4x + 4$$

Lösung:

```
>> roots([1 -5 2 8])
ans =
     4.0000
     2.0000
     -1.0000
>> roots([1 4 4])
ans =
     -2
```

```
-2
>> roots([1 -3 4 0 -4 4])
ans =
-1.0000
1.0000 + 1.0000i
1.0000 - 1.0000i
1.0000 + 1.0000i
1.0000 - 1.0000i
```

53.3. Multiplikation von Polynomen

Mit der Funktion conv kann man Polynome miteinander multiplizieren. Das Polynom $p_1(x) = x^2 + 2x - 3$ multipliziert mit dem Polynom $p_2(x) = 2x^3 - x^2 + 3x - 4$ ergibt das Polynom $p_3(x) = p_1(x) \cdot p_2(x) = 2x^5 + 3x^4 - 5x^3 + 5x^2 - 17x + 12$

```
>> p1 = [1 2 -3];

>> p2 = [2 -1 3 -4];

>> p3 = conv(p1,p2)

p3 =

2 3 -5 5 -17 12
```

53.4. Addition und Subtraktion von Polynomen

In Matlab gibt es keine Funktion zur Addition zweier Polynome. Haben die Polynome den gleichen Grad, werden die entsprechenden Zeilenvektoren addiert. Das Polynom $p_1(x) = x^2 + 2x - 3$ addiert mit dem Polynom $p_4(x) = -x^2 + 3x - 4$ ergibt das Polynom $p_5(x) = p_1(x) + p_4(x) = 5x - 7$

```
>> p1 = [1 2 -3];

>> p4 = [-1 3 -4];

>> p5 = p1+p4

p5 =

0 5 -7
```

Dies setzt jedoch Vektoren gleicher Länge, das heißt Polynome gleichen Grades, voraus. Will man Polynome unterschiedlichen Grades addieren, so muss man den Zeilenvektor zum Polynom kleineren Grades durch Nullen auffüllen. Das Polynom $p_1(x) = x^2 + 2x - 3$ addiert zu dem Polynom $p_2(x) = 2x^3 - x^2 + 3x - 4$ ergibt $p_6(x) = p(1) + p_2(x) = 2x^3 + 5x - 7$

```
>> p1 = [1 2 -3];

>> p2 = [2 -1 3 -4];

>> p6 = [0 p1]+p2

p6 =

2 0 5 -7
```

Die folgende Matlab-Funktion automatisiert diesen Prozess.

Um die Funktion addpoly zu verstehen, betrachte man die folgenden Zeilen:

Das Ergebnis ist das gleiche wie oben. Diese Matlab-Funktion addpoly kann auch dazu verwendet werden, um Polynome zu subtrahieren. Dies zeigen die folgenden Matlab-Zeilen:

 $p_1(x) = x^2 + 2x - 3$ minus $p_2(x) = 2x^3 - x^2 + 3x - 4$ ergibt das Polynom $p_7(x) = -2x^3 + 2x^2 - x + 1$.

53.5. Division von Polynomen

Mit der Funktion deconv kann man Polynome dividieren. Das Polynom $p_2(x) = 2x^3 - x^2 + 3x - 4$ dividiert durch das Polynom $p_1(x) = x^2 + 2x - 3$ ergibt das Polynom q(x) = 2x - 5 mit dem Restpolynom r(x) = 19x - 19.

53.6. Ableiten von Polynomen

Die Matlab-Funktion polyder differenziert ein Polynom. Die Ableitung des Polynoms $p_2(x) = 2x^3 - x^2 + 3x - 4$ ergibt das Polynom $p'_2(x) = 6x^2 - 2x + 3$.

```
>> p2 = [2 -1 3 -4];
>> dp2 = polyder(p2)
dp2 =
6 -2 3
```

53.7. Integrieren von Polynomen

Die Matlab-Funktion polyint integriert ein Polynom. Die Integration des Polynoms $p(x) = 6x^2 - 2x + 3$ ergibt das Polynom $q(x) = 2x^3 - x^2 + 3x + c$. Die Konstante c wird Null gesetzt.

53.8. Auswerten von Polynomen

Polynome können mit der Funktion polyval ausgewertet werden. Die folgenden Kommandos berechnen p(2.5) für das Polynom $p(x) = 4x^3 - 2x^2 + x - 7$.

Die folgenden Befehle berechnen 100 Polynomwerte für das Polynom $p(x) = 4x^3 - 2x^2 + x - 7$ zu 100 verschiedenen äquidistanten Werten im Intervall von -1 bis 4.

```
>> p = [4 -2 1 -7];
>> x = linspace(-1,4);
```

Aufgabe 106 (Polynomfunktionen) Gegeben seien die folgenden Polynomfunktionsterme

$$f_1(x) = x^3 - 3x^2 - x + 3$$

$$f_2(x) = x^3 - 6x^2 + 12x - 8$$

$$f_3(x) = x^3 - 8x^2 + 20x - 16$$

$$f_4(x) = x^3 - 5x^2 + 7x - 3$$

$$f_5(x) = x - 2$$

Zeichnen Sie die Polynome im Intervall [0, 4]. Benutzen Sie eingebaute Matlab-Funktionen, um folgende Polynome in den Punkten 0 und 1 auswerten zu können.

- (a) $f_2(x) 2f_4(x)$
- (b) $3f_5(x) + f_2(x) 2f_3(x)$
- (c) $f_1(x)f_3(x)$
- (d) $f_4(x)/(x-1)$

Lösung: Das folgende Script löst die Aufgabe.

```
p1 = [1 -3 -1 3];
p2 = [1 -6 12 -8];
p3 = [1 -8 20 -16];
p4 = [1 -5 7 -3];
p5 = [0 0 1 -2];
pa = p2-2*p4;
pb = 3*p5+p2-2*p3;
pc = conv(p1,p3);
pd = deconv(p4,[1 -1]);
x = linspace(0,4);
ypa = polyval(pa,x);
ypb = polyval(pb,x);
ypc = polyval(pc,x);
ypd = polyval(pd,x);
```

53.9. Zusammenfassung

In Matlab gibt es eine Reihe von nützlichen Funktionen, die dem Anwender das Arbeiten mit Polynomen erleichtern. Dies kommt vor allem dann zum Tragen, wenn Interpolationsund Approximationsaufgaben behandelt werden. Die Tabelle 37 fasst die Funktionen nochmals zusammen, siehe doc polyfun (help polyfun).

Funktion	Beschreibung
conv	Multipliziert Polynome
deconv	Dividiert Polynome
poly	Polynom aus Nullstellen
polyder	Berechnet Ableitung
polyint	Berechnet Integral
polyval	Berechnet Polynomwerte
roots	Berechnet Nullstellen

Tabelle 31: Polynome in MATLAB

54. Polynominterpolation

Gegeben sind n Punkte in der Ebene \mathbb{R}^2 , finde ein Polynom minimalen Grades, das durch alle Punkte geht. Ein solches Polynom hat höchstens den Grad n-1 und ist eindeutig bestimmt; man nennt es das Interpolationspolynom. Mit der Matlab-Funktion polyfit ist es bequem möglich, dieses Polynom zu berechnen. Als Beispiel betrachten wir die drei

Punkte (-2, -27), (0, -1) und (1, 0). Gesucht $1, 2, \ldots, m$ und gesucht ist ein Parametervektor ist also ein quadratisches Polynom $p_2(t, \mathbf{x}) =$ $x_1 + x_2t + x_3t^2$, das die drei gegebene Punkte interpoliert. Die folgenden MATLAB-Zeilen berechnen das Polynom und stellen das Problem grafisch dar.

```
t = [-2 \ 0 \ 1];
y = [-27 -1 0];
x = polyfit(t,y,length(t)-1);
ti = linspace(min(t), max(t));
yi = polyval(x,ti);
plot(ti,yi,t,y,'ro'), grid on
```

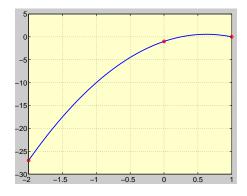


Abbildung 47: Polynominterpolation

Lässt man sich den Zeilenvektor x ausgeben, so findet man darin die gewünschten Koeffizienten des quadratischen Polynom; man erhält $p_2(t) = -1 + 5t - 4t^2$.

55. Polynomapproximation

Die Funktion polyfit kann auch verwendet werden, um Daten durch Polynome zu approximieren. Gegeben sind m Punkte (t_i, y_i) , i =

 $x \in \mathbb{R}^n$, sodass die polynomiale Modellfunktion $p(t, \mathbf{x}) = x_1 + x_2t + x_3t^2 + \dots + x_nt^{n-1}$ die gegebenen Punkte "bestmöglichst" approximiert, wobei bestmöglichst im Sinne kleinster Fehlerquadrate zu verstehen ist:

Minimiere
$$\sum_{i=1}^{m} (y_i - p(t_i, \mathbf{x}))^2$$
.
 $\mathbf{x} \in \mathbb{R}^n$

Da die gesuchten Parameter x_i in der Modellfunktion linear vorkommen, liegt eine lineare Ausgleichsaufgabe vor. Als Beispiel betrachten wir folgende Aufgabe. Angenommen es liegen die folgenden Messdaten vor, siehe Tabelle 32. Da diese Daten nahezu auf einer parabolischen Kurve liegen, entscheiden wir uns dafür, ein quadratisches Polynom durch diese Punkte zu legen. Die folgenden Matlab-Zeilen zeigen, wie man das Polynom zweiten Grades findet und wie man die Ergebnisse grafisch darstellen kann.

```
t = [0.0:0.5:10];
y = [2.9 \ 2.7 \ 4.8 \ 5.3 \ 7.1 \ 7.6 \ 7.7..]
7.6 9.4 9.0 9.6 10.0 10.2 9.7...
8.3 8.4 9.0 8.3 6.6 6.7 4.1];
x = polyfit(t,y,2);
ti = linspace(min(t),max(t));
yi = polyval(x,ti);
plot(ti,yi,t,y,'ro'), grid on
```

Lässt man sich den Zeilenvektor x ausgeben, so findet man darin die gewünschten Koeffizienten des quadratischen Polynom; man erhält $p_2(t) = 2.1757 + 2.6704t - 0.2384t^2$. Diese Polynom ist das beste Polynom zweiten Grades in dem Sinne, dass es die Summe der Fehlerquadrate minimiert.

t_i	y_i
0.0	2.9
0.5	2.7
1.0	4.8
1.5	5.3
2.0	7.1
2.5	7.6
3.0	7.7
3.5	7.6
4.0	9.4
4.5	9.0
5.0	9.6
5.5	10.0
6.0	10.2
6.5	9.7
7.0	8.3
7.5	8.4
8.0	9.0
8.5	8.3
9.0	6.6
9.5	6.7
10.0	4.1

Tabelle 32: Zur Polynomapproximation

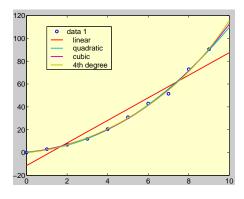


Abbildung 48: Polynomapproximation

Die Polynomapproximation bzw. Ausgleichsaufgabe ist in der Statistik von besonderer Bedeutung und wird dort unter dem Namen Regressionsrechnung (*Engl.:data fitting*) studiert.

56. Kubische Splineinterpolation

Für viele Anwendungen ist die Polynominterpolation ungeeignet, insbesondere dann, wenn der Grad des Interpolationspolynoms "groß" gewählt werden muss. Abhilfe schafft in diesem Fall eine stückweise polynomiale Interpolation, insbesondere eine kubische Splineinterpolation. In MATLAB kann man ein kubisches Splineinterpolationspolynom mithilfe der Funktion spline berechnen. Ein kubischer Spline ist ein stückweises kubisches Polynom, das zweimal stetig differenzierbar ist. Das folgende Beispiel zeigt eine kubische Splineinterpolation im Vergleich zu einer ungeeigneten Polynominterpolation anhand von neun Datenpunkte, die zur Quadratwurzelfunktion gehören. Während das Interpolationspolynom achten Grades zu Ozillationen führt, ist eine kubische Splineinterpolation gut geignet die gegebenen Punkte zu interpolieren.

```
%-Daten.
t = [0 1 4 9 16 25 36 49 64];
y = [0 1 2 3 4 5 6 7 8];
%-Polynominterpolation.
x = polyfit(t,y,length(t)-1);
ti = linspace(min(t),max(t));
yi = polyval(x,ti);
%-Kubische Splineinterpolation.
ys = spline(t,y,ti);
plot(ti,yi,t,y,'ro',ti,ys),
```

grid on, axis([0,64,0,10]),
legend('Polynom','Daten','Spline')

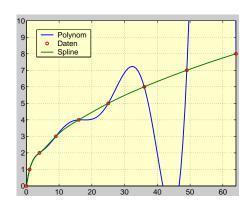


Abbildung 49: Kubische Splineinterpolation

57. Stückweise lineare Interpolation

Angenommen es sind m Punkte (t_i, y_i) gegeben. Setzt man $t = (t_1, t_2, ..., t_m)$ und $y = (y_1, y_2, \dots, y_m)$ und führt plot(t,y) aus, so werden diese m Punkte geradlinig verbunden, das heißt man sieht den Graph einer stückweisen (affin) linearen Interpolationsfunktion. Mithilfe der Funktion interp1 kann man stückweise lineare Interpolationsfunktionen berechnen. Wir zeigen dies an einen Beispiel. Angenommen es soll die stückweise lineare Interpolationsfunktion zu den Punkten (t_i, y_i) mit t = linspace(0,3,10) und y = humps(t) berechnet werden, so kann man das mit den folgenden Zeilen erreichen. Beachten Sie, dass humps eine eingebaute MATLAB-Funktion ist, deren Funktionsterm gegeben ist durch hums $(t) = 1/((t - 0.3)^2 + 0.01) + 1/((t - 0.9)^2 + 0.04) - 6.$

```
t = linspace(0,3,10);
y = humps(t);
tt = linspace(min(t),max(t));
yi = interp1(t,y,tt);
plot(tt,yi,t,y,'ro',tt,humps(tt)),
grid on,
legend('Interpolationsfunktion',...
'Punkte','humps'),
```

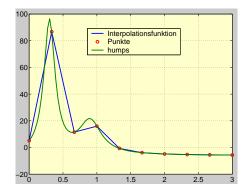


Abbildung 50: Lineare Interpolation

58. Nichtlineare Gleichungen (1)

In Matlab gibt es die eingebaute Funktion fzero zur Lösung einer nichtlinearen Gleichung in einer Variablen. Der Algorithmus ist eine Kombination aus dem Intervallhalbierungsverfahren, der Sekantenmethode und der inversen quadratischen Interpolation. Dieses hybride Verfahren hat eine lange und interessante Geschichte, siehe [7]. Es ist ein guter Algorithmus, aber leider nur für skalarwertige Funktionen mit einer skalaren Variablen ein-

setzbar. Den m-File fzero findet man im Verzeichnis Toolbox/MATLAB/funfun.

Die Buckelfunktion humps hat zwei Nullstellen, siehe Abbildung 14. Wir berechnen diese mit fzero.

```
>> fzero(@humps,1)
ans =
    1.2995
>> fzero(@humps,0)
ans =
    -0.1316
```

Das zweite Argument des Funktionsaufrufes gibt jeweils den Startpunkt des iterativen Verfahren an, $x_0 = 1$ bzw. $x_0 = 3$. Wir wollen fzero noch an der nichtlinearen Gleichung $x^2 - 4\sin(x) = 0$ illustrieren. Die Gleichung hat die beiden Nullstellen: $x_1 = 0$ und $x_2 \approx 1.9$. Man erhält diese durch zum Beispiel durch

```
>> f = @(x)x^2-4*sin(x);
>> x1 = fzero(f,1)
x1 =
-1.2470e-026
```

bzw.

```
>> x2 = fzero(f,3)
x2 =
1.9338
```

Sie können sich die einzelnen Iterationsschritte auch ausgeben lassen, indem Sie die entsprechende Option mit der optimset-Funktion setzen. Dies geht wie folgt:

Mit der Matlab-Funktion optimset können Sie verschiedene Parameter (zum Beispiel Genauigkeitswerte) von Optimierungsfunktionen setzen. Da das Lösen nichtlinearer Gleichungen eng verwandt ist mit dem Lösen von Optimierungsproblemen wird auch der Name optimset verständlich.

Parameterabhängige nichtlineare Gleichungen können Sie ebenfalls mit fzero lösen. Gleichungen mit Polynomfunktionen können Sie effizient mit der Funktion roots lösen. Nichtlineare Gleichungen mit mehr als zwei Variablen sind mit der Funktion fsolve anzugehen, siehe Abschnitt 68. Diese Funktion gehört aber zur *Optimization Toolbox* und ist deshalb nur dann verwendbar, wenn diese installiert ist.

Aufgabe 107 (Nichtlineare Gl.) Berechnen Sie mit fzero die Lösung der nichtlinearen Gleichung

$$e^{-x} = x$$

Lösung: Die Lösung findet man wie folgt:

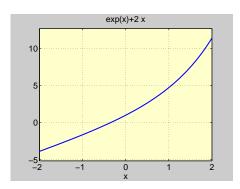
```
>> f = @(x)exp(-x)-x;
>> x = fzero(f,2)
x =
0.5671
```

Aufgabe 108 (Nichtlineare Gl.) Berechnen Sie mit fzero die Lösung der nichtlinearen Gleichung

$$e^x + 2x = 0$$

Die Abbildung 51 zeigt die Geometrie. Lösung: Die Lösung findet man wie folgt:

```
>> f = @(x) exp(x)+2*x;
>> x = fzero(f,3)
```



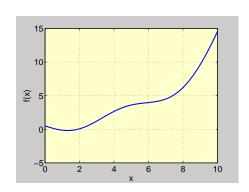


Abbildung 51: Graph von $f(x) = e^x + 2x$, $x \in \mathbb{R}$ Abbildung 52: Graph der Funktion f_1 im Intervall [0,10]

Aufgabe 109 (Nichtlineare Gl.) Finden Sie die positiven Nullstellen der folgenden Funktionen:

(a)
$$f_1(x) = 1/2e^{x/3} - \sin(x)$$

(b)
$$f_2(x) = \log(x+1) - x^2$$

(c)
$$f_3(x) = e^x - 5x^2$$

Lösung:

(a) Man zeichnet zunächst den Graphen der Funktion (s. Abbildung 52) mit

Die Nullstelle liegt offensichtlich im Intervall [0,1]. Mit

0.6772

sieht man, dass die Nullstelle der Punkt x = 0.6772 ist.

(b) Wir zeichnen zunächst den Graphen der Funktion (s. Abbildung 53) mit

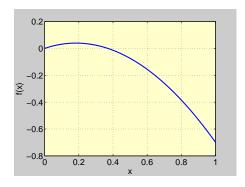


Abbildung 53: Graph der Funktion f_2 im Intervall [0,1]

Die Nullstelle liegt offensichtlich im Intervall [0.3,0.5]. Mit

```
>> fzero(f,[0.3,0.5])
ans =
0.3696
```

sieht man, dass die Nullstelle im Punkt x = 0.3696 liegt.

(c) Statt des @-Befehls kann man auch ein m-File verwenden, das die entsprechende Funktion enthält. Dazu legt man ein m-File unter dem Namen f3.m an, dass folgenden Inhalt hat:

```
function y = f3(x)
y = exp(x)-5*x.^2;
```

Mit Hilfe des Graphen der Funktion, der durch

```
>> fplot(@f3,[0 6])
>> xlabel('x'),ylabel('f(x)')
>> grid on
```

erzeugt werden kann (s. Abbildung 54), erkennt man, dass die beiden Nullstellen

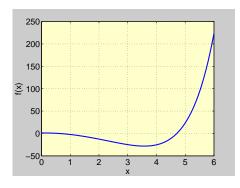


Abbildung 54: Graph der Funktion f_3 im Intervall [0,6]

im Intervall [0,1] und im Intervall [4,5] liegen. Mit

```
>> fzero(@f3,[0,1])
ans =
0.6053
```

und

```
>> fzero(@f3,[4,5])
ans =
4.7079
```

erkennt man, dass sie bei $x_1 = 0.6053$ und $x_2 = 4.7079$ liegen.

Aufgabe 110 (Exponentialgleichung) Berechnen Sie in Matlab die Lösung von

$$(0.5)^x = x$$

Lösung: Erste Möglichkeit

```
>> solve('0.5^x=x')
ans =
.64118574450498598448620048211482
```

Die Funktion solve ist eine symbolische Funktion, das heißt es wird versucht, eine symbolische Lösung zu finden. Dies ist aber nicht möglich, daher schaltet solve auf ein numerisches Verfahren um und liefert eine numerische Lösung.

Zweite Möglichkeit:

```
>> f = @(x) 0.5^x-x;
>> fzero(f,[0,1])
ans =
0.6412
```

Aufgabe 111 (Logarithmusgleichung) Berechnen Sie in Matlab die Lösung von

$$\log_{0.5} x = x$$

Lösung: Erste Möglichkeit:

```
>> solve('log(x)/log(0.5)-x')
ans =
.64118574450498598448620048211482
```

oder

```
>> solve('ln(x)/ln(0.5)-x')
ans =
.64118574450498598448620048211482
```

Die Funktion solve ist eine symbolische Funktion, das heißt es wird versucht, eine symbolische Lösung zu finden. Dies ist aber nicht möglich, daher schaltet solve auf ein numerisches Verfahren um und liefert eine numerische Lösung.

Zweite Möglichkeit:

Aufgabe 112 (Polynomgleichungen) Berechnen Sie mit roots die Lösung der Polynomgleichung

$$x^3 + 6x^2 - 11x - 6 = 0$$

Lösung: Die drei Lösungen findet man wie folgt:

```
-7.3803
1.8256
-0.4453
```

59. Optimierung (Teil 1)

Matlab besitzt zwei Funktionen, um zu optimieren: fminbnd und fminsearch. Mit fminbnd kann man ein lokales Minimum einer reellwertigen Funktion einer reellen Variablen ausfindig machen und mit fminsearch kann man eine reellwertige Funktion mehrerer Variablen minimieren.

Das Kommando x = fminbnd(fun, x1, x2) versucht von der Funktion fun über dem Intervall $[x_1, x_2]$ ein lokales Minimum zu finden. Im Allgemeinen hat eine Funktion mehrere Minima. In Matlab gibt es jedoch keine Funktion, dieses schwierige Problem zu lösen, nämlich ein globales Minimum zu finden.

Wollen Sie eine Funktion f maximieren statt minimieren, dann können Sie -f minimieren, da $Max_x f(x) = -Min_x (-f(x))$ ist.

Als Beispiel betrachten wir das eindimensionale (univariate) Optimierungsproblem:

Minimiere
$$\sin x - \cos x$$

 $x \in [-\pi, \pi]$

Die Lösung erhalten wie folgt:

```
>> f = @(x)sin(x)-cos(x);
>> [x,fWert] = fminbnd(f,-pi,pi)
x =
-0.7854
```

```
fWert = -1.4142
```

Wie bei der Funktion fzero können Sie Optionen in einer Strukturvariablen durch die optimset-Funktion spezifizieren. Der von fminbnd verwendete Algorithmus ist eine Kombination des goldenen Schnittes und parabolischer Interpolation.

Mit der Funktion fminsearch ist es möglich, ein lokales Minimum einer reellwertigen Funktion von $n \ge 1$ Variablen zu berechnen. Die Syntax ist ähnlich wie die von fminbnd: $\mathbf{x} = \text{fminsearch}(\text{fun}, \mathbf{x0}, \text{options})$, außer dass das zweite Argument $\mathbf{x0}$ ein Startvektor ist anstatt eines Intervalls. Die quadratische Funktion $f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2, (x_1, x_2) \in \mathbb{R}^2$ hat in $\mathbf{x}_* = (0, 0)$ eine Minimalstelle, denn es ist:

```
>> f = @(x)x(1)^2+x(2)^2-x(1)*x(2);

>> x = fminsearch(f,ones(2,1))

x =

1.0e-004 *

-0.4582

-0.4717
```

Das Verfahren hinter der Funktion fminsearch basiert auf dem Nelder-Mead Simplexverfahren, eine direkte Suchmethode, die ohne Ableitungsinformationen auskommt.

Es ist möglich, dass das Verfahren zu keinem lokalen Minimum gelangt oder sehr langsam konvergiert. Dagegen hat es den Vorteil, robust gegenüber Funktionsunstetigkeiten zu sein.

Weitere verfeinerte Algorithmen finden Sie in der *Optimization Toolbox*, siehe doc optim (help optim).

Aufgabe 113 (Optimierung) Lösen Sie das Problem

Minimiere
$$e^{x_1-x_2} + x_1^2 + x_2^2$$
.
 $\mathbf{x} \in \mathbb{R}^2$

Geben Sie die Lösung auf vier Nachkommastellen an.

Lösung: Die Lösung ist $x^* = (x_1^*, x_2^2) \approx (-0.2836, 0.2836)$ mit $f(x_1, x_2) \approx 0.7281$. Mit der Matlab-Funktion fminsearch gewinnt man die Lösungen wie folgt. Man schreibe eine m-File, in dem man die Zielfunktion definiert

```
function f = myf(x)
f = exp(x(1)-x(2))+x(1).^2+...
x(2).^2;
```

und ruft dann die fminsearch Funktion auf.

```
>> fminsearch(@myf,[1;1])
ans =
   -0.2836
   0.2835
```

60. FFT

Die diskrete Fourier-Transformation (DFT) ist eine unitäre lineare Abbildung von \mathbb{C}^n nach \mathbb{C}^n , wobei einem Vektor $\mathbf{x} \in \mathbb{C}^n$ der Vektor $\mathbf{y} = \mathbf{F}_n \mathbf{x}$ zugeordnet wird. Hierbei ist $\mathbf{F}_n \in \mathbb{C}^{n \times n}$ die unitäre Matrix, die die diskrete Fourier-Transformation beschreibt. Für n = 4 hat diese

die Form

$$\boldsymbol{F}_{n} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^{2} & \omega^{3} \\ 1 & \omega^{2} & \omega^{4} & \omega^{6} \\ 1 & \omega^{3} & \omega^{6} & \omega^{9} \end{bmatrix}, \quad \omega = e^{-2\pi i/4}.$$

Die schnelle Fourier-Transformation (Fast Fourier Transform = FFT) ist ein effizientes Verfahren um den Bildvektor y anstatt der obigen Matrix-Vektor Multiplikation zu berechnen. Die Funktion fft realisiert diese Methode und wird mit y = fft(x) aufgerufen. Die Effizienz von fft hängt vom Wert n ab; Primzahlen sind schlecht, zerlegbare Zahlen sind besser und Vielfache von 2 sind am Besten. Die inverse FFT, $x = n^{-1}F_ny$, wird durch die Funktion ifft ausgeführt: x = ifft(y). Wir geben ein Beispiel:

Aufgabe 114 (FFT, "Konstante") Berechnen Sie die diskrete Fourier-Transformation des Vektors $\mathbf{x} = (1, 1, 1, 1)$. Machen Sie sich jedoch erst klar, was heraurauskommen sollte.

Lösung: Es ist

Die Abbildung 55 zeigt die Vektoren x =

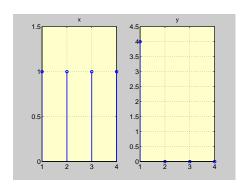


Abbildung 55: Zu Aufgabe 117

$$(1, 1, 1, 1)$$
 und $y = (4, 0, 0, 0)$. $\odot \dots \odot$

Aufgabe 115 (FFT, "Kosinus") Berechnen Sie die diskrete Fourier-Transformation des Vektors x = (1, 0, -1, 0). Überlegen Sie jedoch zuerst, was herauskommen sollte.

Lösung: Es ist

Aufgabe 116 (FFT, "Sinus") Berechnen Sie die diskrete Fourier-Transformation des Vektors x = (0, 1, 0, -1). Machen Sie sich jedoch erst klar, was heraurauskommen sollte.

Lösung: Es ist

Aufgabe 117 (FFT, "Sinus") Berechnen Sie je-

Aufgabe 117 (FFT, "Sinus") Berechnen Sie jeweils die diskrete Fourier-Transformation

(a)
$$x_1 = (4, 1, 2, 1)$$

(b)
$$x_2 = (1, 4, 1, 2)$$

(c)
$$x_3 = (4, 0, 1, 0, 2, 0, 1, 0)$$

Lösung: Es ist

Es existieren in Matlab auch Funktionen zur mehrdimensionalen diskreten Fourier-Transformation: Siehe doc fft2 (help fft2), doc fftn (help fftn), doc ifft2 (help ifft2) und doc ifftn (help ifftn).

Die FFT-Algorithmen in Matlab werden durch das Softwarepaket FFTW (the "Fastest Fourier Transform in the West") realisiert. Dieses Paket ist ein Beispiel einer adaptiven Software in dem Sinne, dass dieses dafür sorgt, dass die Ausführungszeit am schnellsten ist, unabhängig davon auf welcher Computerumgebung man sich gerade befindet. Die Funktion fftw stellt Parameter zur Verfügung um diesen Prozess zu tunen, siehe doc fftw (help fftw).

Bemerkungen zur kontinuierlichen Fourier-Transformation findet man im Abschnitt 67.20.

61. Integration

Unter numerischer Integration versteht man die Berechnung einer Approximation an bestimmte Integrale der Form $\int_a^b f(x)dx$. Matlab stellt die beiden Funktionen quad und quadl als adaptive Integrationsfunktionen zur Verfügung. Sie berechnen eine Näherung an das bestimmte Integral $\int_a^b f(x)dx$. Hierbei müssen die Integrationsgrenzen a und b endlich sein und der Integrand f(x) darf im Intervall [a,b] keine Singularitäten besitzen.

Wir diskutieren die Funktion quad; quadl funktioniert analog. Ein Aufruf der Form

berechnet das bestimmte Integral. Der Integrand fun kann sowohl in einem m-File als auch als @-Funktion definiert werden. Er muss jedoch in vektorisierter Form vorliegen, so dass ein Vektor zurückgegeben werden kann, wenn das Argument x von f ein Vektor ist.

Wegen

```
>> quad(@humps,0,1)
ans =
29.8583
```

ist der Flächeninhalt unter dem Graph von humps im Intervall [0, 1] ungefähr gleich 30. Die Abbildung 56 zeigt die Geometrie in Form

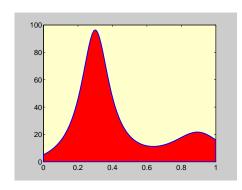


Abbildung 56: Fläche unter humps in [0, 1]

von Graph und Flächeninhalt. Die Abbildung wurde dabei mit folgenden Anweisungen erstellt:

```
>> x = linspace(0,1);
>> y = humps(x);
>> area(x,y,'FaceColor','r',...
'LineWidth',2,'EdgeColor','b');
```

Das bestimmte Integral

$$\int_0^1 x^x dx$$

kann von Matlab nicht exakt (symbolisch) berechnet werden.

Das Integral wird unverändert als nichtberechenbar ausgegeben. Wir können das Integral aber numerisch berechnen. Es ist:

```
>> quad(@(x)x.^x,0,1)
ans =
0.7834
```

Die Abbildung 57 zeigt die dazugehörige Geo-

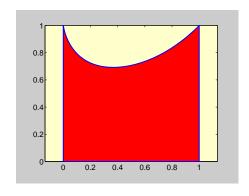


Abbildung 57: Fläche unter x^x in [0, 1]

metrie.

Aufgabe 118 (Numerische Integration) Berechnen Sie die Länge der Zykloide

$$f: \mathbb{R} \to \mathbb{R}^2$$
$$t \mapsto (t - \sin t, 1 - \cos t)$$

von t = 0 bis $t = 2\pi$. Lösung: Es ist $f'(t) = (1 - \cos t, \sin t)$ und somit gilt

$$L_f = \int_0^{2\pi} |f'(t)| dt$$
$$= \int_0^{2\pi} \sqrt{(1 - \cos t)^2 + \sin^2 t} \, dt$$

und damit in Matlab

```
>> f = @(t)sqrt((1-cos(t)).^2+...
sin(t).^2);
>> quad(f,0,2*pi)
ans =
    8.0000
```

Aufgabe 119 (Numerische Integration) Berechnen Sie

$$\int_2^4 x \ln(x) dx.$$

Lösung: Ist die Funktion

```
function f = fxlog(x)
f = x.*log(x);
```

definiert, so erhält man nach Eingabe von

```
>> quad(@fxlog,2,4)
ans =
6.7041
```

eine Approximation an das zu berechnende Integral.

Die Funktion quad basiert auf der Simpson-Regel, welche eine Newton-Cotes 3-Punkt Regel ist, das heißt sie ist für Polynome bis zum

Grad 3 exakt. Die Funktion quad1 setzt eine genauere Gauss-Lobatto 4-Punkt Regel mit einer 7-Punkt Kronrod Erweiterung ein. Somit ist diese für Polynome bis zum Grad 5 bzw. 9 genau. Beide Funktionen arbeiten adaptiv. Sie unterteilen das Integrationsintervall in Teilintervalle und verwenden über jedem Teilintervall die zugrundeliegende Integrationsregel. Die Teilintervalle werden aufgrund des lokalen Verhaltens des Integranden gewählt, wobei das kleinste Intervall dort ausgesucht wird, wo der Integrand sich am meisten ändert. Eine Warnung wird ausgegeben, wenn die Teilintervalle zu klein werden oder falls zu viele Funktionsauswertungen gemacht werden müssen, worauf oft geschlossen werden kann, dass der Integrand eine Singularität besitzt.

61.1. Mehrfachintegrale

Man spricht von einem zweifachen Integral (Doppelintegral) wenn es die Form

$$\int \int_D f(x, y) dx dy$$

hat. Hierbei ist D ein beschränktes Gebiet im zweidimensionalen Raum \mathbb{R}^2 . Ein dreifaches Integral (Dreifachintegral) liegt vor, wenn die Gestalt

$$\int \int \int_G f(x, y, z) dx dy dz$$

hat, wobei G ein beschränktes Gebiet im dreidimensionalen Raum \mathbb{R}^3 ist. Analog sind vierund mehrfache Integrale definiert. Die Berechnung mehrfacher Integrale lässt sich auf die Berechnung mehrerer einfacher Integrale zurückführen.

Doppelintegrale können mit dblquad numerisch berechnet werden. Angenommen wir wollen das Integral

$$\int_{y=4}^{6} \int_{x=0}^{1} (y^2 e^x + x \cos y) dx dy$$

approximieren, dann können wir wie folgt verfahren. Zunächst definieren wir den Integranden, der nun ein Funktionsterm von zwei unabhängigen Variablen ist.

```
function out = fxy(x,y)
out = y^2*exp(x)+x*cos(y);
```

Dann geben wir ein:

```
>> dblquad(@fxy,0,1,4,6)
ans =
87.2983
```

Die Funktion, die dblquad im ersten Argument übergeben wird, muss einen Vektor x und einen Skalar y vertragen können. Zurückgegeben wird ein Vektor. Zusätzliche Argumente für dblquad sind möglich, um die Toleranz und die Integrationsmethode zu bestimmen. Standardmäßig ist dies quad.

Aufgabe 120 (Doppelintegral) Berechnen Sie das Integral

$$\int_{y=0}^{2} \int_{x=0}^{1} (x^2) dx dy$$

Lösung: Es ist

Dreifachintegrale können mit triplequad berechnet werden.

```
>> triplequad(@(x,y,z)...
(y*sin(x)+z*cos(x)),0,pi,0,1,-1,1)
ans =
2.0000
```

Aufgabe 121 (Dreifachintegral) Berechnen Sie das Integral

$$\int_{z=-1}^{1} \int_{y=0}^{1} \int_{x=0}^{\pi} (x^3) dx dy dz$$

Lösung: Es ist

```
>> triplequad(@(x,y,z) x.^3,...
0, pi, 0, 1, -1, 1)
ans =
48.7045
```

61.2. Tabellarische Daten

Eine weitere Integrationsroutine ist trapz, welche wiederholt die Trapezregel anwendet. Sie unterscheidet sich von quad und quadl dadurch, dass die Eingabeargumente Vektoren x_i , $f(x_i)$ und nicht der analytische Funktionsterm f(x) ist. Deshalb kann die Funktion auch nicht adaptiv arbeiten. Wir zeigen die Funktionsweise an einem Beispiel. Der exakte Wert von $\int_0^{\pi} \sin x \, dx$ ist 2. Wir bestätigen dies mit der Funktion trapz. Hierzu diskretisieren wir das Intervall $[0,\pi]$ gleichmäßig und berechnen die dazugehörigen Sinusfunktionswerte, damit die Eingabeargumente tabellarisch vorliegen.

```
>> x = 0:pi/50:pi;
>> y = sin(x);
```

Als Näherung erhalten wir somit

```
>> trapz(x,y)
ans =
1.9993
```

Wir zeigen noch ein Beispiel für eine nicht gleichmäßige Diskretisierung des Intervalls $[0,\pi]$

```
>> x = sort(rand(1,101)*pi);
>> y = sin(x);
>> trapz(x,y)
ans =
1.9983
```

Im allgemeinen sind jedoch die Funktionen quad und quad1 vorzuziehen, wenn der Integrand analytisch verfügbar ist.

Steht der Integrand f in tabellarischer Form $(t_i, f(t_i)), i = 1, 2, ..., m$ zur Verfügung und ist man an der tabellarischen Form der Integralfunktion J_{t_1} mit

$$J_{t_1}(x) = \int_{t_1}^x f(t)dt$$

interessiert, so kann man die Matlab-Funktion cumtrapz verwenden. Das bestimmte Integral von t_1 bis zu irgendeinem Punkt $x = t_i$ wird dann durch die Trapezregel berechnet. Der Aufruf

```
z = cumtrapz(t,f)
```

liefert einen Vektor z zurück, der die gleiche Länge wie t hat und in dem die Werte $J_{t_1}(x = t_i)$ für i = 1, 2, ..., m stehen. Das folgende

Beispiel zeigt die Funktionsweise der Funktion cumtrapz an der humps-Funktion.

```
>> t = linspace(-1,2);
>> f = humps(t);
>> z = cumtrapz(t,f);
```

Die Abbildung 58 zeigt den Graph der humps-

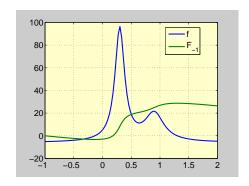


Abbildung 58: humps-Funktion und Integralfunktion im Intervall [-1, 2]

Funktion und den Graph der dazugehörigen Integralfunktion, die mit cumtrapz berechnet wurde. Die Abbildung wurde einfach mit

```
>> plot(t,f,t,z), grid,
>> legend('f','F_{-1}')
```

erzeugt.

Aufgabe 122 (Integration) Zeichnen Sie den Graph der Integralfunktion F_{-3} von $f(t) = 1/\sqrt{2\pi}e^{-t^2/2}$, $t \in \mathbb{R}$ im Intervall von -3 bis 3, und in dieselbe Figur auch den Graph von f. Kennen Sie die Bedeutung der Funktionen f und $F_{-\infty}$.

Lösung: Die Abbildung 59 zeigt die Graphen. f ist die Dichtefunktion und $F_{-\infty}$ ist die Verteilungsfunktion der Standardnormalverteilung

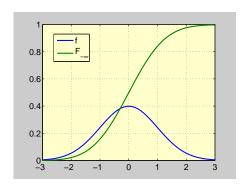


Abbildung 59: Die Graphen zu Aufgabe 122

aus der Statistik. Der dazugehörige Matlab-Code ist

```
>> t = linspace(-3,3,500);
>> f = 1/sqrt(2*pi)*exp(-t.^2/2);
>> z = cumtrapz(t,f);
>> plot(t,f,t,z), grid,
>> legend('f','F_{-\infty}')
```

61.3. Numerische uneigentliche Integration

Die numerische Integration kann auch bei uneigentlicher Integration eingesetzt werden, zum Beispiel etwa für $\int_0^\infty f(x)dx$. Eine Möglichkeit besteht darin, einen Punkt α zu finden, so dass der Wert von $\int_{\alpha}^\infty f(x)dx$ im Vergleich zu $\int_0^\alpha f(x)dx$ vernachlässigt werden kann; man beschränkt sich dann auf die Berechnung des letzten Integrals mit einer bekannten Quadraturformel. Wir zeigen die Idee an dem Beispiel $\int_1^\infty 1/x^5dx = 1/4$, siehe auch Aufgabe 167. Statt $\int_1^\infty 1/x^5dx$ berechnen wir $\int_1^7 1/x^5dx$. Wir

erhalten

```
>> quad(@(x) 1./x.^5,1,7)
ans =
0.2499
```

was für $\int_{1}^{\infty} 1/x^5 dx = 1/4$ eine sehr gute Näherung darstellt.

61.4. Zusammenfassung

Zusammenfassend können wir sagen: Mat-LAB stellt zur numerischen Berechnung von Integralen mehrere Funktionen zur Verfügung. Liegt die zu integrierende mathematische Funktion f in analytischer Form vor, so kann der Funktionsterm in einem m-File oder als anonyme Funktion definiert werden. Die Matlab-Funktionen quad, quadl, quadv, dblquad, triplequad und alle Funktionen zur numerischen Lösung von Differenzialgleichungen, siehe Abschnitt 62, können dann eingesetzt werden. Die Funktion quadv ist eine vektorrielle Form der Funktion quad. Liegt der Integrand tabellarisch vor, so sind die Funktionen trapz und cumtrapz zu verwenden. Zur symbolischen Berechnung von Integralen stehen die Funktionen int und dsolve zur Verfügung, siehe Abschnitt 67.12 und 67.19.

62. Differenzialgleichungen

Eine Gleichung zur Bestimmung einer Funktion heißt Differenzialgleichung, wenn sie mindestens eine Ableitung der gesuchten Funktion enthält. Die Ordnung der in der Differenzialgleichung vorkommenden höchsten Ableitung der gesuchten Funktion heißt Ordnung der Differenzialgleichung. Hängt die in der Differenzialgleichung gesuchte Funktion nur von einer Variablen ab, so nennt man die Differenzialgleichung gewöhnlich. Enthält die Differenzialgleichung partielle Ableitungen, so heißt sie partiell. Zum Beispiel ist die Bestimmungsgleichung für die reellwertige Funktion y einer Variablen t

$$y'(t) = ay(t)$$

eine gewöhnliche Differenzialgleichung. Hierbei ist $a \neq 0$ eine reelle Konstante. Die Gleichung

$$u_{tt}(x,t) = c^2 u_{xx}(x,t)$$

ist eine partielle Differenzialgleichung. Gesucht ist die reellwertige Funktion u mit den beiden unabhängigen Variablen x und t, wobei $c \neq 0$ eine reelle Konstante ist. Siehe Abschnitt 67.19 für symbolisches Lösen von Differenzialgleichungen.

62.1. Anfangswertaufgaben

In Matlab gibt es mehrere Funktionen, um Anfangswertaufgaben von gewöhnlichen Differenzialgleichungen numerisch zu lösen. Anfangswertaufgaben haben folgende Form

AWA:
$$\begin{cases} \frac{d}{dt}\mathbf{y}(t) = \mathbf{f}(t, \mathbf{y}(t)) \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases} \quad t_0 \le t \le t_f$$

wobei t eine reelle Variable, y eine unbekannte vektorwertige Funktion und die gegebene Funktion f ebensfalls vektorwertig ist. Konkret kann man sich t als die Zeit vorstellen. Die Funktion f bestimmt die gewöhnliche Differenzialgleichung und zusammen mit

der Anfangsbedingung $y(t_0) = y_0$ wird die Anfangswertaufgabe definiert. Der einfachste Weg solch ein Problem in Matlab zu lösen, besteht darin, eine Funktion zu schreiben, die f auswertet und dann einen der Mat-LAB-Löser aufzurufen. Die geringste Information, die man dem Löser mitteilen muss, ist der entsprechende Funktionsname, das Intervall $[t_0, t_f]$ wo man die Lösung sucht und die Anfangsbedingung y_0 . Zusätzlich können weitere extra Ein- und Ausgabeargumente optional angegeben werden, die mehr über das mathematische Problem aussagen und wie es gelöst werden soll. Jeder einzelne Löser ist in speziellen Situationen besonders effizient, jedoch können sie prinzipiell gegeneinander ausgetauscht werden. Im nächsten Anschnitt zeigen wir Beispiele, die den Löser ode45 illustrieren. Diese Funktion realisiert ein adaptives Runge-Kutta Verfahren und ist für die meisten Probleme effizient.

Um das skalare (m = 1) Anfangswertproblem ($0 \le t \le 3$)

AWA:
$$\begin{cases} \frac{d}{dt}y(t) = -y(t) - 5e^{-t}\sin(5t) \\ y(0) = 1 \end{cases}$$

mit ode45 zu lösen, kreieren wir als erstes den m-File rSeite.m, der die rechte Seite der Differenzialgleichung definiert.

```
function dy = rSeite(t,y)
dy = -y-5*exp(-t)*sin(5*t);
```

Danach machen wir folgende Eingabe

```
tspan = [0,3]; y0 = 1;
[t,y] = ode45(@rSeite,tspan,y0);
plot(t,y,'-'), grid,
```

xlabel t, ylabel y(t)

Dies erzeugt die Abbildung 60 (Beachten Sie, dass wir beim Setzen der *x*- und *y*-Labels die Kommando/Funktionsdualität ausgenutzt haben). Die Eingabeargumente von ode45 sind

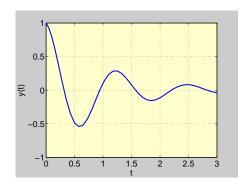


Abbildung 60: Skalares Anfangswertproblem

die Funktion rSeite, der 2-Vektor tspan, der die Zeitspanne der Simulation spezifiziert und der Anfangswert y0. Zwei Argumente t und y werden zurückgegeben. Die t-Werte liegen alle im Intervall [0,3] und y(i) approximiert die Lösung zur Zeit t(i). Die Werte t(2:end-1) werden von der Funktion ode45 automatisch gewählt. In den Bereichen, wo sich die Lösungsfunktion rapid ändert, werden mehr Punkte ausgewählt. Für weitere Informationen siehe doc funfun (help funfun).

Mit der Funktion ode15i kann man Anfangswertaufgaben, deren Differenzialgleichung in impliziter Form vorliegt, numerisch lösen. Wir betrachten ein skalares Problem. Die Eingabe ist entsprechend den anderen ODE-Funktionen: ode15i(odefun,tspan,y0,yp0). Das erste Argument spezifiziert die Funktion

f(t, y, y'), das Zweite und Dritte den Simulationsbereich (Integrationsintervall) $[t_0, t_f]$ bzw. den Anfangswert $y(t_0)$ und das vierte Argument den Anfangswert der Ableitung, das heißt $y'(t_0)$. Als Beispiel betrachten wir das Anfangswertproblem

AWA:
$$\begin{cases} y'(t)y(t) + t = 0\\ y(0) = 1 \end{cases}$$

im Intervall $[t_0, t_f] = [0, 0.9]$. In diesem Problem liegt die Differenzialgleichung y'(t)y(t) + t = 0 in impliziter Form vor; sie ist von erster Ordnung. Ich habe diese Differenzialgleichung gewählt, weil deren analytische Lösung bekannt ist und man diese so mit dem numerischen Ergebnis vergleichen kann. Die allgemeine implizite Lösung von y'(t)y(t) + t = 0 ist die Kreisgleichung $t^2 + y^2 = c$, $c \ge 0$. Wir bestätigen dieses Ergebnis symbolisch mit der Funktion dsolve, siehe Abschnitt 67.19.

Mit der Anfangsbedingung y(0) = 1 ergibt sich die Konstante zu 1 und die Lösung ist

$$y(t) = \sqrt{1 - t^2}.$$

Hier die Bestätigung:

Wir lösen die Anfangswertaufgabe nun numerisch mit ode15i. Die Funktion f definieren

wir im m-File fuerode15i wie folgt:

```
function res = fuerode15i(t,y,yp)
res = yp*y+t;
```

Dann ergibt sich die numerische Lösung durch die Anweisungen:

```
y0 = 1; yp0 = 0;
t0 = 0; tf = 0.9;
[t,y] = ode15i(@fuerode15i,...
[t0 tf],y0,yp0);
```

Die Abbildung 61 zeigt die numerische und

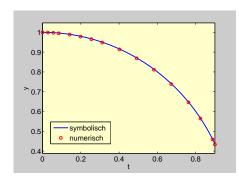


Abbildung 61: Implizites
Anfangswertproblem

symbolische (analytische) Lösung der impliziten Anfangswertaufgabe. Die Abbildung wurde mit den Anweisungen

```
ezplot(ytrue,[0,0.9])
hold on
plot(t,y,'ro')
```

erzeugt. Für weitere Informationen siehe doc ode15i (help ode15i).

Die folgende Aufzählung gibt eine komplette Übersicht über alle AWA-Löser in Matlab. Dazu wird angegeben für welchen Problemtyp der Löser geeignet ist und welches Verfahren dahinter steht.

- ode45 für nicht steife Differenzialgleichungen. Verfahren: Runge-Kutte-Verfahren der Ordnungen 4 und 5
- ode23 für nicht steife Differenzialgleichungen. Verfahren: Runge-Kutte-Verfahren der Ordnungen 2 und 3
- ode113 für nicht steife Differenzialgleichungen. Verfahren: Explizites lineares Mehrschrittverfahren der Ordnungen 1 bis 13
- ode15s für steife Differenzialgleichungen.
 Verfahren: Implizites lineares Mehrschrittverfahren der Ordnungen 1 bis 5
- ode15i für implizite Differenzialgleichungen. Verfahren: Implizites lineares Mehrschrittverfahren der Ordnungen 1 bis 5
- ode23s für steife Differenzialgleichungen. Verfahren: Modifiziertes Rosenbrock-Verfahren der Ordnungen 2 und 3
- ode23t für mäßig steife Differenzialgleichungen. Verfahren: Implizite Trapezregel der Ordnungen 2 und 3
- ode23tb für steife Differenzialgleichungen. Verfahren: Implizites Runge-Kutta-Verfahren der Ordnungen 2 und 3

Nicht alle schwierigen Probleme sind steif, aber alle steifen Probleme sind schwierig für ODE-Löser, die nicht für steife Probleme ausgelegt sind.

Die Entwickler dieser Algorithmen, SHAMPINE und REICHELT haben Ergebnisse ihrer Arbeit in [24] dargestellt. Die Funktionen sind so konstruiert, dass man sie gegeneinander leicht aus-

tauschen kann. So ist es zum Beispiel leicht möglich den ode45-Löser aus den obigen Beispielen durch einen anderen Löser auszutauschen. Aus [24] entnehmen wir:

The experiments reported here and others we have made suggest that except in special circumstances, ode45 should be the code tried first. If there is reason to believe the problem to be stiff, or if the problem turns out to be unexpectedly difficult for ode45, the code ode15s code should be tried.

62.2. Randwertaufgaben

Treten in den Bedingungsgleichungen zur eindeutigen Charakterisierung der Lösung einer Differenzialgleichung die Funktions- und Ableitungswerte der gesuchten Funktion nicht nur—wie bei Anfangswertproblemen (AWP)— an einer einzigen, sondern an zwei Stellen auf, und ist man nur an einer Lösung auf dem Intervall zwischen diesen beiden Stellen interessiert, so spricht man von einer Randwertaufgabe (RWA) (Randwertproblem (RWP); englich: Boundary Value Problem (BVP)).

Die Funktion bvp4c realisiert eine Kollokationsmethode, um Zweipunkt-Randwertaufgaben zu lösen. Diese Systeme können in folgender Form geschrieben werden:

RWA:
$$\begin{cases} \frac{d}{dx}y(x) = f(x, y(x)) \\ g(y(x_0), y(x_f)) = \mathbf{0} \end{cases} \quad x_0 \le x \le x_f$$

Hier ist, wie auch bei Anfansgwertaufgaben, y eine unbekannte vektorwertige Funktion und f eine gegebene vektorwertige Funktion von

x und y. Die Lösung wird auf dem Intervall $[x_0, x_f]$ erwartet und die gegebene Funktion g spezifiziert die Nebenbedingungen. Bei Randwertaufgaben ist es üblich, die unabhängige Variable mit x statt mit t zu bezeichnen, weil sie bei Anwendungen meist eine Ortsvariable ist. Dies ist auch konsistent mit der Matlab-Dokumentation. Das Lösen von Randwertaufgaben ist im Allgemeinen anspruchsvoller als das Lösen von Anfangswertaufgaben. Insbesondere ist es nicht ungewöhnlich, dass eine Randwertaufgabe mehrere Lösungen hat. Daher ist es beim Aufruf von bvp4c notwendig, eine Schätzung für die Lösungsfunktion mitanzugeben.

Als Beispiel betrachten wir die Zweipunkt-Randwertaufgabe:

RWA:
$$\begin{cases} u'' = 6x \\ u(0) = 0 & u(1) = 1 \end{cases} 0 \le x \le 1.$$

Diese ist analytisch lösbar und hat die eindeutige Lösung

$$u(x) = x^3$$
.

Dies sieht man so. Die allgemeine Lösung der Differenzialgleichung u'' = 6x ist

$$u(x) = x^3 + c_1 x + c_2$$
.

Die Randbedingungen bestimmen dann die Konstanten c_1 und c_2

$$0 = u(0) = c_2 \implies c_2 = 0$$

 $1 = u(1) = 1 + c_1 \implies c_1 = 0$

Wir verwenden nun bvp4c, um diese Lösung zu bestätigen. Im einfachsten Fall hat bvp4c drei Funktionsargumente: Eine Funktion, in der das Differenzialgleichungssystem definiert ist, eine Funktion mit den Randbedingungen und eine dritte Funktion, die ein Startgitter sowie eine Anfangsvermutung für die Lösungsfunktion auf diesem Gitter beinhaltet. Das Differenzialgleichungssystem wird genauso gehandhabt wie bei den Anfangswertproblemen, nämlich als System erster Ordnung. Es ist daher als erstes notwendig, die Differenzialgleichung zweiter Ordnung in ein System erster Ordnung umzuschreiben. Wir schreiben gleich das ganze Randwertproblem zweiter Ordnung in ein System erster Ordnung. Mit den Definitionen $y_1(x) = u(x)$ und $y_2(x) = u'(x)$ erhalten wir die Randwertaufgabe erster Ordnung:

RWA:
$$\begin{cases} y_1'(x) = y_2(x) \\ y_2'(x) = 6x \\ y_1(0) = 0, y_1(1) = 1 \end{cases} \quad 0 \le x \le 1$$

Das Differenzialgleichungssystem wird nun in der Funktion odes definiert

```
function dydx = odes(x,y)
dydx = [y(2);6*x];
```

Die Randbedingungen werden durch eine Residuenfunktion ausgedrückt. Die beiden Randbedingungen $y_1(0) = 0$, $y_1(1) = 1$ werden dann wie folgt (zum Beispiel in der Funktion bcs) programmiert:

```
function res = bcs(yx0,yxf)
res = [yx0(1);yxf(1)-1];
```

Als Startlösungsfunktionen wählen wir die Nullfunktion.

```
function yinit = guess(x)
yinit = [0;0];
```

Die folgenden Matlab-Zeilen berechnet die Lösung und stellt sie grafisch dar.

Der Aufruf der Funktion bypinit kreiert die Struktur solinit, welche die Daten beinhaltet, die durch Auswerten von guess an fünfäquidistanten Punkten von 0 bis 1 berechnet werden; in diesem Fall alle Null.

Im Allgemeinen hat ein Aufruf von bvp4c die Form:

```
sol = bvp4c(@odefun,@bcfun,solinit,
options,p1,p2,...)
```

In odefun wird das Differenzialgleichungssystem ausgewertet, in befun werden die Randbedingungen ausgewertet, solinit beschreibt den Anfangszustand, in options können Optionen übergeben werden und die Eingabeargumente p1, p2 erlauben zusätzliche Parameter.

Zusammenfassend sind für die Durchführung folgende Schritte empfehlenswert:

- Schreiben Sie das Problem als Randwertproblem erster Ordnung.
- 2. Schreiben Sie eine Matlab-Funktion, in der das System erster Ordnung definiert ist.
- 3. Schreiben Sie eine Funktion für die Randbedingungen.
- 4. Schreiben Sie eine Funktion für die Startnäherung.
- 5. Rufen Sie den RWA-Löser bvp4c auf.

6. Zeigen Sie die Resultate.

Aufgabe 123 (Randwertaufgaben) Wir betrachten die Randwertaufgabe

RWA:
$$\begin{cases} y'' = -y \\ y(0) = 3, y(\pi/2) = 7 \end{cases} \quad 0 \le x \le \pi/2.$$

(a) Zeigen Sie, dass

$$y(x) = 7\sin(x) + 3\cos(x), x \in [0, \pi/2]$$

die eindeutige Lösung ist.

(b) Bestätigen sie (a), indem Sie die Lösung mit der Funktion bvp4c numerisch berechnen.

Lösung:

(a) Die allgemeine Lösung der Differenzialgleichung ist

$$y(x) = c_1 \sin(x) + c_2 \cos(x).$$

Die unbekannten Konstanten c_1 und c_2 können dann aus den Randbedingungen bestimmt werden:

$$3 = y(0) = c_1 \sin(0) + c_2 \cos(0) = c_2$$

und

$$7 = y(\pi/2) = c_1 \sin(\pi/2) + c_2 \cos(\pi/2)$$
$$= c_1.$$

Die Lösung der Randwertaufgabe ist somit

$$y(x) = 7\sin(x) + 3\cos(x), x \in [0, \pi/2].$$

(b) Die numerische Lösung mit der Mat-LAB-Funktion bvp4c erledigt der folgende Function-File:

```
function RWA
   solinit = bvpinit(linspace(0,...
   pi/2,5),@guess);
   sol = bvp4c(@odes,@bcs,solinit);
   xint = linspace(0,pi/2);
   sxint = deval(sol,xint);
   plot(xint,sxint(1,:),'r')
   % Subfunctions.
   function dydx = odes(x,y)
   dydx = [y(2); -y(1)];
13
   function res = bcs(ya,yb)
   res = [ya(1)-3;yb(1)-7];
16
   function yinit = guess(x)
17
   yinit = [0;0];
```

62.3. Partielle Differenzialgleichungen

Mit der Funktion pdepe kann man partielle Differenzialgleichungen lösen. Matlabs pdepe löst eine Klasse von parabolischen bzw. elliptischen partiellen Differenzialgleichungssysteme (PDE: Partial Differential Equation). Die allgemeine Klasse hat die Form:

$$c(x,t,u,u_x)u_t = x^{-m}(x^m f(x,t,u,u_x))_x + s(x,t,u,u_x),$$

wobei u die gesuchte Funktion ist. Sie hängt von den Variablen x (Raumvariable) und t (Zeitvariable) ab und kann vektorwertig sein. Für die unabhängigen Variablen x und t gilt: $x_0 \le x \le x_f$ und $t_0 \le t \le t_f$. Die Zahl m kann die Werte 0, 1 oder 2 haben, je nachdem,

ob keine, Zylinder- oder Kugelsymmetrie vorliegt. Die Funktion c ist matrixwertig und die Fluß- bzw. Quellenfunktionen f, s sind vektorwertig. Anfangs- und Randbedingungen müssen in folgender Form zur Verfügung gestellt werden. Für $x_0 \le x \le x_f$ und $t = t_0$ muss die Lösung gleich $u_0(x)$ sein, wobei die Funktion u_0 gegeben ist. Für $x = x_0$ und $t_0 \le t \le t_f$ muss die Lösung dem Gleichungssystem

$$p_{x_0}(x, t, \mathbf{u}) + q_{x_0}(x, t)\mathbf{f}(x, t, \mathbf{u}, \mathbf{u}_x) = \mathbf{0}$$

genügen, wobei die Funktionen p_{x_0} und q_{x_0} gegeben sind. Ähnlich muss für $x = x_f$ und $t_0 \le t \le t_f$

$$p_{x_f}(x, t, u) + q_{x_f}(x, t) f(x, t, u, u_x) = 0$$

gelten, wobei die Funktionen p_{x_f} und q_{x_f} gegeben sind. Für weitere Details siehe doc pdepe (help pdepe).

Ein Aufruf von pdepe hat die allgemeine Form

was der Syntax von bvp4c gleicht.

Als Beispiel betrachten wir die eindimensionale Wärmeleitungsgleichung

$$u_t(x,t) = u_{xx}(x,t)$$

für $0 \le x \le 1$ und $0 \le t \le 0.5$ mit der Anfangsbedingung

$$u(x,0) = \sin(\pi x)$$

und den Randbedingungen

$$u(0,t) = u(1,t) = 0.$$

Dieses Problem ist mit pdepe lösbar, da es die von pdepe erlaubte Form besitzt. Es ist m=0, c(x,t,u)=1, $f(x,t,u,u_x)=u_x$ und $s(x,t,u,u_x)=0$. Für $x=x_0$ sind p(x,t,u)=u und q(x,t,u)=0 die Randbedingungen, und für $x=x_f$ haben wir p(x,t,u)=u und q(x,t,u)=0. Die Function pdeWaerme realisiert die Lösung des Problems.

```
function pdeWaerme
  %Löst die eindimensionale Wärme-
  %leitungungsgleichung mit den
  %angegebenen Anfangs- und Rand-
  %bedingungen.
   m = 0;
   t0 = 0;
   tf = 0.5;
   x0 = 0;
   xf = 1;
   xmesh = linspace(x0,xf,40);
   tspan = linspace(t0,tf,20);
   sol = pdepe(m,@pdefun,...
    @pdeic,@pdebc,xmesh,tspan);
14
   u = sol(:,:,1);
15
   mesh(xmesh,tspan,u)
16
   xlabel('x'), ylabel('t'),
17
   zlabel('u')
18
19
   %Subfunctions.
20
   %-----
21
   function [c,f,s] =
22
             pdefun(x,t,u,DuDx)
23
  %Differenzialgleichung.
24
   c = 1;
   f = DuDx;
   s = 0;
27
   function u0 = pdeic(x)
   %Anfangsbedingungen.
30
   u0 = sin(pi*x);
31
  %-----
  function [px0,qx0,pxf,qxf] =
```

```
pdebc(x0,u0,xf,uf,t)
   %Randbedingungen.
35
   px0 = u0;
   qx0 = 0;
   pxf = uf;
   qxf = 0;
```

Hierbei haben wir die Funktion linspace verwendet, um ein Gitter zwischen x_0 und x_f , sowie eines zwischen t_0 und t_f zu erzeugen. Um die Lösung zu plotten, haben wir mesh verwendet. Die Abbildung 62 zeigt das Ergebnis.

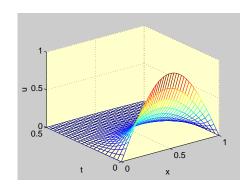


Abbildung 62: Lösung der PDE

Aufgabe 124 (Partielle DGL) Bestätigen Sie, dass

$$u(x,t) = e^{-\pi^2 t} \sin(\pi x)$$

eine Lösung des Wärmeleitungsproblems

$$\begin{cases} u_t(x,t) = u_{xx}(x,t) \\ u(x,0) = \sin(\pi x) & 0 \le x \le 1, t \ge 0 \\ u(0,t) = u(1,t) = 0 \end{cases}$$

Lösung: Leiten wir $u(x,t) = e^{-\pi^2 t} \sin(\pi x)$ Die Funktion factorial berechnet die nach t ab, so erhalten wir die linke Seite der Fakultät-Funktion n!. Da in doppelter Genau-

Differenzialgleichung

$$u_t(x,t) = -\pi^2 \sin(\pi x)e^{-\pi^2 t}$$
.

Leiten wir $u(x, t) = e^{-\pi^2 t} \sin(\pi x)$ zweimal nach x ab, so erhalten wir die rechten Seite der Differentialgleichung

$$u_{xx}(x,t) = -\pi^2 \sin(\pi x) e^{-\pi^2 t}$$
.

Da beide Seiten für alle x und t gleich sind, ist

63. Statistik

Vorhersagen sind schwierig – vor allem über die Zuknnft NIELS BOHR.

Mit der Statistics Toolbox stehen Ihnen Statistikfunktionen zur Verfügung. Die Tabelle 33 zeigt ein paar grundlegende Funktionen für die beschreibende Statistik. Ein paar Dichtefunktionen sind in Tabelle 34 zu sehen.

Mit doc stats (help stats) erhalten Sie einen Überblick über alle Funktionen der Statistics Toolbox.

64. Kombinatorik

Wir zeigen, wie man in MATLAB kombinatorische Probleme lösen kann.

64.1. Fakultäten, Binomial- und Polynomialzahlen

Beschreibende Statistik		
corrcoef	Korrelationsmatrix	
cov	Kovarianzmatrix	
geomean	Geometrischer Mittelwert	
mad	Mittlere absolute Abw.	
mean	Arithmetischer Mittelwert	
median	Zentralwert (Median)	
mode	Häufigster Wert	
range	Spannweite	
std	Standardabweichung	
tabulate	Häufigkeitstabelle	
var	Varianz	

Tabelle 33: Grundlegende Funktionen

Dichtefunktionen		
binopdf	Binomialdichte	
geopdf	Geometrische Dichte	
normpdf	Normaldichte	
unidpdf	Diskrete gleichmäßige Dichte	
unifpdf	Stetige gleichmäßige Dichte	

Tabelle 34: Dichtefunktionen

igkeit nur ungefähr 15 Stellen exakt sind, ist das Ergebnis nur für $n \le 21$ exakt richtig. Sonst sind nur die ersten 15 Stellen exakt genau.

```
>> factorial(6)
ans =
720
```

Eine Alternative stellt die Funktion prod dar.

```
>> prod(1:6)
ans =
720
```

Mit der *Symbolic-Toolbox* hat man Zugriff auf Maple-Funktionen, zum Beispiel auf die Funktion factorial, die die Fakultäts-Funktion symbolisch berechnet, siehe mhelp factorial.

```
>> maple('factorial',6)
ans =
720
```

```
>> syms n
>> maple('factorial',n)
ans =
n!
```

oder einfach

```
>> maple('n!')
ans =
n!
```

```
>> maple('6!')
ans =
720
```

da Maple das !-Zeichen als Fakultätszeichen erkennt.

Die Funktion perms gibt alle Permutationen an. Von den drei verschiedenen Objekten 1, 2 und 3 gibt es genau sechs Permutationen. Diese sind:

```
>> perms(1:3)
ans =
      3
              2
                     1
      3
              1
                     2
      2
              3
                     1
      2
                     3
              1
      1
              2
                     3
      1
              3
                     2
```

Mit der Funktion nchoosek kann man Binomialzahlen berechnen.

Aufgabe 125 (Binomialzahlen) Wie viele Möglichkeiten gibt es aus n = 49 verschiedenen Objekten k = 6 verschiedene Objekte auszuwählen?

Lösung: Es ist gibt $\binom{n}{k}$ Möglichkeiten. Also ist

```
>> nchoosek(49,6)
ans =
13983816
```

Demnach beträgt die Wahrscheinlichkeit beim Lotto sechs richtige zu haben $1/13983816 \approx 7.1511 \cdot 10^{-8} = 0.000000071511. \odot \dots \odot$

Mit der Anweisung nchoosek(1:n,k) erhalten Sie alle Anordnungsmöglichkeiten für k Objekte ohne Wiederholung und ohne Berücksichtigung der Reihenfolge aus n verschiedenen Objekten. Zum Beispiel ist dies für n = 5 und k = 3:

```
>> nchoosek(1:5,3)
ans =
      1
              2
                      3
      1
              2
                      4
      1
              2
                      5
              3
      1
              3
      1
                      5
      1
              4
                      5
      2
              3
      2
              3
                      5
      2
              4
                      5
      3
                      5
```

Aufgabe 126 (Binomialzahlen) Geben Sie alle Teilmengen der Menge {1, 2, 3, 4, 5, 6} mit genau vier Elementen an. Wieviel gibt es?

Lösung: Es ist:

	0			
>> n	choose	k(1:6,	4)	
ans	=			
	1	2	3	4
	1	2	3	5
	1	2	3	6
	1	2	4	5
	1	2	4	6
	1	2	5	6
	1	3	4	5
	1	3	4	6
	1	3	5	6
	1	4	5	6
	2	3	4	5
	2	3	4	6
	2	3	5	6
	2	4	5	6
	3	4	5	6

Demnach gibt es

```
>> nchoosek(6,4)
ans =
15
```

Mit der Maple-Funktion multinomial kann man Polynomialzahlen berechnen.

Aufgabe 127 (Polynomialzahlen) Berechnen Sie mit Matlab $\binom{8}{2,3,3}$.

Lösung: Es ist

```
>> maple('with(combinat)');
>> maple('multinomial',8,2,3,3)
ans =
560
```

64.2. Permutationen ohne Wiederholung

Die Anzahl der Permutationen ohne Widerholung kann man mit den Funktionen factorial, prod, Maple-factorial und dem Maple-Zeichen! berechnen, siehe Abschnitt 64.1. Die Permutationen selbst sind mit der Funktion perms zu erhalten.

64.3. Variationen ohne Wiederholung

Die Anzahl der Variationen ohne Wiederholung kann mit den Funktionen zur Berechnung der Fakultäten und Binomialzahlen ausgerechnet werden, siehe Abschnitt 64.1.

Angenommen wir wollen V(30, 5) berechnen, so geht das zum Beispiel wie folgt:

```
>> factorial(30)/factorial(30-5)
```

```
ans = 17100720
```

oder aber

```
>> factorial(5)*nchoosek(30,5)
ans =
17100720
```

64.4. Kombinationen ohne Wiederholung

Siehe Abschnitt 64.1.

64.5. Permutationen mit Wiederholung

Aufgabe 128 (Permutationen m.W.) Berechnen Sie mit Matlab $P_W(2,3,3)$.

Lösung: Es ist $P_W(2,3,3) = 8!/(2! \cdot 3! \cdot 3!)$ auszurechnen.

```
>> maple('with(combinat)');
>> maple('multinomial',8,2,3,3)
ans =
560
```

64.6. Variationen mit Wiederholung

Aufgabe 129 (Variationen m.W.) Berechnen Sie mit Matlab $V_W(26, 6)$.

Lösung: Es ist $V_W(n,k) = n^k$.

```
>> VW = 26^6
VW =
308915776
```

64.7. Kombinationen mit Wiederholung

Aufgabe 130 (Variationen m.W.) Berechnen Sie mit MATLAB $K_W(6, 2)$.

Lösung: Es ist $K_W(n,k) = \binom{n+k-1}{k}$.

```
>> KW = nchoosek(6+2-1,2)
KW = 21
```

oder

```
>> KW = maple('binomial',6+2-1,2)
KW =
21
```

64.8. Weitere Funktionen

Verfügt man über die *Extended Symbolic Tool-box*, also über alle Maple-Funktionen, so gibt es das combinat-Paket. Damit stehen noch mehr Funktionen

bell	binomial	cartprod
character	Chi	choose
composition	conjpart	decodepart
encodepart	fibonacci	firstpart
graycode	inttovec	lastpart
multinomial	nextpart	numbcomb
numbcomp	numbpart	numbperm
partition	permute	powerset
prevpart	randcomb	randpart
randperm	setpartition	stirling1
stirling2	subsets	vectoint

zur Lösung kombinatorischer Probleme zur Vefügung, siehe mhelp combinat.

65. Zufallszahlen

Zufallszahlen sind im wissenschaftlichen Rechnen ein nützliches Hilfsmittel. In vielen Fällen werden Zufallszahlen in einer rechnergestützten Simulation eines komplexen Problems eingesetzt, zum Beispiel bei der Planung von Produktionssystemen oder von Großprojekten. Diese Simulation kann dann auf dem Rechner immer und immer wieder ausgeführt werden, und die Resultate können analysiert werden. Oft werden Zufallszahlen auch als Testdaten benutzt. Hat man zum Beispiel einen Algorithmus entwickelt, der ein beliebiges Gleichungssystem lösen soll, so kann man eine Matrix als auch eine rechte Seite des Systems zum Testen des Algorithmus mit Zufallszahlen generieren.

Zufallszahlen werden am Computer mit Hilfe spezieller Algorithmen berechnet. Solche Algorithmen nennt man Zufallsgeneratoren. Grundlegend ist dabei die Erzeugung von Zufallszahlen $x_1, x_2, ..., x_n$, deren Werte sich in sehr guter Näherung wie Realisierungen von unabhängigen auf [0,1] gleichverteilten Zufallsvariablen X_1, X_2, \ldots, X_n verhalten. Da die Werte x_1, x_2, \ldots, x_n tatsächlich jedoch berechnet werden, sind sie nicht echt zufällig. Man spricht deshalb auch von Pseudo-Zufallszahlen, die sich (fast) wie echte verhalten. Mit Hilfe von gleichverteilten Zufallszahlen lassen sich Zufallszahlen für andere Verteilungen durch geeignete Transformationen erzeugen. Je nach Verteilung kann dies sehr einfach oder aber auch kompliziert sein. In der Statistik Toolbox wird Ihnen diese Arbeit abgenommen, siehe Abschnitt 65.4.

In Matlab gibt es zwei eingebaute Funktionen rand und randn, mit denen man Zufallszahlen erzeugen kann. Wir wollen im folgenden diese beiden Funktionen vorstellen und miteinander vergleichen. Dabei werden uns ihre statistischen Eigenschaften durch grafische Darstellungen verdeutlichen.

65.1. Gleichverteilte Zufallszahlen

Zufallszahlen sind durch die Verteilung ihrer Werte charakterisiert. Zum Beispiel sind gleichverteilte Zufallszahlen dadurch gekennzeichnet, dass alle Werte der Zahlenfolge in einem bestimmten Intervall gleichverteilt liegen. So erzeugt die Funktion rand(10,1) einen Spaltenvektor mit 10 gleichmäßig über das Intervall]0,1[verteilten Zufallszahlen (Genaugenommen erzeugt rand eine Gleitpunktzahl im abgeschlossenen Intervall [eps/2,(1 – eps/2)]):

Der Aufruf rand (50,2) erzeugt eine Matrix mit 50 Zeilen und 2 Spalten mit Werten zwischen 0 und 1. Testen Sie dies! Allgemein

erzeugt der Aufruf rand(m,n) eine (m,n)-Matrix mit gleichverteilten Zufallszahlen zwischen 0 und 1.

In den Anwendungen werden oft auch Zufallszahlen gesucht, die in einem anderen Intervall als]0, 1[liegen. Mit der MATLAB-Funktion rand ist auch dies leicht möglich. Zum Beispiel erzeugt die Anweisung

```
5 + 3*rand(n,1)
```

einen Spaltenvektor mit *n* gleichverteilten Werten im Intervall [5, 8[.

Mit rand und der eingebauten Matlab-Funktion floor lassen sich auch leicht ganzzahlige Zufallszahllen generieren. Die Rundungsfunktion floor(A) rundet alle Elemente von A auf die nächstkleinere ganze Zahl (nächste ganze Zahl in Richtung −∞). Der folgende Function-File realisiert das Erzeugen von ganzzahligen Zufallszahlen. Außer der Größe der gewünschten Matrix lässt sich außerdem mit k ein Intervall [-k:k] angeben, aus dem die Zahlen zufällig erzeugt werden. Wird k nicht angegeben, so wird k=9 gewählt.

```
function A = randganz(m,n,k)
if nargin == 1, n=m; end;
if nargin < 3, k=9; end;
A = floor( (2*k+1)*rand(m,n)-k );</pre>
```

Aufgabe 131 (Zufallszahlen) Die Matlab-Funktion rand(n) erzeugt eine $n \times n$ -Matrix, deren Einträge gleichverteilte Zufallszahlen aus dem Intervall]0, 1[sind. Schreiben Sie einen Function-File, der $n \times n$ -Matrizen erzeugt, deren Einträge Zahlen aus $\{1, 2, 3, 4, 5, 6\}$ sind.

Lösung: Dies erreicht man zum Beispiel mit

folgender Funktion. Die Funktion ceil rundet zur nächsten ganzen Zahl auf (nächste ganze Zahl in Richtung $+\infty$).

```
function A = randganz1bisk(m,n,k)
f
```

```
©.....
```

65.2. Normalverteilte Zufallszahlen

Erzeugen wir Zufallszahlen mit der Funktion rand, so treten alle Werte in einem bestimmten Intervall gleichmäßig häufig auf. In den Anwendungen sucht man oft auch Zufallszahlen, deren Werte nicht gleichmäßig auftreten, sondern wo bestimmte Werte häufiger vorkommen als andere, man nennt sie normalverteilte Zufallszahlen (auch Gausssche Zufallszahlen genannt).

In Matlab erzeugt man normalverteilte Zufallszahlen mit Mittelwert 0 und Varianz 1 mit der Funktion randn. Der Befehl randn(10,1) erzeugt zum Beispiel folgende normalverteilte Zufallszahlen:

```
ans =
-0.4326
-1.6656
0.1253
0.2877
-1.1465
1.1909
1.1892
-0.0376
0.3273
0.1746
```

Will man eine normalverteilte *n*-wertige Zufallsfolge mit Mittelwert 10 und Standardabweichung 5 erzeugen, so erreicht man dies durch:

```
10+5*randn(n,1)
```

65.3. Im Vergleich: Gleich- und normalverteilte Zufallszahlen

Histogramme bieten eine geeignete Möglichkeit, um Zufallszahlen grafisch darzustellen. Der Script-File

```
subplot(2,1,1)
yg = rand(10000, 1);
hist(yg, 25)
h = findobj(gca,'Type','patch');
set(h,'FaceColor','b',...
'EdgeColor','w')
axis([-1 2 0 600])
title('Gleichmäßigverteilte
Zufallszahlen')
yn = randn(10000, 1);
x = min(yn):0.2:max(yn);
subplot(2,1,2)
hist(yn,x)
h = findobj(gca,'Type','patch');
set(h,'FaceColor','b',...
'EdgeColor','w')
title('Normalverteilte
Zufallszahlen')
```

illustriert die beiden Matlab-Funktionen rand und randn grafisch, siehe Abbildung 63. Das obere Histogramm zeigt, wie sich die

10000 Zufallszahlen gleichmäßig über das Intervall [0,1] verteilen. Mit dem Befehl rand(10000,1) wird ein Spaltenvektor mit 10000 Werten erzeugt und yg zugeordnet. Die Matlab-Funktion hist erstellt nun das Histogramm. Mit dieser Funktion kann man verschiedene Ziele erreichen. Ein Aufruf wie hist(yg,25) erzeugt ein Histogramm, das angibt, wie oft ein Wert in einem Subintervall von]0, 1[vorkommt. Die Zahl 25 schreibt vor, das Intervall]0, 1[in 25 gleichgroße Subintervalle einzuteilen und 25 Rechtecksflächen zu zeichnen, deren Breite die Subintervallänge und deren Höhe die Anzahl der Zufallswerte in dem jeweiligen Subintervall angeben. Das untere Histogramm in Abbildung 63 zeigt, wie

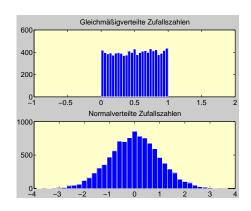


Abbildung 63: Zufallszahlen

sich 10000 Zufallszahlen mit Mittelwert 0 und Varianz 1 ähnlich einer Gaußschen Glockenkurve verteilen.

Der Skript-File

```
Punkte = rand(10000,2);
subplot(1,2,1)
plot(Punkte(:,1),Punkte(:,2),'.')
```

```
title('Gleichmäßigverteilte
Zufallszahlen')
axis([0 1 0 1])
axis('square')

Punkte = randn(10000,2);
subplot(1,2,2)
plot(Punkte(:,1),Punkte(:,2),'.')
title('Normalverteilte
Zufallszahlen')
axis([-3 3 -3 3])
axis('square')
```

erzeugt die Abbildung 64, um Unterschiede

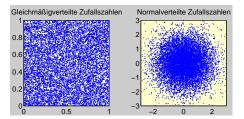


Abbildung 64: Zufallszahlen

zwischen rand und randn klar zu machen.

Aufgabe 132 (Zufallszahlen) Erzeugen Sie zehn gleichverteilte Zufallszahlen im jeweils angegebenen Intervall

- (a) zwischen 0 und 10.
- (b) zwischen -1 und 1.
- (c) zwischen -20 und 10.
- (d) zwischen $-\pi$ und π .

Lösung: Dies kann wie folgt erreicht werden.

- (a) 10*rand(10,1).
- (b) -1+2*rand(1,10).
- (c) -20+30*rand(1,10).

163

(d) -pi+2*pi*rand(1,10).

(Varianz) eins haben. \odot \odot

Aufgabe 133 (Zufallszahlen) Erzeugen Sie tausend gleichverteilte Zufallszahlen

- (a) mit Mittelwert 1/2 und Varianz 1/12
- (b) mit Mittelwert 0 und Varianz 1/12
- (c) mit Mittelwert 0 und Varianz 1
- (d) mit Mittelwert 0 und Varianz 3

Lösung: Dies kann man wie folgt erreichen:

- (a) rand(1000,1)
- (b) rand(1000,1)-0.5
- (c) sqrt(12)*(rand(1000,1)-0.5)
- (d) sqrt(3*12)*(rand(1000,1)-0.5)

Aufgabe 134 (Zufallszahlen) Erzeugen Sie tausend normalverteilte Zufallszahlen

- (a) mit Mittelwert 1.0 und Varianz 0.5.
- (b) mit Mittelwert -5.5 und Standardabweichung 0.25.
- (c) mit Mittelwert -5.5 und Standardabweichung 1.25.

Lösung: Dies kann wie folgt erreicht werden.

- (a) sqrt(0.5)*randn(1000,1)+1;.
- (b) 0.25*randn(1000,1)-5.5;.
- (c) 1.25*randn(1000,1)-5.5;.

Dies gilt wegen den Transformationsformeln für den arithmetischen Mittelwert bzw. für die Varianz: Ist $y_i = ax_i + b$, so gilt $\bar{y} = a\bar{x} + b$ und $s_Y^2 = a^2 s_X^2$. Hier gilt, dass die Zahlen x_i den Mittelwert null und die Standardabweichung

65.4. Andere Verteilungen

Wenn Sie über die *Statistik Toolbox* verfügen, so können Sie Zufallszahlen konfortabel und interaktiv mit randtool erzeugen; für ingesamt 20 verschiedene Verteilungen zum Beispiel für die Exponential- oder Binomialverteilungen. Die Abbildung 65 zeigt eine Exponen-

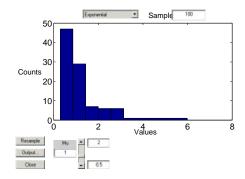


Abbildung 65: Exponentialverteilung

tialverteilung mit Erwartungwert 1.

66. Stochastische Simulationen

Eine stochastische Simulation liegt vor, wenn es um die Lösung von Problemen geht, bei denen Zufallseinflüsse auftreten. Stochastische Simulationsmethoden werden auch Monte Carlo Methoden genannt. Solche Methoden sind nützlich bei:

Nichtdeterministischen (stochastischen)
 Prozessen.

- Deterministischen Systemen, die zu kompliziert sind, um sie analytisch zu behandeln.
- Deterministischen Problemen, die so hoch dimensional sind, dass Standarddiskretisierungsverfahren ungeignet sind (zum Beispiel Monte Carlo Integrationen).

Die beiden grundsätzlichen Voraussetzungen stochastischer Simulationsmethoden sind:

- Kenntnis über relevante Wahrscheinlichkeitsverteilungen.
- Unterstützung bei der Erzeugung von Zufallszahlen.

Mit gleichverteilten Zufallszahlen können wir zum Beispiel Spiele mit dem Glücksrad am Rechner simulieren. Wenn wir etwa die relative Häufigkeit von Werten berechnen, die im Intervall [0.2, 0.6] liegen, und mit der Wahrscheinlichkeit $P(0.2 \le X \le 0.6) = 0.4$ vergleichen, dann sollte für größeres n die relative Häufigkeit nach dem Gesetz großer Zahlen mit hoher Wahrscheinlichkeit bei 0.4 liegen. Auch sollte die empirische Verteilungsfunktion der gezogenen Zahlen x_1, x_2, \ldots, x_n sich der [0, 1]-Gleichverteilung annähern.

Wir "spielen" nun Glücksrad am Computer, indem wir wiederholt auf [0, 1] gleichverteilter Zufallszahlen ziehen. Das Intervall [0, 1] wird wie in Abbildung 66 in zehn Teilintervalle der Länge 0.1 zerlegt und zu jedem "Spiel" wird festgelegt, in welches Teilintzervall die gezogene Zufallszahl fällt. Abbildung 66 zeigt links das resultierende Histogramm mit den relativen Häufigkeiten für die zehn Teilklassen nach n=100 Spielen. Man sieht, dass die relativen Häufigkeiten zum Teil noch deutlich von dem zu erwartenden Wert 0.10 abweichen. Für

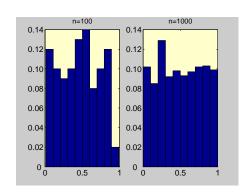


Abbildung 66: Empirische Verteilungen

n=1000 haben sich diese Häufigkeiten bereits wesentlich besser an den Wert 0.10 stabilisiert. Anders ausgedrückt: Die empirische Verteilung der gezogenen Zahlen approximiert die wahre Gleichverteilung besser.

Aufgabe 135 (Stochastische S.) Schreiben Sie eine Simulation, die das Volumen der Einheitskugel $\{x \in \mathbb{R}^4 \mid |x|^2 \le 1\}$ in \mathbb{R}^4 schätzt. *Lösung:* Sei

$$B_n = \{ \boldsymbol{x} \in \mathbb{R}^n \mid |\boldsymbol{x}| \le 1 \}$$

und sei

$$Q_n = [-1, 1]^n$$

= $\{ \mathbf{x} \in \mathbb{R}^n \mid |x_i| \le 1, \ i = 1, ..., n \}$

Dann ist $B_n \subset Q_n$. rand(n,1)*2-1 erzeugt uniform verteilte Zufallsvariablen in Q_n . Die Wahrscheinlichkeit, dass einer dieser Vektoren in B_n ist, ist $\frac{Vol(B_n)}{Vol(Q_n)}$ mit $Vol(Q_n) = 2^n$. Betrachtet man also z Zufallsvariablen, die durch rand(n,1)*2-1 erzeugt werden und definiert die Variable Inside als die Anzahl derjenigen Vektoren, die in B_n liegen, so liefert $Vol(Q_n)$.

 $\frac{Inside}{z}$ eine Schätzung für $Vol(B_n)$. Man nennt nen mindestens das Quadrat getroffen wurde. Schreibt man die relativen Häufigkeiten der

Somit löst das folgende Matlab Script-File die gestellte Aufgabe:

```
n=4;
Inside=0;
VolBn=zeros(500,1);
for k=1:500
    A=-1+2*rand(n,100);
    Inside=Inside+sum(sum(A.^2)<=1);
    VolBn(k)=Inside/(k*100)*2^n;
end
plot(VolBn)
VolBn(500)
```

66.1. Näherung für π

Mit stochastischen Simulationen ist es möglich, Antworten auch auf nicht-stochastische Fragestellungen zu geben. Mit einer stochastischen Simulation werden wir nun eine Näherung für die Kreiszahl $\pi=3.14159\ldots$ berechnen.

Wir denken uns zunächst folgendes Experiment, wozu man eine Zielscheibe und Dartpfeile benötigt. Die Zielscheibe besteht aus einem Quadrat und einem einbeschriebenen Kreis. Dann werfen wir nach dem Zufallsprinzip mit Dartpfeilen auf diese Scheibe und bestimmen die relative Häufigkeit der Kreistreffer, das heißt

$$h = \frac{\text{Anzahl der Treffer}}{\text{Anzahl der Würfe}}.$$

Hierbei werden nur Würfe gezählt, bei de-

nen mindestens das Quadrat getroffen wurde. Schreibt man die relativen Häufigkeiten der Kreistreffer auf, so stellt man fest, dass sie sich mit zunehmender Wurfzahl dem Wert $\pi/4$ nähert, denn die relative Häufigkeit h der Kreistreffer wird mit zunehmender Wurfzahl etwa dem Verhältnis (Flächeninhalt des Kreises) zu (Flächeninhalt des Quadrates) entsprechen. Ist also r der Kreisradius, so gilt: $h \approx \frac{r^2\pi}{4r^2} = \frac{\pi}{4}$ und somit

$$\pi \approx 4h$$
.

Wir führen dieses Experiment nicht real aus (selbstverständlich können Sie dies tun, wenn Sie wollen), sondern simulieren es auf dem Computer. Wie ist das möglich? Die Wahrscheinlichkeit eines zufälligen Ereignisses ist die idealisierte relative Häufigkeit, mit welcher dieses vorkommt, wenn das zugehörige Zufallsexperiment sehr oft wiederholt wird. Der folgende Script-File realisiert die Simulation.

```
plot(PiNaeherung)
title(sprintf('Näherung von Pi =
%5.3f',PiNaeherung(end)));
xlabel('mal Hundert')
```

Die Abbildung 67 zeigt das Ergebnis der Si-

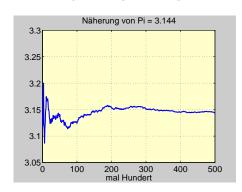


Abbildung 67: Näherung für π

mulation. Siehe auch Aufgabe 184.

66.2. Zum Ziegenproblem

In einer Spielshow ist als Hauptpreis ein Auto ausgesetzt. Hierzu sind auf der Bühne drei verschlossene Türen aufgebaut. Hinter einer rein zufällig ausgewählten Tür befindet sich der Hauptpreis, hinter den beiden anderen jeweils eine Ziege. Der Kandidat wählt eine der Türen, beispielsweise Tür 1 aus; diese bleibt aber vorerst verschlossen. Der Spielleiter, der weiß, hinter welcher Tür das Auto steht, öffnet daraufhin mit den Worten: "Soll ich Ihnen mal etwas zeigen?"eine der beiden anderen Türen, zum Beispiel Tür 3, und eine Ziege schaut ins Publikum. Der Kandidat hat nun die Möglichkeit, bei seiner ursprünglichen Wahl zu bleiben oder die andere verschlossene Tür (in unserem

Beispiel Nr.2) zu wählen. Er erhält dann den Preis hinter der von ihm zuletzt gewählten Tür. In dem folgenden Skript finden Sie die Antwort auf diese Frage.

```
% Script-File: Ziegen
%
% Simulation des Ziegenproblems.
% Es werden zehn Versuche mit
% jeweils zehntausend
% Durchgängen simuliert.
%Zehn Versuche.
for i=1:10
    %Zehntausend Durchgänge.
    %Zufall wird vorbereitet.
    %Es wird eine (10000,3)-Zu-
    %fallsmatrix mit den Werten
    % 1,2 und 3 erzeugt.
    a = ceil(3*rand(10000,3));
    %Zufallswahl der Autotür A.
    A = a(:,1);
    %Zufällige Erstwahl W1.
    W1 = a(:,2);
    %Moderator will Tür M öffnen.
    M = a(:,3);
    %M darf keine Autotür (A) sein
    %und auch nicht die gewählte
    %Tür (W1).
    for j=1:10000
        while ( (M(j)==A(j)) |...
                (M(j)==W1(j))
        M(j) = ceil(3*rand(1,1));
        end
    end
    %W2 wäre die Tür, zu der nun
    %gewechselt werden kann. Es
```

```
%muss sein: M+W1+W2=6.
    W2 = 6-M-W1;
    %Wenn W2 die Autotür ist, ein
    %Punkt für 'Wechseln ist
    %richtig'. Sonst ein Punkt für
    %'Wechseln ist falsch', da W1
    %gleich A ist. Die Variablen R
    %und F zählen das richtige bzw.
    %falsche Wechseln.
    R = sum(W2==A);
    F = 10000-R;
    sprintf('Wechseln ist richtig:
    %3.0f\t Wechseln ist falsch:
    %3.0f',R,F)
    subplot(2,5,i); pie([R,F]);
end
%Grafische Ausgabe als
%Kreisdiagramm.
legend('Wechseln ist richtig',
       'Wechseln ist falsch');
```

66.3. Das Geburtstagsparadox

Stellen wir uns vor, dass sich in einem Raum eine gewisse Anzahl von Personen befindet. Wir interessieren uns dafür, ob zwei am gleichen Tag Geburtstag haben. Wie viele Personen müssen vorhanden sein, damit garantiert zwei am gleichen Tag Geburtstag haben? Das ist eine einfache Frage: Da das Jahr 365 Tage hat, haben von 366 Personen bestimmt zwei am gleichen Tag Geburtstag (Wir gehen auf Schaltjahre nicht ein). Eine wesentlich interessantere Frage ist: Wie viele Personen müssen vorhanden sein, damit es wahrscheinlich ist, dass zwei am gleichen Tag Geburtstag haben? Genauer: Wie viele Personen braucht

man, dass mit einer Wahrscheinlichkeit von über 50% zwei am gleichen Tag Geburtstag haben? Die Antwort ist überraschend: Bereits 23 Personen reichen aus, um die Wahrscheinlichkeit für einen gleichen Geburtstag auf über 50% zu bringen. Achtung: Es ist weder danach gefragt, dass zwei am 1. Januar Geburtstag haben oder dass jemand am gleichen Tag wie Sie Geburtstag hat, sondern danach, ob zwei Leute am 1. Januar oder am 2. Januar oder usw. Geburtstag haben.

In einem Raum sind *n* Personen versammelt. Unter der Voraussetzung, dass jedes Jahr 365 Tage hat (es gibt keine Schaltjahre) und dass die Geburtstage übers Jahr gleichverteilt sind, ist die Wahrscheinlichkeit dafür, dass mindestens zwei Personen am gleichen Tag Geburtstag haben gegeben durch die Formel:

$$P = 1 - \frac{364 \cdot 363 \cdots (365 - n + 1)}{365^{n-1}}$$

Für n = 23 ergibt sich 0.5073 = 50.73% und für n = 50 bereits 0.9704 = 97.04%.

Diese Resultate der Wahrscheinlichkeitsrechung lassen sich simulationsmäßig bestätigen. Dazu dient der folgende Script-File.

```
r = 0:
for k=1:kmax
    %Zufallszahlen zwischen 1
    %und 365.
    g = ceil(365*rand(n,1));
    for i=1:n
        a = g-g(i)*ones(n,1);
        if( sum(\sim a)>=2 )
            %Zwei Zahlen sind
            %gleich.
            r = r+1;
            break
        end
    end
end
%Wahrscheinlichkeit.
p = r/kmax;
[n,p]
end
```

Die Ergebnisse der theoretischen Berechnungen und der Simulation sind in Tabelle 35 gegenübergestellt.

66.4. Bestimmte Integrale

Das bestimmte Integral

$$\int_0^1 x^x dx$$

kann von Matlab nicht exakt (symbolisch) berechnet werden, siehe Abschnitt 61. Wir können aber mit Hilfe der Näherung

$$\int_0^1 f(x)dx \approx \frac{1}{n} \sum_{i=1}^n f(x_i)$$

einen geschätzten Wert erhalten. Hierbei sind x_i gleichmäßig verteilte Zufallszahlen aus dem

n	Theorie	Simulation
5	0.0271	0.0248
10	0.1169	0.1118
15	0.2529	0.2526
20	0.4114	0.4216
22	0.4757	0.4846
23	0.5073	0.5156
25	0.5687	0.5642
30	0.7063	0.6980
35	0.8144	0.8064
40	0.8912	0.8910
45	0.9410	0.9424
50	0.9410	0.9700
55	0.9863	0.9880

Tabelle 35: Zum Geburtstagsparadox

Intervall]0, 1[.

```
>> n = 1000;

>> x = rand(n,1);

>> y = x.^x;

>> sum(y)/n

ans =

0.7850
```

Somit ist also

$$\int_0^1 x^x dx \approx 0.7850$$

67. Symbolisches Rechnen

Symbolisches Rechnen ist der Versuch, die Methoden, die man beim Rechnen mit Papier und Bleistift kennt, auf Computern abzubilden und dort schließlich durchzuführen.

Um mit Matlab symbolisch rechnen zu kön-



nen, muss die *Symbolic Math Toolbox* installiert sein.

Symbolische Rechnungen basieren auf Variablen, denen nicht unbedingt Zahlen zugewiesen sind. Man rechnet mit Symbolen und Termen, wie man es vom Rechnen mit Papier und Bleistift kennt. Arithmetische Operationen können exakt durchgeführt werden. Außerdem kann man Näherungen bis auf eine beliebig vorgegebene gewünschte Anzahl von Stellen finden. Man kann Polynome oder rationale Ausdrücke symbolisch addieren, subtrahieren und dividieren. Ausdrücke können differenziert werden und man erhält die gleichen Ergebnisse, die bisher nur mit Bleistift und Papier zu erzielen waren. Ausdrücke können sogar unbestimmt integriert werden, sofern sie Integrale in geschlossener Form besitzen. Diese Möglichkeiten erleichtern das ermüdende und fehlerbedrohte Manipulieren komplizierter Ausdrücke, das auch häufig das Vorspiel numerischer Behandlung bildet. Mit symbolischem Rechnen lassen sich auch kleinere lineare Gleichungssysteme ohne Rundungsfehler lösen. Auf jeden Fall ist symbolisches Rechnen ein sich ständig entwickelndes Gebiet, dessen Bedeutung für das wissenschaftliche Rechnen zunehmen wird.

67.1. Erste Schritte

Durch

```
>> syms x t
```

werden die beiden symbolischen Variablen x und t erzeugt, mit denen man durch

den Ausdruck $3x^2 + xt - 1$ kreiert.

Bekanntlich ist

$$\frac{a^2 - b^2}{a - b} = a + b.$$

```
>> syms a b
>> simplify((a^2-b^2)/(a-b))
ans =
a+b
```

Es ist

$$(a+b+c)^2 = a^2 + b^2 + c^2 + 2(ab + ac + bc)$$

```
>> syms a b c
>> expand((a+b+c)^2)
ans =
a^2+2*a*b+2*a*c+b^2+2*b*c+c^2
```

Mit dem pretty-Befehl kann man Formeln in etwas lesbarerer Form ausgeben.

Aufgabe 136 (expand-Funktion) Die 6. Fibo-NACCI-Zahl ist 8. Diese kann man durch

$$\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^6 - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^6$$

berechnen. Bestätigen Sie dies.

Lösung: Es ist

```
a = expand(((1+sqrt(sym(5)))/2)^6);
b = expand(((1-sqrt(sym(5)))/2)^6);
simplify(1/sqrt(5)*a-1/sqrt(5)*b)
ans =
8
```

Mit der Funktion simplify können symbolische Terme vereinfacht werden. In der Mathematik lernt man die Gültigkeit von $\sin(x)^2 + \cos(x)^2 = 1$. Mit der Funktion simplify kann man dies bestätigen:

```
>> syms x
>> simplify( sin(x)^2+cos(x)^2 )
ans =
1
```

Substitutionen können mit der Funktion subs durchgeführt werden. Die folgenden Anweisungen substituieren cos(x) anstelle von x.

```
>> syms x
>> subs( sqrt(1-x^2),x,cos(x) )
ans =
(1-cos(x)^2)^(1/2)
```

Matlab verfügt über viele eingebaute mathematische Funktionen mit denen man symbolisch rechnen kann. Mit diesen vordefinierten Funktionen ist es uns dann möglich, weitere Funktionen zu konstruieren, indem man die entsprechenden Funktionsterme f(x) miteinander verküpft.

Funktionsterme sind Ausdrücke, das heißt, sie können wie gewöhnliche Terme erzeugt wer-

den. Angenommen wir wollen mit dem Funktionsterm $f(x) = 2x^2 + 3x + 4$ arbeiten, zum Beispiel differenzieren oder integrieren, so definiert man diesen wie folgt.

```
>> syms x
>> f = 2*x^2+3*x+4;
```

Danach kann man zum Beispiel f'(x) = 4x + 3 bilden.

```
>> diff(f)
ans =
4*x+3
```

Aufgabe 137 (Rangordnung) Berechnen Sie symbolisch:

- (a) $2^10/10$
- (b) 2+3*4
- (c) 1+2/3*4.

Lösung: Es ist

```
a) >> sym(2^10/10)
    ans =
    512/5

b) >> sym(2+3*4)
    ans =
    14

c) >> sym(1+2/3*4)
    ans =
    11/3
```

Aufgabe 138 (Potenzregeln) Bestätigen Sie die folgenden Rechenregeln:

```
(a) a^m \cdot a^n = a^{m+n}
```

(b)
$$\frac{a^m}{a^n} = a^{m-n}$$

Verwenden Sie die Matlab-Funktion simplify.

Lösung: » syms a n m

Aufgabe 139 (Rechnen) Berechnen Sie den Ausdruck

$$\frac{\sqrt{16} + \cos(\pi/3)}{\sqrt[3]{8} + 4}$$

symbolisch und numerisch.

Lösung: Symbolische Rechnung:

Sowohl

also auch

liefert

Numerische Rechnung:

Aufgabe 140 (Vereinfachen) Vereinfachen Sie den Ausdruck

$$\frac{x^2 + 2xy + y^2}{x^2 - y^2}$$

Lösung: Mit

erhält man

Somit ist die Vereinfachung dann also

$$\frac{x^2 + 2xy + y^2}{x^2 - y^2} = \frac{x - y}{x + y} .$$

Anmerkung: Man kann statt simplify auch den Befehl simple verwenden. ©

Aufgabe 141 (Multiplikation) Multiplizieren Sie

$$(x^2 + x + 1) \cdot (x^3 - x^2 + 1).$$

Lösung: Mit

sieht man, dass

$$(x^2 + x + 1) \cdot (x^3 - x^2 + 1) = x^5 + x + 1$$

ist. \odot \odot

Aufgabe 142 (Potenzieren) Berechnen Sie

$$(1+x)^4$$
.

Lösung: Mit

```
>> syms x
>> expand((1+x)^4)
ans =
1+4*x+6*x^2+4*x^3+x^4
```

sieht man, dass

$$(1+x)^4 = x^4 + 4x^3 + 6x^2 + 4x + 1$$

Aufgabe 143 (Faktorisieren) Faktorisieren Sie das Polynom

$$x^6 + x^4 - x^2 - 1$$
.

Lösung: Mit

sieht man, dass

$$(x^6 + x^4 - x^2 - 1) = (x - 1) \cdot (x + 1) \cdot (x^2 + 1)^2$$

ist ©

67.2. Wie man Maple-Funktionen verwendet

Die Funktion maple erlaubt es, jede Maple-Funktion direkt zu verwenden. Kennt man den Namen der gewünschten Maple-Funktion, so kann man mit

```
mhelp <Name>
```

Maple-Online-Informationen erhalten. Mit dem Aufruf

```
>> mhelp index[function]
```

erhält man eine Liste aller Maple-Funktionen. Als Beispiel betrachten wir den Befehl

```
>> mhelp gcd
```

Da man auf diese Art die Syntax der Maple-Funktion gcd erfahren hat, kann man nun die Funktion maple benutzen, um gcd zu verwenden. Die folgenden Zeilen zeigen, wie man den größten gemeinsamen Teiler der Polynome $x^2 - y^2$ und $x^3 - y^3$ berechnet:

```
>> syms x y
p1 = x^2-y^2;
p2 = x^3-y^3;
maple('gcd',p1,p2)
ans =
-y+x
```

Die Anweisungen

```
>> maple('gcd',225,725)
ans =
25
```

bestimmen den größten gemeinsamen Teiler von 225 und 725.

67.3. Mathematische Funktionen

Matlab verfügt über viele eingebaute Mathematische Funktionen mit denen man symbolisch rechnen kann. Mit diesen vordefinierten

Funktionen ist es uns dann auch möglich, weitere Funktionen zu konstruieren, indem man die entsprechenden Funktionsterme f(x) miteinander verküpft.

Funktionsterme sind Ausdrücke, das heißt, sie können wie gewöhnliche Terme erzeugt werden. Angenommen wir wollen mit dem Funktionsterm $f(x) = 2x^2 + 3x + 4$ arbeiten, zum Beispiel differenzieren oder integrieren, so definiert man diesen wie folgt.

```
>> syms x
>> f = 2*x^2+3*x+4;
```

Danach kann man zum Beispiel f'(x) = 4x + 3 bilden.

```
>> diff(f)
ans =
4*x+3
```

Will man einen abstrakten Funktionsterm f(x) erzeugen, das heißt einen Term, der irgendein spezieller Funktionsterm sein kann, so ist das durch die Anweisung

```
>> f = sym('f(x)')
```

möglich. Hierdurch wird das symbolische Objekt f erzeugt, das wie f(x) agiert, das heißt, dem Objekt f ist bekannt, dass eine unabhängige Variable x existiert. Das dem so ist, zeigen die folgenden Zeilen.

```
>> diff(f)
ans =
diff(f(x),x)
>> diff(f,'t')
ans =
0
```

Der Funktionsterm f(x) nach x abgeleitet, ergibt f'(x) bzw. in Matlab-Terminologie diff(f(x),x); nach t abgeleitet ist er aber 0, denn f(x) hängt nicht von t ab.

Damit können wir nun den Differenzenquotient der Funktion f im Punkt \bar{x} bilden

```
>> syms xbar h
>> ( subs(f,xbar+h)-subs(f,xbar) )/h
ans =
(f(xbar+h)-f(xbar))/h
```

und dann auch die Ableitung (Differenzialquotient) $f'(\bar{x})$ im Punkt \bar{x} .

```
>> limit(ans,h,0)
ans =
D(f)(xbar)
```

Als weiteres Beispiel bestätigen wir die Produktregel.

```
>> diff(f*g)
ans =
diff(f(x),x)*g(x)+f(x)*diff(g(x),x)
```

Aufgabe 144 (Funktionsterme) Bestätigen Sie die Quotientenregel

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{2g(x)}$$

Lösung: Hier die Bestätigung:

```
f = sym('f(x)'); g = sym('g(x)');
diff(f/g)
ans =
diff(f(x),x)/g(x)-f(x)/g(x)^2*
diff(g(x),x)
```

Häufig hat man einen symbolischen Funktionsterm definiert und möchte dann symbolische oder numerische Funktionswerte berechnen. Als Beispiel betrachten wir den symbolischen Funktionsterm $f(x) = \sqrt{x}$. Es soll f(2) zunächst symbolisch ausgewertet werden; hierbei hilft die subs-Funktion.

```
>> syms x
>> f = sqrt(x);
>> pretty(subs(f,sym(2)))

1/2
2
```

Auch die numerische Funktionsberechnung kann mit der subs-Funktion erfolgen.

```
>> subs(f,2)
ans =
1.4142
```

Wir können f(x) aber auch für mehrere x-Werte auswerten. Numerisch:

```
>> subs(f,[1, 2, 3, 4])
ans =
1.0000 1.4142 1.7321 2.0000
```

und symbolisch:

```
>> subs(f,sym([1, 2, 3, 4]))
ans =
[ 1, 2^(1/2), 3^(1/2), 4^(1/2)]
```

Mit zwei gegebenen Funktionen f und g und den Operationen Addition, Subtraktion, Multiplikation und Division kann man die Summe f+g, die Differenz f-g, das Produkt fg und den Quotienten f/g bilden. Zusätzlich kann man die Komposition von f und g, in Zei-

chen $f \circ g$, definieren.

Die Sinusfunktion $f=\sin$ und die Exponentialfunktion $g=\exp$ sind eingebaute Funktionen. Wir können damit die Summenfunktion $f+g=\sin+\exp$ oder die Verkettung $f\circ g=e^{\sin}$ dieser beiden Funktionen bilden. Die folgenden Anweisungen zeigen die Umsetzung:

```
>> syms x
>> sin(x) + exp(x)
ans =
sin(x)+exp(x)
>> exp(sin(x))
ans =
exp(sin(x))
```

oder alternativ mit der Funktion compose für die Komposition.

```
>> syms x
>> compose(exp(x),sin(x))
ans =
exp(sin(x))
```

Wir können auch (symbolische) Funktionswerte berechnen.

```
>> sin(sym(1.5))
ans =
sin(3/2)
```

Den dazugehörigen numerischen Werte erhalten wir durch die Anweisung

```
>> double(ans)
ans =
0.9975
```

```
>> sin(sym(pi/4))
ans =
```

```
1/2*2^(1/2)
```

Wir können auch die Funktion subs verwenden, um Funktionswerte zu berechnen.

```
>> syms x
>> exp(sin(x));
>> subs(ans,sym(2))
ans =
exp(sin(2))
```

Der numerische Wert ist

```
>> double(ans)
ans =
2.4826
```

oder direkt

```
>> exp(sin(x));
>> subs(ans,2)
ans =
2.4826
```

Mit dem Befehl type ist die Überprüfung möglich, ob eine vorliegende Funktion f gerade oder ungerade ist. Eine reellwertige Funktion f mit symmetrischem Definitionsbereich $D_f \subseteq \mathbb{R}$ wird als gerade bezeichnet, wenn f(x) = f(-x) für alle $x \in D_f$ gilt. Demnach ist die Kosinusfunktion cos eine gerade Funktion. Wir überprüfen diese Behauptung.

```
>> syms x
>> f = cos(x);
>> maple('type',f,'evenfunc(x)')
ans =
true
```

Die Exponentialfunktion exp ist dagegen weder gerade noch ungerade.

```
>> g = exp(x);
>> maple('type',g,'evenfunc(x)')
ans =
false
>> maple('type',g,'oddfunc(x)')
ans =
false
```

Mit der Funktion finverse kann man von der Funktion f die Umkehrfunktion f^{-1} finden, genauer: man kann den Funktionsterm von f^{-1} berechnen. Das folgende Beispiel zeigt dies für die Exponentialfunktion $f(x) = e^x$, $x \in \mathbb{R}$.

```
>> syms x
>> f = exp(x);
>> finverse(f)
ans =
log(x)
```

Die Umkehrfunktion ist somit $f^{-1}(x) = \ln(x)$, x > 0, wobei der Definitionsbereich von Mar-LAB nicht angegeben wird, sondern in Eigenverantwortung hingeschrieben werden muss.

Ist bei der Umkehrfunktion besonders auf den Definitionsbereich zu achten, so gibt MATLAB eine Warnung heraus, wie folgendes Beispiel zeigt:

```
>> g = x^2;
>> finverse(g)
Warning: finverse(x^2) is not
unique.
> In C:\MATLAB6p1\toolbox\symbolic\
@sym\finverse.m at line 43
ans =
x^(1/2)
```

Die Umkehrfunktion von der quadratischen Funktion $f(x) = x^2$, $x \in \mathbb{R}$ ist die Wurzelfunktion, der Definitionsbereich dieser ist aber nicht ganz \mathbb{R} , sondern $\mathbb{R}_{\geq 0}$.

Der Stetigkeitsbegriff ist ein zentraler Begriff der Analysis bzw. der Differenzial- und Integralrechung. Die Maple-Funktion iscount überprüft, ob eine vorliegende Funktion f auf einem anzugebendem Intervall stetig ist. Die Funktion f(x) = 1/x ist auf [1,3] stetig, nicht aber auf [-1,3]. Hier die Bestätigung:

```
>> syms x
>> maple('iscont',1/x,'x=1..3')
ans =
true
>> maple('iscont',1/x,'x=-1..3')
ans =
false
```

Stückweise definierte Funktionen lassen sich mit der Funktion piecewise definieren. Hier ein Beispiel.

67.4. Algebraische Gleichungen

Mit der Funktion solve kann man Gleichungen lösen. Das folgende Beispiel löst $x^2 - x = 0$.

```
>> solve('x^2-x = 0')
ans =
[0]
[1]
```

Findet man keine exakte symbolische Lösung, so wird die Lösung in variabler Genauigkeit ausgegeben.

```
>> solve('cos(x) = x')
ans =
.739...87
```

Aufgabe 145 (Algebraische Gl.) Lösen Sie die algebraischen Gleichungen $\cos x = e^x$ und $e^x + 2e^{-x} - 9 = 0$.

Lösung: Es ist

```
>> solve('cos(x)=exp(x)')
ans =
0
>> solve('exp(x)+2*exp(-x)-9=0')
ans =
[ log(9/2+1/2*73^(1/2))]
[ log(9/2-1/2*73^(1/2))]
>> double(ans)
ans =
2.1716
-1.4784
```

Die Funktion fzero erlaubt es uns alternativ die Nullstellen numerisch zu berechnen. Mit dem Starwert $x_0 = 1$ (zweites Argument in fzero) erhalten wir die Lösung

und mit dem Starwert $x_0 = -1$ die zweite Lösung

Aufgabe 146 (Algebraische Gl.) Berechnen

Sie die Lösung der Gleichung $xe^x = 1$.

Lösung:

Aufgabe 147 (Algebraische Gl.) Lösen Sie das Gleichungssystem

$$3x^2 + 2y = 0$$
$$2x - 3y^2 = 0$$

Lösung:

67.5. Grenzwerte

Die Folge

$$a_n = \frac{2n+1}{4n}, \qquad n \ge 1$$

hat den Grenzwert 1/2. Der folgende MATLAB-Code bestätigt dies:

Die Eulersche Zahl e = 2.7183... ist der Grenzwert der Folge mit der Bildungsvorschrift $(1 + 1/n)^n$, $n \ge 1$ für $n \to \infty$. Die folgenden Anweisungen zeigen dies:

```
>> syms n
>> limit((1+1/n)^n,inf)
ans =
exp(1)
```

Mit der Funktion limit lassen sich auch Grenzwerte von Funktionen berechnen. Die Tabelle 36 Möglichkeiten mit limit.

Mathematik	Matlab
$\lim_{x\to 0} f(x)$	limit(f)
$\lim_{x \to a} f(x)$	<pre>limit(f,x,a)</pre>
x ru	oder limit(f,a)
$\lim_{\substack{x \to a \\ x < 0}} f(x)$	<pre>limit(f,x,a,'left')</pre>
$\lim_{\substack{x \to a \\ x > 0}} f(x)$	<pre>limit(f,x,a,'right')</pre>

Tabelle 36: Grenzwerte berechnen

Besitzt der Differenzenquotient $(f(x) - f(\bar{x}))/(x - \bar{x})$ der Funktion f für $x \to \bar{x}$ einen Grenzwert, so heißt f an der Stelle \bar{x} differenzierbar. Der Grenzwert wird mit $f'(\bar{x})$ bezeichnet und man schreibt

$$f'(\bar{x}) = \lim_{x \to \bar{x}} \frac{f(x) - f(\bar{x})}{x - \bar{x}}.$$

Man nennt den Grenzwert auch Differenzialquotient und die Berechnungsmethode von $f'(\bar{x})$ die x-Methode.

Wir können limit einsetzen, um die Ableitung einer Funktion f an der Stelle \bar{x} zu berechnen. Als Beispiel berechnen wir für die Funktion f(x) = 3/x, x > 0 die Ableitung f'(2).

Somit ist f'(2) = -3/4 oder

$$\lim_{x \to 2} \frac{3/x - 3/2}{x - 2} = -\frac{3}{4}$$

Anstatt die Ableitung von f an der Stelle \bar{x} mit Hilfe der x-Methode zu berechnen, ist es manchmal zweckmäßiger die sogenannte h-Methode einzusetzen. In diesem Fall wird $f'(\bar{x})$ wie folgt berechnet:

$$f'(\bar{x}) = \lim_{h \to 0} \frac{f(\bar{x} + h) - f(\bar{x})}{h}.$$

Für obiges Besipiel mit f(x) = 3/x, x > 0 und $\bar{x} = 2$ erhält man:

```
>> syms h
>> limit( (3/(2+h)-3/2)/h, h, 0)
ans =
-3/4
```

Bekanntlich hat der Differenzenquotient $(f(x) - f(\bar{x}))/(x - \bar{x})$ für die Betragsfunktion $f(x) = abs(x) = |x|, x \in \mathbb{R}$ und $\bar{x} = 0$ keinen Grenzwert. In Matlab erhalten wir:

```
>> syms x
>> limit((abs(x)-abs(0))/(x-0),x,0)
ans =
NaN
```

Zwar ist die Betragsfunktion abs im Punkt 0 nicht differenzierbar, hat aber dort einen rechten und linken Grenzwert, das heißt es gilt

$$\lim_{\substack{x \to 0 \\ x > 0}} \frac{abs(x) - abs(0)}{x - 0} = 1$$

und

$$\lim_{\substack{x \to 0 \\ x < 0}} \frac{abs(x) - abs(0)}{x - 0} = -1$$

Wir bestätigen dies:

```
>> limit( (abs(x)-abs(0))/(x-0),...
x,0,'right')
ans =
1
\Rightarrow limit( (abs(x)-abs(0))/(x-0),...
x,0,'left')
ans =
-1
```

Die Ableitung einer Funktion basiert bekanntermaßen auf dem Prozeß der Grenzwertbildung. Zum Beispiel ist die Ableitungsfunktion der Kosinusfunktion cos die Funktion – sin. Mathematisch bedeutet dies:

$$\frac{d}{dx}\cos(x) = \lim_{h \to 0} \frac{\cos(x+h) - \cos(x)}{h}$$
$$= -\sin(x)$$

Dies können wir mit der limit-Funktion bestätigen.

```
>> syms x h
>> limit( (\cos(x+h)-\cos(x))/h, h, 0) Somit ist der Grenzwert
ans =
-sin(x)
```

Die folgenden Matlab-Anweisungen berechnen den Grenzwert

$$\lim_{x \to 0} \cos(x) = 1$$

```
>> syms x
>> limit(cos(x))
ans = 1
```

Aufgabe 148 (Grenzwerte) Berechnen Sie den Grenzwert

$$\lim_{x \to 0} \frac{\sin x}{x}$$

Lösung: Berechnen Sie den Grenzwert mit

```
>> syms x
>> limit(sin(x)/x)
ans =
```

Somit ist der Grenzwert

$$\lim_{x \to 0} \frac{\sin(x)}{x} = 1.$$

Aufgabe 149 (Grenzwerte) Berechnen Sie den Grenzwert

$$\lim_{h \to 0} \frac{\cos(h) - 1}{h}$$

Lösung: Berechnen Sie den Grenzwert mit

```
>> syms h
>> limit( (cos(h)-1)/h )
ans =
0
```

$$\lim_{h\to 0}\frac{\cos(h)-1}{x}=0.$$

Aufgabe 150 (Grenzwerte) Bestimmen Sie den Grenzwert

$$\lim_{x\to\infty} xe^{-x}$$

und interpretieren Sie ihn.

Lösung: Der Grenzwert ist Null, denn es ist:

Die Exponentialfunktion $f(x) = e^{-x}, x \in \mathbb{R}$ fällt für $x \to \infty$ schneller als die lineare Funktion $g(x) = x, x \in \mathbb{R}$ für $x \to \infty$ steigt. $\odot \ldots \odot$

Aufgabe 151 (Grenzwerte) Bestimmen Sie den Grenzwert

$$\lim_{h \to 0} \frac{(x+h)^3 - x^3}{h}.$$

Was bedeutet dieser Grenzwert?

Lösung: Es ist:

Der Grenzwert ist der Ableitungsterm der Funktion $f(x) = x^3, x \in \mathbb{R}. \odot \dots \odot$

Aufgabe 152 (Grenzwerte) Bestätigen Sie zunächst rechnerisch mit Bleistift und Papier und dann in Matlab die folgenden Grenzwerte:

(a)
$$\lim_{x \to 0} (x^2 - 2x)$$

5)/(cos x) = 5

$$1) - 1/2$$

$$\lim_{x \to \infty} \frac{3^x}{4^x}$$

(a)
$$\lim_{x\to 0} (x^2 - 2x + 1) = 1/2$$

 $5)/(\cos x) = 5$ (d) $\lim_{x\to \infty} 3^x/4^x = 0$
(b) $\lim_{x\to \infty} (2x^2 + 2x)$
 $4)/(x^2 - 1) = 2$ (e) $\lim_{x\to 0} (\sqrt{x+1} + 2x)$
 $1)/(x^2 - 1)/(x^2 - 2x)$
(c) $\lim_{x\to 1} (x - 1)/(x^2 - 2x)$

(e)
$$\lim_{x \to 0} (\sqrt{x+1} + 1)/x = 1/2$$

(c)
$$\lim_{x \to 1} (x - 1)/(x^2 - 1)$$

Lösung:

Aufgabe 153 (Grenzwerte) Die folgenden Grenzwerte existieren nicht. Was liefert Mat-LAB?

(a)
$$\lim_{x \to 0} 1/x$$

(b)
$$\lim_{x\to 0} 1/x$$

(b)
$$\lim_{\substack{x \to 0 \\ x > 0}} 1/x$$
 (c) $\lim_{\substack{x \to 0 \\ x < 0}} 1/x$

Lösung:

67.6. Endliche und unendliche Summen

Endliche oder unendliche Reihen lassen sich mit der Matlab-Funktion symsum berechnen, die symbolisches Summieren erlaubt. Die Summe der endlichen Reihe

$$\sum_{k=1}^{10} 2^k = 2 + 4 + 8 + \dots + 1024$$

ist 2046, was man durch folgende Zeilen bestätigen kann:

Ob die Summe einer unendlichen Reihe existiert und welchen Wert sie gegebenenfalls hat, lässt sich ebenfalls mit der Matlab-Funktion symsum beantworten. In der Analysis beweist man²:

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}.$$

Mit dem Kommando symsum kann man dies überprüfen:

Aufgabe 154 (Endliche Summen) Berechnen Sie

$$\sum_{k=1}^{10} \frac{1}{k}.$$

Lösung: Mit

erhält man

Es ist also

$$\sum_{k=1}^{10} \frac{1}{k} = \frac{7381}{2520} \,.$$

Aufgabe 155 (Endliche Summen) Bestätigen Sie die folgenden Ergebnisse

(a)
$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

(b)
$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

(c)
$$\sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4}$$
.

Lösung: Definiert man zunächst

so erhält man

Aufgabe 156 (Unendliche Summen) Berechnen Sie die Summe der Reihe

$$\sum_{k=1}^{\infty} \frac{1}{4k^2 - 1}.$$

Lösung: Mit

erhält man

Also ist die Summe der Reihe

$$\sum_{k=1}^{\infty} \frac{1}{4k^2 - 1} = \frac{1}{2} .$$

²Zum Beispiel mit Hilfe der Theorie der Fourier-Reihen.

Aufgabe 157 (Unendliche Summen) Bestätigen Sie, dass für die harmonische Reihe gilt

$$\sum_{k=1}^{\infty} (-1)^{k-1} \frac{1}{k} = \ln(2).$$

Lösung: Mit

$$>>$$
 syms k

 $>> symsum((-1)^{(k-1)/k},1,inf)$

erhält man

109(2)

Aufgabe 158 (Unendliche Summen) Berechnen Sie die Leibnizsche Reihe

$$\sum_{k=1}^{\infty} (-1)^k \frac{1}{2k+1}$$

Lösung: Mit

$$>>$$
 syms k

$$>> symsum((-1)^k/(2*k+1),1,inf)$$

erhält man

-1+1/4*pi

Also ist die Leibnizsche Reihe

$$\sum_{k=1}^{\infty} (-1)^k \frac{1}{2k+1} = \frac{\pi}{4} - 1.$$

Aufgabe 159 (Endliche Summen) Berechnen Sie die endliche Reihe

$$\sum_{k=1}^{n} (2k-1)$$

das heißt die Summe der ersten $n \ge 1$ ungeraden natürlichen Zahlen.

Lösung: Mit

erhält man

n^2

Also ist

$$\sum_{k=1}^{n} (2k-1) = n^2.$$

Aufgabe 160 (Endliche Summen) Berechnen Sie die endliche Reihe

$$\sum_{k=1}^{n} (2k-1)^2$$

das heißt die Summe der Quadrate der ersten $n \ge 1$ ungeraden natürlichen Zahlen.

Lösung: Mit

erhält man

Also ist

$$\sum_{k=1}^{n} (2k-1)^2 = \frac{4}{3}n^3 - \frac{1}{3}n.$$

Aufgabe 162 (Ableitungen) Berechnen Sie

$$\frac{d}{dx} \arctan(x)$$

67.7. Differenziation

Die Funktion diff berechnet die symbolische Ableitung eines Funktionsterms. Um einen symbolischen Ausdruck zu erzeugen, muss man zuerst die verwendeten symbolischen Variablen definieren, und dann den gewünschten Ausdruck bilden, und zwar genauso, wie man es mathematisch tun würde. Die folgenden Anweisungen

erzeugen eine symbolische Variable x, bilden den symbolischen Ausdruck $x^2 \exp(x)$ und berechnen die Ableitung von f nach x. Das Ergebnis ist:

Aufgabe 161 (Ableitungen) Berechnen Sie

$$\frac{d}{dx}(x^2e^x)$$

Lösung: Mit

sieht man, dass

$$\frac{d}{dx}(x^2e^x) = 2xe^x + x^2e^x$$

Lösung: Mit den Anweisungen

erkennt man, dass

$$\frac{d}{dx}\arctan(x) = \frac{1}{1+x^2}$$

Aufgabe 163 (Ableitungen) Berechnen Sie

$$\frac{d}{dx}(x^{x^x})$$

Lösung: Mit den Anweisungen

erkennt man, dass

$$\frac{d}{dx}\left(x^{x^{x}}\right) = x^{x^{x}}\left(x^{x}\left(\ln\left(x\right) + 1\right)\ln\left(x\right) + \frac{x^{x}}{x}\right)$$
ist. \odot . . . \odot

67.8. Partielle Differenziation

Es können auch partielle Ableitungen berechnet werden:

```
>> syms w x y
>> diff(sin(w*x*y))
```

berechnet die Ableitung von sin(wxy) nach x. Das Ergebnis ist cos(wxy)wy. Das Ergebnis von

```
>> diff(sin(w*y))
```

ist cos(wy)w. Selbstverständlich kann man auch steuern, nach welcher Variablen differenziert werden soll.

```
>> diff(sin(w*y),w)
```

liefert das Ergebnis

```
ans = cos(w*y)*y
```

Es ist auch möglich, Ableitungen höherer Ordnung zu berechnen.

67.9. Der Gradient

Mit der Funktion jacobian ist es möglich, den Gradient einer reellwertigen Funktion zu berechnen. Für $f: \mathbb{R}^3 \to \mathbb{R}$ mit $f(x, y, z) = e^{x+2y} + 2x\sin(z) + z^2xy$ ist

$$\nabla f(\mathbf{x}) = \begin{bmatrix} e^{x+2y} + 2\sin(z) + z^2y \\ 2e^{x+2y} + z^2x \\ 2x\cos(z) + 2xyz \end{bmatrix}.$$

Wir bestätigen dies in MATLAB.

```
>> syms x y z real
>> jacobian(exp(x+2*y)+...
2*x*sin(z)+z^2*x*y)'
ans =
[ exp(x+2*y)+2*sin(z)+z^2*y]
```

67.10. Die Hesse-Matrix

Der Gradient der Funktion $f(x, y) = 2x^4 + 3x^2y + 2xy^3 + 4y^2$, $(x, y) \in \mathbb{R}^2$ ist

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 8x^3 + 6xy + 2y^3 \\ 3x^2 + 6xy^2 + 8y \end{bmatrix}$$

und die Hesse-Matrix ergibt sich zu

$$H_f(x) = \begin{bmatrix} 24x^2 + 6y & 6x + 6y^2 \\ 6x + 6y^2 & 12xy + 8 \end{bmatrix}.$$

Wir bestätigen dies in Matlab; hilfreich ist wieder die jacobian-Funktion.

```
>> syms x y real

>> f = 2*x^4+3*x^2*y+2*x*y^3+4*y^2;

>> gradf = jacobian(f)'

gradf =

[ 8*x^3+6*x*y+2*y^3]

[ 3*x^2+6*x*y^2+8*y]

>> hessf = jacobian(gradf)

hessf =

[ 24*x^2+6*y, 6*x+6*y^2]

[ 6*x+6*y^2, 12*x*y+8]
```

Im Punkt $(\bar{x}, \bar{y}) = (-2, 3)$ gilt

$$\nabla f(-2,3) = \left[\begin{array}{c} -46 \\ -72 \end{array} \right]$$

und

$$H_f(-2,3) = \begin{bmatrix} 114 & 42 \\ 42 & -64 \end{bmatrix}.$$

In Matlab:

```
>> subs(gradf,{x,y},{-2,3})
ans =
    -46
    -72
>> subs(hessf,{x,y},{-2,3})
ans =
    114    42
    42    -64
```

67.11. Die Jасові-Matrix

Bei Funktionen mehrerer Veränderlicher berechnet man die Funktionalmatrix oder Jacobi-Matrix mittels jacobian. Will man die Jacobi-Matrix der Funktion

$$f: \mathbb{R}^3 \to \mathbb{R}^3$$
$$(x, y, z) \mapsto (xy, x, z)$$

berechnen, so geht das wie folgt:

```
>> syms x y z
>> J = jacobian([x*y,x,z],[x,y,z])
```

```
J =
[ y, x, 0]
[ 1, 0, 0]
[ 0, 0, 1]
```

Im ersten Argument der Function jacobian steht der Funktionsterm und im Zweiten stehen die unabhängigen Variablen.

Aufgabe 164 (Jacobi-Matrizen) Berechnen Sie die Jacobi-Matrix der Funktion $f: \mathbb{R}^2 \to \mathbb{R}^3$, $(x,y) \mapsto (x+y,x^2\sin y,e^{xy})$ in einem beliebigen Punkt $(x,y) \in \mathbb{R}^2$. Wie lautet die Jacobi-Matrix im Punkt (1,0)?

Lösung: Es ist

Die Jacobi-Matrix im Punkt (1,0) ist:

67.12. Integration

Die Funktion int erlaubt symbolische Integration.

Als erstes Beispiel wollen wir die Gleichung $\int \sin(x)dx = -\cos(x) + c \text{ bestätigen.}$

```
>> syms x
>> f = sin(x);
>> int(f)
ans =
-cos(x)
```

Bei parameterabhängigen unbestimmten Integralen der Form

$$F_t(x) = \int f(x, t) dx$$

(*t* ist der Parameter) empfielt es sich *x* und *t* als symbolische Variablen zu definieren. Als Beispiel betrachten wir $\int (x-t)^2 dx = 1/3(x-t)^3 + c$

Sie können die *Symbolic Math Toolbox* heranziehen, um bestimmte Integrale

$$\int_{a}^{b} f(x)dx$$

symbolisch zu lösen. Auch parameterabhängige bestimmte Integrale

$$\phi_t = \int_a^b f(x, t) dx$$

können gelöst werden. Es ist

$$\int_0^{\pi} \sin(x) dx = 2.$$

Wir bestätigen dies mit der Funktion int aus der *Symbolic Math Toolbox*.

Aufgabe 165 (Integration) Berechnen Sie das unbestimmte Integral

$$\int x \sin(x) dx.$$

Lösung: Mit

sieht man, dass

$$\int x \sin(x) dx = \sin(x) - x \cos(x) + c$$

Aufgabe 166 (Integration) Berechnen Sie das bestimmte Integral

$$\int_0^{\pi} x \sin(x) dx$$

symbolisch.

Lösung: Mit

folgt also, dass

$$\int_0^{\pi} x \sin(x) dx = \pi$$

Aufgabe 167 (Integration) Berechnen Sie das uneigentliche Integral

$$\int_{1}^{\infty} \frac{1}{x^5} dx.$$

Lösung: Mit

folgt also, dass

$$\int_{1}^{\infty} \frac{1}{x^5} dx = \frac{1}{4}$$

ist. ©......

Aufgabe 168 (Integration) Berechnen Sie das uneigentliche Integral

$$\int_0^\infty \sin(2t)e^{-st}dt$$

symbolisch. *Lösung:* Mit

>> syms t; syms s positive;
>> int(sin(2*t)*exp(-s*t),t,0,inf);
>> F = simple(ans)
F =
2/(s^2+4)

folgt also, dass

$$\int_0^\infty \sin(2t)e^{-st}dt = \frac{2}{s^2 + 4}$$

ist. Es handelt sich hier um die Laplace-Transformierte der Funktion $f(t) = \sin(2t)$, $t \in \mathbb{R}$. Mit der eingebauten Funktion laplace kann man die Laplace-Transformierte auch unmittelbar berechnen:

Aufgabe 169 (Ableitungen, Integrale) Berechnen Sie symbolisch mit den MATLAB-Funktionen diff und int die Ableitungen bzw. Integrale folgender Exponentialfunktionen:

- (a) e^x
- (b) 2^x
- (c) 10^{x}

(d) a^x

Lösung: » syms x a

(a) >> diff(exp(x))
 ans =
 exp(x)
 >> int(exp(x))
 ans =
 exp(x)

(b) >> diff(2^x)
 ans =
 2^x*log(2)
 >> int(2^x)
 ans =
 1/log(2)*2^x

(c) >> diff(10^x)
ans =
10^x*log(10)
>> int(10^x)
ans =
1/log(10)*10^x

Damit gilt:

$$\begin{array}{cccc}
f(x) & f'(x) & \int f(x)dx \\
e^x & e^x & e^x \\
2^x & \ln(2)2^x & 2^x/\ln(2) \\
10^x & \ln(10)10^x & 10^x/\ln(10) \\
a^x & \ln(a)a^x & a^x/\ln(a)
\end{array}$$

◎......

67.13. Polynome

Wir zeigen nun Beispiele für den Umgang mit "symbolischen Polynomen", siehe Abschnitt 53 für Rechnungen mit "numerischen Polynomen".

Die Funktion horner erlaubt die Darstellung eines Polynoms in Horner-Form.

```
>> syms x
>> p = -3*x^3+3*x^2+10*x+5;
>> ph = horner(p)
ph =
5+(10+(3-3*x)*x)*x
```

Will man dieses zum Beispiel an der Stelle x = 3 auswerten, so geht das wie folgt:

```
>> subs(ph,x,3)
ans =
-19
```

Will man den Grad eines Polynoms bestimmen, so hilft die MAPLE-Funktion degree.

```
>> maple('degree',p)
ans =
3
```

Weitere Matlab bzw. Maple Funktionen zum Rechnen mit Polynomen finden Sie in der Tabelle 37. Hilfe zu den Maple-Funktionen erhalten Sie mit mhelp <Funktionsname>.

Funktion	Bedeutung
coeff	Koeffizient (MAPLE)
convert	Konvertiert (Maple)
degree	Grad des Polynoms (MAPLE)
factor	Linearfaktoren (MAPLE)
horner	Horner-Darstellung
solve	Nullstellen
subs	Ersetzen

Tabelle 37: Zum symbolischen Rechnen mit Polynomen

67.14. Taylor-Polynome

Die Funktion taylor erlaubt das symbolische Berechnen des Taylor-Polynoms einer Funktion. Zum Beispiel liefert

```
>> syms x
>> taylor(sin(x))
```

das Taylor-Polynom bis zur fünften Ordnung:

```
ans = x-1/6*x^3+1/120*x^5
```

Aufgabe 170 (Taylor-Polynome) Berechnen Sie das Taylor-Polynom der Kosinus-Funktion

$$f(x) = \cos(x)$$

vom Grad vier im Entwicklungspunkt 0.

Lösung: Es ist

```
>> syms x
>> taylor(cos(x),5,x,0)
ans =
1-1/2*x^2+1/24*x^4
```

d.h. das Taylor-Polynom der Kosinus-Funktion vom Grad vier im Entwicklungspunkt 0 ist

$$f(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24} .$$



67.15. Die Funktionen funtool und taylortool

Die Funktionen funtool und taylortool stellen pädagogische Hilfsmittel zur Verfügung. Die Funktion funtool ist eine Art grafischer Taschenrechner, der wichtige Operationen und einfache Visualisierungen bei Funktionen einer Veränderlichen ermöglicht, siehe Abbildung 68.

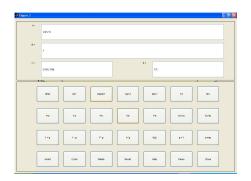


Abbildung 68: Das GUI von funtool

Mit Hilfe des Kommandos taylortool können Sie die grafische Oberfläche des Programms taylortool starten und damit Taylor-Approximationen studieren, siehe Abbildung 69.

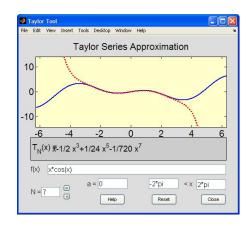


Abbildung 69: Das GUI von taylortool

67.16. Mehrdimensionale TayLor-Polynome

Zur Berechnung des Taylor-Polynoms einer reellwertigen Funktion mehrerer Variablen kann man die Funktion mtaylor aus Maple verwenden.

Das Taylor-Polynom zweiten Grades der Funktion $f(x,y)=e^{x+y}, (x,y)\in\mathbb{R}^2$ mit Entwicklungspunkt (0,0) ist gegeben durch $T_2(x,y)=1+x+y+1/2(x^2+y^2)+xy$. Hier die Bestätigung in Matlab:

Aufgabe 171 (Taylor) Berechnen Sie das Taylor-Polynom zweiter Ordnung mit dem Entwicklungspunkt (-1/2, -1/2) der quadratischen Funktion $q(x, y) = x^2 + y^2 + x + y$, $(x, y) \in \mathbb{R}^2$.

Lösung: Die folgenden Matlab-Zeilen berechnen die Lösung:

```
>> syms x y
>> T2 = maple('mtaylor',...
x^2+y^2+x+y,'[x=-1/2,y=-1/2]',3)
T2 =
-1/2+(x+1/2)^2+(y+1/2)^2
```

Wir bestätigen das Ergebnis:

```
>> simplify(T2)
ans =
x^2+y^2+x+y
```

Aufgabe 172 (Taylor) Approximieren Sie die Funktion $f(x,y) = xe^y - x^3y$, $(x,y) \in \mathbb{R}^2$ in der Nähe des Nullpunktes durch ihr Taylor-Polynom T_2 zweiten Grades. Vergleichen Sie dann die Funktionswerte von f und T_2 an der Stelle (0.5,0.5), indem Sie den relativen Fehler $|T_2(0.5,0.5) - f(0.5,0.5)|/|f(0.5,0.5)|$ berechnen.

Lösung: Es ist

```
>> syms x y
>> f = x*exp(y)-x^3*y;
>> T2 = maple('mtaylor',f,'[x,y]',3)
T2 =
    x+x*y
>> subs(T2,[x,y],[0.5,0.5]),...
subs(f,[x,y],[0.5,0.5])
ans =
        0.7500
ans =
        0.7619
>> abs(subs(T2,[x,y],[0.5,0.5])...
-subs(f,[x,y],[0.5,0.5]))...
```

```
/abs(subs(f,[x,y],[0.5,0.5]))
ans =
    0.0156
```

Der relative Fehler ist demnach zirka 1.5%. ©

67.17. Lineare Algebra

Siehe Abschnitt 48

67.18. Differenzengleichungen

Maple kann Differenzengleichungen lösen. Die Funktion rsolve kann lineare Differenzengleichungen mit konstanten Koeffizienten, Systeme linearer Differenzengleichungen mit konstanten Koeffizienten und manche nichtlineare Gleichungen erster Ordnung. Implementiert sind Standardtechniken wie erzeugende Funktionen und z-Transformationen, sowie Methoden basierend auf Substitutionen und charakteristischen Gleichungen.

Die lineare Differenzengleichung erster Ordnung

$$y_t = -0.9y_{t-1}$$

hat die Lösung

$$y_t = (-1)^t (0.9)^t y_0.$$

Hier die Bestätigung in MATLAB.

```
>> maple('rsolve',...
'y(t)=-0.9*y(t-1)','y(t)')
ans =
y(0)*(-9/10)^t
```

Die Komplexität der Gaussschen Elimination ist

Aufgabe 173 (Differenzengleichungen) Bestätigen Sie die Lösung $y_t = (0.5)^t (y_0 - 12) + 12$ der Differenzengleichung

$$y_t = 0.5y_{t-1} + 6$$

Lösung: Es ist

```
>> maple('rsolve',...
'y(t)=0.5*y(t-1)+6','y(t)')
ans =
y(0)*(1/2)^t-12*(1/2)^t+12
```

67.19. Differenzialgleichungen

Mit Hilfe der Matlab-Funktion dsolve ist es möglich, allgemeine Lösungen von Differenzialgleichungen zu berechnen, Anfangswertaufgaben und Randwertaufgaben zu lösen. Hierbei kann eine einzelne Differenzialgleichung oder aber auch ein System von Gleichungen vorliegen. Mit der Funktion pdsolve können sogar einfache partielle Differenzialgleichungen bzw. partielle Differenzialgleichungssysteme gelöst werden.

Die gewöhnliche Differenzialgleichung y' = ay hat die allgemeine Lösung $y(t) = ce^{at}$. In Matlab bestätigt man das wie folgt:

```
>> dsolve('Dy = a*y')
ans =
exp(a*t)*C1
```

Aufgabe 174 (Differenzialgleichung) Berechnen Sie die allgemeine Lösung der gewöhnlichen Differenzialgleichung

$$y'(t) = -y(t) - 5e^{-t}\sin(5t)$$

siehe Abschnitt 62.

Lösung: Die allgemeine Lösung ist

Aufgabe 175 (Anfangswertaufgabe) Berechnen Sie die Lösung der Anfangswertaufgabe

AWA:
$$\begin{cases} y'(t) = -y(t) - 5e^{-t}\sin(5t) \\ y(0) = 1 \end{cases} t \ge 0$$

aus Abschnitt 62.

Lösung: Die Lösung ist

```
>> y = dsolve('Dy=-y-5*...
exp(-t)*sin(5*t)','y(0)=1')
y =
exp(-t)*cos(5*t)
```

Aufgabe 176 (Anfangswertaufgabe) Berech-

nen Sie die Lösung der Anfangswertaufgabe

AWA:
$$\begin{cases} y''(t) + 144y(t) = \cos(11t) \\ y(0) =, \ y'(0) = 0 \end{cases} \quad t \ge 0$$

und stellen Sie die Lösung im Intervall $[0, 6\pi]$ dar.

Lösung: Die Lösung ist

Eine Differenzialgleichung dieses Types nennt man die Gleichung des (ungedämpften) harmonischen Oszillators mit äußerer harmonischer Anregung, hier $\cos(11t)$. Die Lösung $y(t) = 1/23\cos(11t) - 1/23\cos(12t)$ ist eine Überlagerung von zwei Schwingungen mit unterschiedlichen Frequenzen. Man spricht von einer Schwebung (amplitudenmodulierte Schwingung), eine Bewegungsform, die für die Akustik und Funktechnik von großer Bedeutung ist. Die Abbildung 70 zeigt die Lösung.

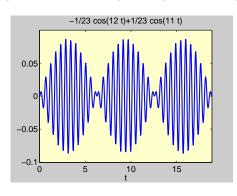


Abbildung 70: Lösung der AWA

Wir berechnen die Lösung der Randwertaufgabe

RWA:
$$\begin{cases} u'' = 6x \\ u(0) = 0, u(1) = 1 \end{cases} 0 \le x \le 1.$$

aus Abschnitt 62.2.

Aufgabe 177 (Randwertaufgabe) Berechnen Sie die Lösung der Randwertaufgabe

RWA:
$$\begin{cases} y'' = -y \\ y(0) = 3, y(\pi/2) = 7 \end{cases} \quad 0 \le x \le \pi/2.$$

aus Abschnitt 62.2.

Lösung: Es ist

Mit der Maple-Funktion pdsolve kann man einfache partielle Differenzialgleichungen lösen. Als Beispiel betrachten wir die eindimensionale homogene Wellengleichung

$$u_{tt} - c^2 u_{xx} = 0$$
 $(c \neq 0)$.

Dies ist eine lineare partielle hyperbolische Differenzialgleichung zweiter Ordnung. Um diese symbolisch zu lösen, definieren wir zunächst die symbolischen Variablen x, t, c und die Funktion u(x,t):

```
>> syms x t c
>> u = sym('u(x,t)');
```

Danach berechnen wir die Ableitungen zweiter Ordnung

```
>> uxx = diff(u,x,2);
>> utt = diff(u,t,2);
```

und schließlich die allgemeine Lösung

```
>> maple('pdsolve',utt-c^2*uxx)
u(x,t) = _F1(t*c+x)+_F2(t*c-x)
```

Hierbei sind _F1 und _F2 zwei beliebige zweimal stetig differenzierbare Funktionen. Wählt man für diese beiden Funktionen die Funktion e^{-z^2} , $z \in \mathbb{R}$ und für den Parameter c den Wert 1, so erhält man die Lösungsfunktion $u(x,t) = e^{-(t+x)^2} + e^{-(t-x)^2}$. Die Abbildung 71 zeigt den

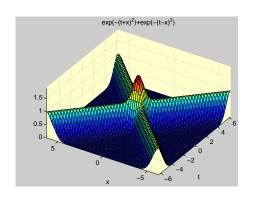


Abbildung 71: Lösung der eindimensionalen Wellengleichung

Graph der Lösungsfunktion, der mit

```
>> ezsurf('exp(-(t+x)^2)+
exp(-(t-x)^2)')
```

erzeugt wurde. Man erkennt, dass die allgemeine Lösung der homogenen Wellengleichung eine Überlagerung zweier gegenläufiger Wellen darstellt.

Aufgabe 178 (Differenzialgleichung) Berechnen Sie mit pdsolve eine Lösung der partiellen Differenzialgleichung

$$axu_x + btu_t = 0.$$

Lösung: Es ist

```
>> syms x t a b
u = sym('u(x,t)');
ux = diff(u,x);
ut = diff(u,t);
maple('pdsolve',a*x*ux+b*t*ut)
ans =
u(x,t) = _F1(t/(x^(b/a)))
```

Aufgabe 179 (Differenzialgleichung) Zeigen Sie mit der Matlab *Symbolic Toolbox*, dass die jeweils angegebene Lösung *y* die vorgegebene Differenzialgleichung löst.

(a)
$$y' + ty = t$$

 $y(t) = 1 + e^{-t^2/2}$.

(b)
$$y' + y^2 = 0$$

 $y(t) = 1/(t-3)$.

(c)
$$yy' + ty = 0$$

 $y(t) = 10 - t^2/2$.

Lösung: Es ist

```
>> dsolve('Dy+t*y=t')
ans =
1+exp(-1/2*t^2)*C1
```

Aufgabe 180 (Differenzialgleichung) Bestätigen Sie mit der *Symbolic*-Toolbox anhand der Funktion

$$f(x) = x \cdot \sin^2(x)$$

die Gültigkeit des Hauptsatzes der Differenzial- und Integralrechnung. Eine kleine Rechnung ist nötig.

Lösung: Es ist

©.....

Aufgabe 181 (Differenzialgleichung) Benutzen Sie dsolve, um die Lösung der folgenden Anfangswertaufgaben zu finden. Zeichnen Sie jede Lösung über dem angegebenen Intervall.

(a)
$$y' + ty = t$$

 $y(0) = -1$, [-4, 4].

(b)
$$y' + y^2 = 0$$

 $y(0) = 2$, [0,5].

(c)
$$yy' + ty = 0$$

 $y(1) = 4$, [-4, 4].

Lösung: Es ist

67.20. Die kontinuierliche Fourier-Transformation

Die kontinuierliche Fourier-Transformation (CFT) untersucht, welche Frequenzen mit welchen Amplituden in einem Zeitsignal f(t) enthalten sind, wenn das Zeitsignal nicht peri-

odisch ist. Man nennt dieses Vorgehen –wie bei den Fourier-Reihen– die Frequenzanalyse des Zeitsignals f(t). Man kann sagt auch, die kontinuierliche Fourier-Transformation (CFT) zerlegt ein Signal in sinusförmige Wellen unterschiedlicher Frequenzen.

Die kontinuierliche Fourier-Transformation (CFT) drückt ein Signal f(t) als eine "stetige Linearkombination" von Sinus- und Cosinusfunktionen aus

$$\hat{f}(\omega) = \int_{\mathbb{R}} f(t)e^{-i\omega t}dt.$$

Die Abbildung $f\mapsto \hat{f}$ heißt kontinuierliche Fourier-Transformation (CFT). Stellt die unabhängige Variable t die Zeit dar, so hat die Variable ω die Bedeutung der Frequenz. Somit ist die Frequenz ω eine kontinuierliche Größe, das heißt alle möglichen Frequenzen sind darstellbar. Die kontinuierliche Fourier-Transformation (CFT) bildet die Funktion f vom Originalbereich (Zeitbereich) in den Bildbereich (Frequenzbereich) ab. Die inverse Fourier-Transformation (ICFT) transformiert umgekehrt.

Für den Rechteckimpuls

$$f(t) = \begin{cases} 1, & \text{falls } |t| < T \\ 0, & \text{sonst} \end{cases}$$

soll die Fourier-Transformierte $\hat{f}(\omega)$ berech-

net werden. Es ergibt sich

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = \int_{-T}^{T} e^{-i\omega t}dt$$

$$= \left[\frac{e^{-i\omega t}}{-i\omega}\right]_{-T}^{T} = \frac{1}{-i\omega}(e^{-iT\omega} - e^{iT\omega})$$

$$= \frac{2}{\omega} \frac{1}{2i} \left(e^{iT\omega} - e^{-iT\omega}\right)$$

$$= \frac{2\sin(\omega T)}{\omega}.$$

Die folgenden Anweisungen bestätigen die Rechnung.

```
>> syms t T
>> Rechteck = heaviside(t+T)-...
heaviside(t-T);
>> fourier(Rechteck)
ans =
2/w*sin(T*w)
```

Die folgenden Anweisungen geben eine zweite Möglichkeit, obiges Resultat zu bestätigen. Es ist T = 1.

```
>> maple('T:=1');
>> maple('Rechteck(t/T):=
Heaviside(t+T)-Heaviside(t-T):');
>> maple('simplify(fourier(Rechteck(t/T),t,w))')
ans =
2/w*sin(w)
```

Das folgende Beispiel zeigt, dass die Fourier-Transformierte $\hat{f}(\cdot)$ einer Funktion $f(\cdot)$ im allgmeinen eine komplexwertige Funktion ist.

Für die von den Parametern a > 0 und b > 0 abhängige Funktion

$$f(t) = \begin{cases} ae^{-bt}, & \text{falls } t > 0\\ 0, & \text{sonst} \end{cases}$$

soll die Fourier-Transformierte $\hat{f}(\omega)$ berechnet werden. Es ergibt sich

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt$$

$$= \int_{0}^{\infty} ae^{-bt}e^{-i\omega t}dt = a\int_{0}^{\infty} e^{-(b+i\omega)}dt$$

$$= a\left[\frac{e^{-(b+i\omega)t}}{-(b+i\omega)}\right]_{t=0}^{t=\infty} = \frac{a}{b+i\omega}.$$

Die folgenden Anweisungen bestätigen das Ergebnis symbolisch in MATLAB.

```
>> maple('assume(a>0,b>0)');
>> maple('fourier(a*exp(-b*t)
*Heaviside(t),t,w)')
ans =
a/(b+i*w)
```

Beachten Sie, dass die Konstanten *a* und *b*, so wie es die Theorie erfordert, größer als Null definiert werden müssen, weil sonst gegebenenfalls das Integral bzw. die Fourier-Transformation nicht existiert.

Die Fourier-Transformierte von

$$f(x) = e^{-x^2}$$

ist

$$\hat{f}(\xi) = \sqrt{\pi}e^{-\xi^2/4}.$$

Wir bestätigen dies symbolisch mit der fourier Funktion

```
>> syms x
>> f = exp(-x^2);
>> fhat = fourier(f)
fhat =
pi^(1/2)*exp(-1/4*w^2).
```

Aufgabe 182 (Fourier-T.) Berechnen Sie zunächst per Hand und dann mit Matlab die Fourier-Transformierte von

$$f(x) = e^{-|x|}.$$

Lösung: Die Fourier-Transformierte von

$$f(x) = e^{-|x|}$$

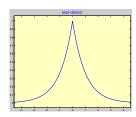
ist

$$\hat{f}(\xi) = \frac{2}{1+\xi^2},$$

denn

$$\begin{split} \hat{f}(\xi) &= \int_{\mathbb{R}} e^{-|x|} e^{-ix\xi} dx \\ &= \int_{0}^{\infty} e^{-|x|} e^{-ix\xi} dx + \int_{-\infty}^{0} e^{-|x|} e^{-ix\xi} dx \\ &= \int_{0}^{\infty} e^{-x} e^{-ix\xi} dx + \int_{0}^{\infty} e^{-x} e^{ix\xi} dx \\ &= \int_{0}^{\infty} e^{x(-1-i\xi)} dx + \int_{0}^{\infty} e^{x(-1+i\xi)} dx \\ &= \lim_{r \to \infty} \left[\frac{e^{-x(1+i\xi)}}{-1-i\xi} + \frac{e^{-x(1-i\xi)}}{-1+i\xi} \right]_{x=0}^{x=r} \\ &= -\frac{1}{-1-i\xi} - \frac{1}{-1+i\xi} \\ &= \frac{2}{1+\xi^2}. \end{split}$$

Wir bestätigen dies symbolisch



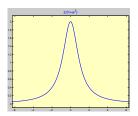


Abbildung 72: Zur kontinuierlichen Fourier-Transformation

Beachten Sie, dass die unabhängige Variable der Fourier-Transformierten w ist. Die Abbildung 72 zeigt die Funktion f und ihre Fourier-Transformierte \hat{f} . \odot \odot

Die diskrete Fourier-Transformierte haben wir in Abschnitt 60 besprochen.

67.21. Laplace-Transformation

Die Laplace-Transformation kann symbolisch mit der Funktion laplace berechnet werden. Siehe Aufgabe 168 für ein Beispiel.

Aufgabe 183 (Laplace-T.) Berechnen Sie die Laplace-Transformation von der Funktion $f(t) = e^{-at}$, $t \in \mathbb{R}$, wobei a ein reeller Parameter ist.

Lösung: Es ist

```
>> syms a t
laplace(exp(-a*t))
ans =
1/(a+s)
```

67.22. Spezielle mathematische Funktionen

Über 50 spezielle mathematische Funktionen stehen mit der *Symbolic Math Toolbox* zur Verfügung. Diese Funktionen werden mit mfun angesprochen und werten die entsprechende Funktion an der gewünschten Stelle numerisch aus. Dies stellt somit auch eine Erweiterung zum Standard-Matlab dar, denn diese klassischen Funktionen stehen dort nicht zur Verfügung. Siehe doc mfunlist (help mfunlist).

67.23. Variable Rechengenauigkeit

Zusätzlich zu Matlab's doppelt genauer Gleitpunktarithmetik und der symbolischen Rechenmöglichkeit wird durch die *Symbolic Toolbox* eine variable Rechengenauigkeitsarithmetik unterstützt, die durch den Maple-Kern durchgeführt wird. Diese Art von Arithmetik zu verwenden, ist dann sinnvoll, wenn eine genaue Lösung verlangt wird, aber eine exakte unmöglich oder aber zu zeitaufwendig zu berechnen ist.

Mit der Funktion digits ist es möglich, die Anzahl der genauen dezimalen Stellen einzustellen, mit der die Rechnungen durchgeführt werden sollen. Standardmäßig sind 32 Stellen eingestellt, was Sie durch den Aufruf von digits überprüfen können. Wollen Sie diese Zahl zu n abändern, so geht das mit digits(n). Die variable Rechengenauigkeitsarithmetik basiert auf dem vpa-Kommando. Der einfachste Gebrauch besteht zum Beispiel darin, die Kreiszahl π auf 32 Stel-

len zu berechnen:

```
>> digits
Digits = 32
>> vpa(pi)
ans =
3.1415926535897932384626433832795
```

Wie der Aufruf vpa (pi, 40) zeigt, können Sie mit dem zweiten Argument von vpa die zur Zeit aktuelle Anzahl von gültigen Dezimalen überschreiben.

Das nächste Beispiel berechnet die Eulersche Zahl *e* auf 50 Nachkommastellen genau, gibt sie aus und überprüft, ob der Logarithmus davon wieder 1 ergibt:

Vorsicht Falle:

```
>> x = vpa(sym(exp(1)))
x =
2.71828182845904553488480814849026
50117874145507813
>> vpa(log(x))
ans =
1.000000000000000011018891328384949
58218182413442295
```

Wir haben auf die Hochkomma in exp(1) verzichtet. Das Resultat ist, dass Matlab exp(1) mit doppelter Genauigkeit berechnet, diese 16-stellige Dezimalzahl in eine 50-stellige Dezimalzahl transformiert (34 Stellen sind somit bedeutungslos) und dann den Logarithmus anwendet, was zu einem ungenauen Ergebnis führt. Erst die Hochkomma sorgen dafür, dass der Matlab-Interpreter übergangen wird, damit Maple den Ausdruck auswerten kann.

Aufgabe 184 (Symbolisches Rechnen) Berechnen Sie π auf 50 Stellen genau!

Lösung: Sowohl

```
>> digits(50)
>> vpa(pi)
```

als auch

```
>> vpa pi 50
```

liefern

ans = 3.14159265358979323846264338327 95028841971693993751

Aufgabe 185 (Geschwindigkeit) Sowohl

$$\sum_{n=0}^{\infty} \frac{1}{n!} = \lim_{n \to \infty} \sum_{k=0}^{n} \frac{1}{k!}$$

als auch

$$\lim_{n\to\infty}(1+\frac{1}{n})^n$$

ergeben die Eulersche Zahl e = 2.71...

- (a) Verifizieren Sie die beiden Aussagen.
- (b) Berechnen Sie die Eulersche Zahl auf zehn Dezimalen.

(c) Vergleichen Sie nun die Konvergenzgeschwindigkeiten der beiden Folgen, die für n = 0, 1, 2, ... durch $s_n = \sum_{k=0}^n 1/k!$ und $a_n = (1 + 1/n)^n$ gegeben sind, indem Sie die folgende Tabelle vervollständigen.

S_n	a_n
2.5000000000	2.2500000000
2.7083333333	2.4414062500

Lösung:

(a) Hier die Verifikation. Zum Einen ist

```
>> symsum(1/sym('n!'),0,inf)
ans =
exp(1)
```

und zum Anderen gilt

(b) Die Eulersche Zahl auf zehn Nachkommastellen ist

```
>> digits(11)
>> e = vpa(exp(1))
e =
2.7182818285
```

(c) Mit den Zeilen

```
>> an = (1+1./n).^n;
>> for i=1:10
>> sn(i,1) = ...
     symsum(1/sym('k!'),0,n(i));
>> end
```

erhält man den gewünschten Vergleich.

```
>> digits(11)
>> [vpa(sn) vpa(an')]
ans =
[ 2.50000000000, 2.25000000000]
[ 2.70833333333, 2.4414062500]
[ 2.7180555556, 2.5216263717]
[ 2.7182787698, 2.5657845140]
[ 2.7182818011, 2.5937424601]
[ 2.7182818283, 2.6130352902]
[ 2.7182818285, 2.6271515563]
[ 2.7182818285, 2.6379284974]
[ 2.7182818285, 2.6464258211]
[ 2.7182818285, 2.6532977051]
```

67.24. Überblick über alle symbolischen Funktionen

Mit doc symbolic (help symbolic) erhalten Sie einen Überblick über alle Funktionen der *Symbolic Math Toolbox*.

67.25. Weitere Bemerkungen und Hinweise

Es gibt zwei Levels der *Symbolic Math Tool-box*.

• Die Standard Symbolic Math Toolbox verfügt über annähernd 50 eingebaute Mat-

Lab-Funktionen und Kommmandos, die die entsprechenden Maple-Befehle aufrufen. Eine Liste dieser Funktionen erhalten Sie mit help symbolic. Darüber hinaus besteht die Möglichkeit, andere Maple-Kommandos anzusprechen, die zur Standardbibliothek von Maple gehören. Hierzu verwendet man die maple-Funktion.

 Die Extended Symbolic Math Toolbox besitzt alle Funktionalitäten der Standard Symbolic Math Toolbox, und darüber hinaus besteht die Möglichkeit, die Maple-Programmierstrukturen, die Ein- und Ausgabe Möglichkeiten als auch Funktionen und Kommandos spezieller Maple-Pakete zu nutzen.

Mit dem Aufruf

```
>> mhelp index[packages]
```

erhalten Sie eine Liste über die Maple-Pakete.

Wenn Sie sich nun näher für ein Paket interessieren, zum Beispiel für das Paket mit den kombinatorischen Funktionen, so hilft der Aufruf mhelp combinat weiter. Dieser listet dann alle relevanten Funktionen auf. Mit maple('with(combinat)') laden Sie das Paket und mit mhelp binomial erhalten Sie zum Beispiel weitere Informationen über die Funktion binomial, etwa wie man sie aufzurufen hat.

Das Paket LinearAlgebra enthält die Funktion Basis. Mit mhelp Basis erhalten Sie die Meldung

```
Multiple matches found:
SolveTools,Basis
Modular,Basis
```

LinearAlgebra, Basis

Dann hilft mhelp ('Linear Algebra, Basis') oder mhelp Linear Algebra [Basis] weiter. Eine Online-Einführung in die *Symbolic Math Toolbox* erhalten Sie mit dem Befehl demos. Geben Sie hierzu demos nach dem Matlab-Prompt ein. Es wird dann das Matlab-Demo-Fenster geöffnet.

68. Nichtlineare Gleichungen (2)

Mit der Funktion fsolve, die zur *Optimizati*on *Toolbox* gehört, kann man nichtlineare Gleichungssysteme lösen.

Wir wollen eine Nullstelle des nichtlinearen Gleichungssystems

$$16x^{4} + 16y^{4} + z^{4} - 16 = 0$$
$$x^{2} + y^{2} + z^{2} - 3 = 0$$
$$x^{3} - y = 0$$

mit drei Gleichungen und drei Variablen finden. Hierzu starten wir den iterativen Algorithmus mit $x_0 = (1, 1, 1)$ und definieren die vektorwertige Funktion $F : \mathbb{R}^3 \to \mathbb{R}^3$, $F(x, y, z) = (16x^4 + 16y^4 + z^4 - 16, x^2 + y^2 + z^2 - 3, x^3 - y)$ in dem Function-File MyFunction.m wie folgt:

Dann rufen wir den Löser fsolve auf und erhalten eine Nullstelle des Systems:

```
>> [x,fval] = fsolve(@MyFunction,x0)
Optimization terminated: ...
x =
     0.8896
     0.7040
     1.1965
fval =
    1.0e-012 *
     0.1243
     0.0022
     0.0022
```

69. Optimierung (Teil 2)



Mit der *Optimization Toolbox* kann man Optimierungsaufgaben lösen, insbesondere solche, wo Nebenbedingungen (Restriktionen) auftreten. Diese sind in der Praxis auch meist vorhanden. Sie treten linear als auch nichtlinear auf. Für folgende Optimierungsmodelle stehen Matlab-Funktionen zur Verfügung: Lineare Optimierung, Quadratische Optimierung, Minmax Optimierung, Mehrzielige Optimierung, Semiinfinite Optimierung und Nichtlineare Optimierung. Siehe doc optim (help optim) für eine komplette Übersicht über al-

le in Matlab lösbaren Optimierungsmodelle. Zu beachten ist, dass jedes Optimierungsproblem als Minimierungsproblem formuliert werden muss.

69.1. Lineare Optimierung

An einem einfachen Beispiel zur Linearen Optimierung zeigen wir, wie man unter MATLAB solche Probleme löst. Als Beispiel betrachten wir die Optimierungsaufgabe (u.d.N. steht als Abkürzung für unter den Nebenbedingungen)

Minimiere

$$(x_1, x_2) \in \mathbb{R}^2$$
 $-2x_1 - 3x_2$
 $x_1 + 2x_2 \le 10$
u.d.N. $x_1 + x_2 \le 6$
 $x_1 \ge 0, x_2 \ge 0$

Mit den Anweisungen

$$c = [-2; -3]; A = [1 2; 1 1];$$

 $b = [10; 6]; u = [0; 0];$

und dem Funktionsaufruf

erhalten wir die Ausgabe

Das bedeutet, dass $x^* = (x_1^*, x_2^*) = (2, 4)$ die Minimalstelle des linearen Optimierungsproblems ist, und -(2)(2) - (3)(4) = -16 der Zielfunktionswert, das heißt das Minimum ist.

Das erste Argument c der Funktion linprog beinhaltet die Koeffizienten der Zielfunktion. Die Matrix A beinhaltet die Koeffizienten der linearen Ungleichungen, wobei die Koeffizienten der rechten Seiten der Ungleichungen im Vektor b stehen. Schließlich werden die unteren Grenzen der Variablen im Vektor u gespeichert und als sechstes Argument in linprog übergeben. Das vierte und fünfte Argument der Funktion linprog ist für eventuell vorkommende lineare Gleichungen vorgesehen.

Allgemein gilt: Mit der Funktion linprog können lineare Optimierungsaufgaben der Form

Minimiere
$$x \in \mathbb{R}^n$$
 $c^T x$ $Ax \le b$ u.d.N. $Bx = d$ $l \le x \le u$

gelöst werden. Die Vektoren l und u dürfen auch Koordinaten $-\infty$ bzw. ∞ haben. Falls der Aufruf

erfolgreich war, findet man in der Variablen x den berechneten Lösungsvektor und in der Variablen fwert den zugehörigen Wert der Zielfunktion $c^Tx =: f(x)$. Wie bei allen anderen Matlab-Funktionen entsprechenden Typs können Argumente weggelassen werden, wenn nicht alle Formen der Nebenbedingungen auftreten. Damit aber die Reihenfolge erhalten bleibt, müssen gegebenenfalls eckige Klammern [] hierfür geschrieben werden. Ferner können Optionen gesetzt werden und weitere Ausgabeargumente angegeben werden. Für weitere Hinweise verweisen

wir auf die Online-Hilfe, siehe doc linprog (help linprog).

Aufgabe 186 (Lineare Optimierung) Die lineare Optimierungsaufgabe

Minimiere
$$x \in \mathbb{R}^2$$
 $-x_1 - x_2$ $x_1 + 3x_2 \le 13$ $3x_1 + x_2 \le 15$ $-x_1 + x_2 \le 3$ $x_1 \ge 0, x_2 \ge 0$

hat die Minimalstelle $x^* = (4,3)$ mit dem Minimum -7, wie die Abbildung 73 grafisch

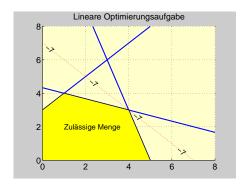


Abbildung 73: Lineare Optimierung

zeigt. Bestätigen Sie die Lösung in Matlab mit der Funktion linprog.

Lösung: Die Lösung erhält man in Matlab wie folgt:

69.2. Quadratische Optimierung

Die Quadratische Optimierung gehört bereits in die Klasse der Nichtlinearen Optimierungsaufgaben, stellt aber bereits einen wichtigen Spezialfall dieser Problemklasse dar, sowohl aus theoretischen als auch aus algorithmisch numerischen Gründen. Quadratische Programme treten im Bereich der Regelungstechnik häufig bei der Online-Optimierung von Stellgrößen linearer Systeme mit quadratischen Gütefunktionalen und linearen Nebenbedingungen auf. Lineare Regressionsprobleme sind wiederum wichtige Spezialfälle Quadratischer Optimierungsaufgaben.

Eine Quadratische Optimierungsaufgabe liegt vor, wenn die Zielfunktion quadratisch und die Nebenbedingungen wie bei der Linearen Optimierung linear sind. Die allgemeine Problemstellung ist:

Minimiere
$$x \in \mathbb{R}^n$$
 $\frac{1}{2}x^T H x + c^T x$
 $Ax \le b$
 $Bx = d$
 $l \le x \le u$

Die Matrix H in der Zielfunktion ist aus $\mathbb{R}^{n\times n}$ und symmetrisch. Ansonsten gilt analoges wie bei Linearen Optimierungsaufgaben. Falls der Aufruf

[x,qWert] = quadprog(c,A,b,B,d,l,u)

erfolgreich war, findet man in der Variablen \mathbf{x} den berechneten Lösungsvektor und in der Variablen qWert den zugehörigen Wert der Zielfunktion $1/2\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x} =: q(\mathbf{x})$. Wie bei allen anderen Matlab-Funktionen entsprechenden Typs können Argumente weggelassen werden, wenn nicht alle Formen der Nebenbedingungen auftreten. Damit aber die Reihenfolge erhalten bleibt, müssen gegebenenfalls eckige Klammern [] hierfür geschrieben werden. Ferner können Optionen gesetzt werden und weitere Ausgabeargumente angegeben werden. Für weitere Hinweise verweisen wir auf die Online-Hilfe, siehe doc quadprod (help quadprog).

Als Beispiel betrachten wir die Aufgabe

Minimiere
$$x \in \mathbb{R}^2$$
 $q(x_1, x_2)$ $x_1 + x_2 \le 200$ $1.25x_1 + 0.75x_2 \le 200$ $x_2 \le 150$ $x_1 \ge 0, x_2 \ge 0.$

mit der quadratischen Zielfunktion $q(x_1, x_2) = -(4+2x_1)x_1 - (5+4x_2)x_2 - 8x_1x_2, \mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$.

Es ist dann

$$H = \begin{bmatrix} -4 & -8 \\ -8 & -8 \end{bmatrix}, \quad c = \begin{bmatrix} -4 \\ -5 \end{bmatrix},$$

$$A = \begin{bmatrix} 1 & 1 \\ 1.25 & 0.75 \\ 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 200 \\ 200 \\ 150 \end{bmatrix}$$

und l = (0, 0). Nach den Zuweisungen

$$H = [-4 -8; -8 -8]; c = [-4; -5];$$
 $A = [1 1; 1.25 0.75; 0 1];$
 $b = [200; 200; 150];$
 $1 = [0; 0];$

und dem Aufruf

erhält man die Ausgabe

Warning: Large-scale method does not currently solve this problem formulation, switching to medium-scale method.

> In quadprog at 236
Optimization terminated.

x = 50.0000 150.0000 qWert =

-155950

69.3. Überblick über alle Funktionen zur Optimierung

Mit doc optim (help optim) erhalten Sie einen Überblick über alle Funktionen der *Optimization Toolbox*.

70. Nichtlineare Ausgleichsaufgaben

Zu der *Optimization Toolbox* gehören auch die Funktionen lsqnonlin und lsqcurvefit, siehe doc lsqnonlin (help lsqnonlin) und doc lsqcurvefit (help

1sqcurvefit). Mit diesen Funktionen kann man nichtlineare Ausgleichsaufgaben lösen. Nichtlineare Ausgleichsaufgaben treten bei Parameteridentifikationsproblemen auf und spielen in den Anwendungen eine große Rolle. Wenn Sie sich für diesen Problemkreis näher interessieren, sehen Sie auch unter den Worten Inverse Probleme nach ([21]). Wir zeigen das Lösen einer nichtlinearen Ausgleichsaufgabe in folgender Aufgabe.

Aufgabe 187 (lsqcurvefit) Gegeben sind die vier Datenpunkte

und die nichtlineare Modellfunktion

$$\phi(t, \mathbf{x}) = x_1 e^{x_2 t}.$$

Bestimmen Sie $\mathbf{x} = (x_1, x_2)$, sodass die Funktion f die Datenpunkte im Sinne kleinster Quadrate bestmöglichst ausgleicht.

Lösung: Das Problem kann in Matlab mit der Funktion lsqcurvefit gelöst werden.

```
t = [0:3];
y = [2 0.7 0.3 0.1];
fxt = @(x,t) x(1)*exp(x(2)*t);
x0 = [1,0];
[x,resnorm]=lsqcurvefit(fxt,x0,t,y)
Optimization terminated
successfully:
Relative function value changing
by less than OPTIONS.TolFun
x =
    1.9950 -1.0095
resnorm =
    0.0020
```

Die Lösungsstelle ist somit $x_* \approx (1.995, -1.0095)$. Die Abbildung 74 zeigt

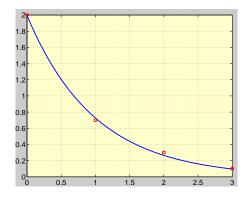


Abbildung 74: Zu Aufgabe 187

die dazugehörige Geometrie und wurde wie folgt erzeugt:

```
tt = linspace(0,3);
ff = x(1)*exp(x(2).*tt);
plot(tt,ff,t,y,'ro'), grid,
```

Für lineare Ausgleichsaufgaben siehe Abschnitt 52.

71. SIMULINK

SIMULINK ist ein signalflussorientiertes (blockorientiertes), interaktives System zur Simulation linearer und nichtlinearer dynamischer Systeme. Es handelt sich um ein mausgesteuertes Grafikprogramm, das ein Modell eines natürlichen, technischen oder wirtschaftlichen Systems, das als Blockdiagramm (Blockschaltbild, Signalflussdiagramm) auf dem Bildschirm darzustellen ist, unter dynamischen

Einwirkungen nachbildet. Es kann für lineare, nichtlineare, zeitkontinuierliche oder zeitdiskrete Prozesse eingesetzt werden. Grundlage sind Matlab-Funktionen zur Lösung gewöhnlicher Differenzialgleichungen (DGL) bzw. Anfangswertaufgaben (AWA). In diesem Sinn kann Simulink als ein numerischer Differenzialgleichungslöser aufgefasst werden. Im zeitdiskreten Fall sind die Modellgleichungen natürlich Differenzengleichungen. Innerhalb der MATLAB-Umgebung ist SIMULINK eine MATLAB-Toolbox, die sich von den anderen Toolboxen eben durch diese spezielle Oberfläche, mit der dann auch eine besondere "Programmiertechnik" verbunden ist, unterscheidet. Variablen können von Simulink nach Matlab problemlos importiert und exportiert werden. Damit stehen zum Beispiel alle Visualisierungsmöglichkeiten von Matlab zur Verfügung, oder Größen des Systems können mathematisch optimiert werden, zum Beispiel mit der Optimization Toolbox.

71.1. Erste Schritte

Der Start des Programms Simulink erfolgt zum Beispiel durch das Kommando simulink im Matlab-command window. Danach öffnet sich der so genannte block Library Browser. Dieser stellt die verfügbaren Blöcke (Funktionsblöcke) der Simulink-Bibliothek in Form einer Liste und in Form von Icons dar. Die Bibliothek ist dabei in Funktionsgruppen organisiert:

- Sources
- Sinks
- Continous

- Discrete
- usw.

Bei der Auswahl eines Listeneintrages (Icon) öffnet sich dann im Simulink Library Browser-Fenster ein Teilfenster mit den zur Funktionsbibliothek gehörenden Funktionssysmbolen. Ein Klick auf Sources etwa öffnet alle Funktionsblöcke zur Erzeugung von Signalen (Funktionen). Zu sehen sind etwa die Blöcke Constant zur Erzeugung eines konstanten Signals oder Sine Wave zur Erzeugung einer Sinusfunktion. Die Funktionsblöcke haben folgende Eigenschaften:

- Name
- Ein- und Ausgabe
- Parameter (Doppelklick)
- Online-Hilfe (Help-Button)
- Kontextmenü (Maus rechts)

SIMULINK ist erweiterbar, das heißt, es können eigene Blöcke mit der gewünschten Funktion definiert werden.

71.2. Konstruktion eines Blockdiagramms

Will man nun mit Hilfe der Bibliotheken ein eigenes System konstruieren, so muss man zunächst im Simulink-Libraray Browser durch Auswahl von File New Model ein leeres Fenster öffnen. Blöcke können nun via Drag&Drop aus der Bibliothek eingefügt werden. Zwei Blöcke können dadurch verbunden werden, dass man mit der linken Maustaste auf einen Blockausgang klickt, die Maustaste festhält und die sich aufbauende Verbindungsli-

nie auf den gewünschten Blockeingang zieht. Am Besten zieht man das Verbindungssignal (Verbindungslinie) immer gegen der Signalflussrichtung vom Eingang des Zielblocks zum Ausgang des Quellblocks. Blöcke und Verbindungen löschen, können Sie mit Hilfe der rechten Maustaste. Blöcke können durch Ziehen an den Ecken auch vergrößert oder verkleinert werden.

Bereits vorhandene Blockdiagramme können mit *File Open* geöffnet und dann bearbeitet werden.

Die Kanten sind den Signalen (oder Größen) des Systems zugeordnet, in den Knoten werden diese Signale generiert oder umgewandelt, also geformt. Da Signale eine bestimmte Flussrichtung haben, sind Blockdiagramme gerichtete Graphen.

71.3. Weitere Arbeitsschritte

Bevor wir mit dem Blockdiagramm jedoch eine Simulation durchführen können, sind noch weitere Arbeitsschritte durchzuführen:

- Parametrierung der verwendeten Blöcke,
- Festlegung des numerischen Lösungsverfahrens (Integrator),
- Angabe der Anfangswerte, Simulationsdauer usw., also allgemeine Parameter der Simulation,
- Dokumentation.

71.4. Ein erstes Beispiel

Wir konstruieren uns nun ein einfaches Beispiel (Modell) eines Blockdiadramms unter Si-

MULINK, führen die zur Simulation notwendigen Arbeitsschritte durch und machen mit diesem Modell ein paar Simulationen.

Es soll ein Quellsignal, die Sinusfunktion $f(t) = 2\sin(t)$, $t \in \mathbb{R}$, (Amplitudenwert 2) integriert werden, und samt ihrer Integralfunktion $F(t) = 2\int_0^t \sin(\tau)d\tau = -2\cos(t) + 2$ von [0, 15] grafisch ausgegeben werden.

Aufgabe 188 Überlegen Sie, warum das Ergebnis der Simulation die Lösung der Anfangswertaufgabe

AWA:
$$\begin{cases} y'(t) = x(t) \\ y(0) = 0 \end{cases} t \in [0, 15]$$

mit y(t) = F(t) und x(t) = f(t) ist.

Lösung: Nennen wir den Ausgang des Integratorblocks y(t), so hat man zwischen Eingangssignal x(t) und Ausgangssignal y(t) die Beziehung

$$y(t) = \int_0^t x(\tau)d\tau + y(0).$$

Durch Differenziation dieser Gleichung erhält man y'(t) = x(t). \odot ... \odot

71.4.1. Konstruktion des Blockdiagramms

Wir öffnen ein neues Fenster und speichern es unter dem Namen ErstesBeispiel.mdl (mdl steht abkürzend für model) ab. In das leere Fenster ziehen wir aus der *Block Library* mit Hilfe der Maus den Block *Sine Wave*; dieser befindet sich in der Bibliothek *Sources*. Gleichermaßen verfahren wir mit den Blöcken *Integrator (Continuouse)*, *Mux (Signal Routing)* und *Scope (Sinks)*. Jetzt werden die Blöcke

verbunden. Der Ausgang von Sine Wave mit dem Eingang von Integrator, der Ausgang von Integrator mit einem Eingang von Mux und der Ausgang von Mux mit dem Eingang von Scope. Schließlich verbinden wir den zweiten Eingang von Mux mit der Verbindungslinie (Signalkante) zwischen Sine Wave und Integrator. Hierzu beginnt man mit dem Eingang von Mux. Bei einer Verbindung "um die Ecke" muss die Maustaste zwischendurch losgelassen werden. Man beachte auch, dass die Verbindung zu einer Signalkante (wie hier zwischen Sine Wave und Integrator) erst dann vollständig ist, wenn an der Kreuzungsstelle ein kleiner rechteckiger Punkt erscheint.

71.4.2. Weitere Arbeitsschritte

Wir beginnen mit der Parametrierung des *Sine Wave*-Blocks. Ein Doppelklick auf den Block öffnet ein Fenster, in dem die Amplitude auf den Wert 2 eingestellt wird. Die anderen Parameter (Frequenz, usw.) bleiben unverändert.

Der Block *Mux* hat zwei Eingänge, was in unserem Beispiel gerade passend ist. Hier ist also nichts zu tun. Prima! Die Zahl der Eingänge kann jedoch im Allgemeinen eingestellt werden.

Der Integratorblock (*Integrator*) verlangt unter anderem den Anfangswert der Integration. Für das Beispiel soll der Wert auf der voreingestellten Null stehen bleiben. Die übrigen Parameter betreffen spezielle Formen von Integratoren und bleiben für unser Beispiel auf den voreingestellten Werten. Der Eintrag 1/s im Integratorblock deutet an, dass es sich um die Laplace-Transformierte der Integration handelt.

Die meisten (linearen) Funktionsblöcke sind mit der Laplace-Transformierten bzw. dem diskreten Pendant dazu, der Z-Transformation, gekennzeichnet.

Der Scope-Block sollte auch parametriert werden. Ein Doppelklick auf den Block und ein Klick aus das Parameters-Icon öffnet ein Karteifenster in dem Parameter eingestellt werden können. Interessant ist die Möglichkeit, das angezeigte Signal direkt in einer Matlab-Variablen speichern zu lassen. Man achte darauf, an dieser Stelle auf das Array-Format umzustellen, wenn man von dieser Möglichkeit des Datenexports in den Matlab-Workspace Gebrauch machen will.

Den letzten Schliff gibt dem Ganzen nun noch die Dokumentation (Beschriftung, Hervorhebung durch Farbe usw.). Durch einen Doppelklick in Modellfenster erscheint ein blickender Cursor. Hier kann frei Text eingetragen werden, der über den Menüpunkt *Format-Font* noch nach Geschmack im Erscheinungsbild verändert werden kann.

Das Ergbenis all unserer Bemühungen sehen Sie in Abbildung 75.

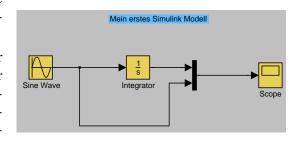


Abbildung 75: Erstes Beispiel

Das eingestellte numerische Lösungsverfahren

ist ode45 (unten im Fenster *ErstesBeispiel*). Wir ändern daran nichts. Wenn Sie doch eine Änderung durchführen wollen, dann unter *Simulation* und weiter unter *Configuration Parameters*. Es würde an dieser Stelle zu weit führen, im Einzelnen auf die numerischen Integrationsverfahren einzugehen. Eine tiefe Diskussion der Verfahren würde den Rahmen dieser Einführung bei weitem sprengen. Es sei jedoch bemerkt, dass die modernsten mathematischen Verfahren zum Einsatz kommen [23, 24], siehe auch Abschnitt 61.

Den Wert der Simulationsdauer setzen wir von 10 auf 15 (zweite Zeile von oben rechts).

71.4.3. Simulation

Nun steht einer Simulation nichts mehr im Wege. Drücken Sie einfach auf das Dreieckssymbol in der zweite Zeile des Modellfensters. Im Scope-Block bauen sich dann das in Abbildung 76 dargestellte Sinussignal f und die zugehörige Integralfunktion F auf.

71.5. Beispiele

In diesem Absschnitt betrachten wir nun weitere Simulink-Modelle. Hierbei handelt es sich um das Lösen von Differenzial- und Differenzengleichungen unter Simulink, oder anders gesagt, um die Simulation dynamischer Systeme

Als erstes Beispiel wählen wir die Anfangswertaufgabe

AWA:
$$\begin{cases} y'(t) = -y(t) - 5e^{-t}\sin(5t) \\ y(0) = 1 \end{cases}$$

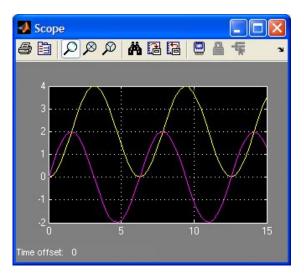


Abbildung 76: Ergebnis

die wir bereits in Abschnitt 62.1 numerisch und in Abschnitt 67 symbolisch behandelt haben. Die Lösung ist

$$y(t) = e^{-t}\cos(5t).$$

Als ersten Schritt löst man die gegebene Differenzialgleichung nach der höchsten Ableitung auf. In diesem Fall ist diese eins und die Gleichung braucht nicht umgeformt zu werden

$$y'(t) = -y(t) - 5e^{-t}\sin(5t).$$

Zur Integration wird der Block *Integrator* (Continous) verwendet. Er sorgt für die numerische Integration der Differenzialgleichung. Danach wird die rechte Seite der Differenzialgleichung mit Hilfe von Integrieren, Additionsstellen, usw. aufgebaut. Im vorliegenden Fall wird mit dem Block fcn (User-Defined Functions) der Ausdruck $-5e^{-t}\sin(5t)$ gebildet

und durch die Summationsstelle *Sum* (Commonly Used Blocks) mit dem Ausgang des Integrierers zusammengefasst. Da die rechte Seite nun komplett ist und diese gleich der ersten Ableitung ist, wird der Ausgang mit dem Integrierereingang verbunden (Rückkopplung). Zur grafischen Darstellung des Signals (Lösung y der Differenzialgleichung) während der Simulation wird der Block *Scope* (Sinks) verwendet. Den Block *Clock* findet man in Source. Das Simulink-Modell zur Lösung der Differenzialgleichung ist in Abbildung 77 dargestellt.

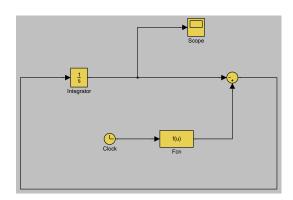


Abbildung 77: Simulink-Modell

Nun setzen wir beim Integrierer die Anfangsbedingungen. Dies geschieht durch Doppelklicken auf das Symbol und dem Eintrag des Anfangwertes, hier 1. Der letzte Schritt besteht in der Wahl der Simulationsparameter. Unter mehreren Integrationsverfahren (das sind gerade die DGL-Löser ode45, usw. von Matlab) muss ein geeignetes aufgewählt werden. Voreingestellt ist ode45. Außerdem ist die Simulationszeit einzustellen. Ich habe das Intervall $[t_0 = 0, t_f = 3]$ gewählt. Das grafische Ergeb-

nis der Simulation, das heißt die Lösung der Differenzialgleichung, ist in Abbildung 60 zu sehen.

Aufgabe 189 (y'(t) = k) Konstruieren Sie eine (signalflussorientierte) SIMULINK-Implementierung der Differenzialgleichung y'(t) = k und simulieren Sie damit. Wählen Sie speziell k = 10 und y(0) = 0 und $0 \le t \le 11$. *Lösung:* Die Abbildung 78 zeigt eine

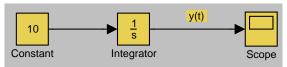


Abbildung 78: y'(t) = k

Simulink-Implementierung der Differenzialgleichung $y'(t) = k. \odot \dots \odot$

Aufgabe 190 (y'(t) = ky(t)) Konstruieren Sie eine (signalflussorientierte) Simulink-Implementierung der Differenzialgleichung y'(t) = ky(t) und simulieren Sie damit. Wählen Sie speziell k = -0.1, y(0) = 1 und $0 \le t \le 30$. Ab wann ist der Zustandswert y(t) kleiner als 0.1? Wie lautet die analytische Lösung? Sie kann gegebenenfalls zu Kontrollzwecken verwendet werden.

Lösung: Die Abbildung 79 zeigt eine SI-MULINK-Implementierung der Differenzialgleichung y'(t) = ky(t). An der grafische Ausgabe kann man erkennt, dass der Zustandswert y(t) für $t \ge 23$ (ungefähr größer) kleiner 0.1 ist. Die analytische Lösung ist $y(t) = e^{-0.1t}$, $t \in [0, 30]$.

Aufgabe 191 $(y_t = ky_{t-1})$ Konstruieren Sie eine (signalflussorientierte) SIMULINK-

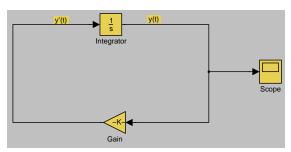


Abbildung 79: y'(t) = ky(t)

Implementierung der Differenzengleichung $y_t = ky_{t-1}$, t = 1, 2, ..., und simulieren Sie damit. Wählen Sie speziell k = -7/10, $y_0 = 1$ und $0 \le t \le 20$.

Lösung: Die Abbildung 80 zeigt eine

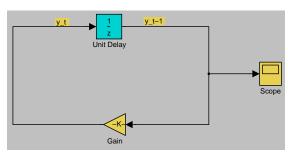


Abbildung 80: $y_t = ky_{t-1}$

SIMULINK-Implementierung der Differenzengleichung $y_t = ky_{t-1}$. Man muss also nur den *Integrator*-Block durch den *Unit Delay*-Block (Discrete) auswechseln. Der Anfangswert $y_0 = 1$ wird in diesem Block eingestellt. Die Abbildung 81 zeigt das Ergebnis der Simulation mit der Parameterwahl k = -7/10, $y_0 = 1$ und $0 \le t \le 20$. \odot . . . \odot

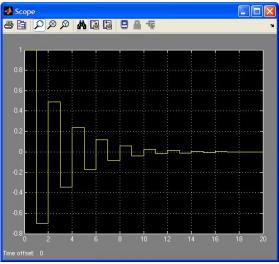


Abbildung 81: Simulation

71.6. Vereinfachungen

Schon für vergleichsweise kleine dynamische Systeme werden relativ schnell viele Blöcke benötigt. Für die gezeigten einfachen Beispiele ist dieser Effekt natürlich in erster Linie darauf zurückzuführen, dass schon für die Operationen der untersten Ebene, wie etwa die Addition oder Skalierung, entsprechende Blöcke eingesetzt wurden. Dies trägt zwar zur Nachvollziehbarkeit der Modellgleichungen innerhalb des Blockdiagramms bei, macht aber dasselbe auch schnell unübersichtlich.

Durch geschickte Verwendung des *Fcn*-Blocks aus der Bibliothek *userDefined Functions* können Simulink-Systeme vereinfacht werden. Mit Hilfe dieses Blocks ist es möglich, ganze Formeln zu einer Einheit zusammenzufassen, so dass auf die Elementarblöcke der untersten Ebene (*Sum* oder *Gain*) verzichtet wer-

den kann.

Eine weitere Möglichkeit zur Vereinfachung von Simulink-Systemen ist eine Zusammenfassung von Teilsystemen zu eigenen Simulink-Blöcken; eine Konstruktion von Subsystemen. Die damit verbundene Hierarchisierung des Problems entspricht der Modularisierung durch Funktionen bei Matlab-Programmen. Eine solche Modularisierung ist für die meisten Probleme der Praxis unumgänglich.

71.7. Kommunikation mit Matlab

Mit Hilfe verschiedenster Blöcke aus der Bibliothek *Sinks* ist es möglich mit MATLAB zu kommunizieren. Als Beispiele seien die Blöcke *Scope* und *To Workspace* erwähnt.

Mit Hilfe der Matlab-Funktion sim kann aus Matlab ein Simulink-Modell aufgerufen werden. Genaue Auskunft über die verschiedenen Formen der Verwendung von sim erhalten sie mit doc sim (help sim).

Eine weiter Kommunikationsart besteht über das Konzept globaler Variablen.

71.8. Umgang mit Kennlinien

In den Anwendungen kann ein funktionaler Zusammenhang zwischen vorkommenden Größen meist nicht explizit angegeben werden, das heißt eine Formel oder Funktionsvorschrift steht nicht zur Verfügung. Vielmehr stehen gemessene oder beobachte Werte tabellarisch zur Verfügung. Eine solche Tabelle nennt man eine Kennlinie. In der Bibliothel *Lookup Tables* stehen Blöcke zur Verfügung mit denen man Kenntlinien behandeln kann (Interpolation, Extrapolation).

71.9. Weitere Bemerkungen und Hinweise

Weitere Informationen und Details über SIMU-LINK finden Sie in [25].

72. Dünn besetzte Matrizen

Rechnet man mit einer Matrix, so geht Mat-LAB davon aus, dass diese dicht besetzt ist, das heißt, nur wenige Matrixelemente sind Null. Sind in einer Matrix jedoch viele Elemente Null (zum Beispiel 95% oder mehr) so spricht man von einer Sparsematrix oder von einer dünn besetzten Matrix. Liegt solch eine Sparsematrix vor und möchte man Matrixoperationen mit dieser durchführen, so kann man meistens Gleitpunktoperationen auf den Nullelementen einsparen. Darüber hinaus muss man nicht die ganze Matrix abspeichern, sondern es genügt, nur die von Null verschiedenen Elemente abzulegen. Trägt man diesen Sachverhalten Rechnung, so lässt sich die Effizienz von Algorithmen in bezug auf Speicher- und Zeitaufwand reduzieren. Damit ist es möglich, größere Probleme zu lösen, die sonst nicht lösbar wären. Matlab kann die dünne Besetztheit einer Matrix ausnutzen.

Matlab verfügt über zwei Speichermodi: full und sparse, wobei full standardmäßig eingestellt ist. Die Funktionen full und sparse erlauben es, zwischen beiden Modi hin und her

zu schalten.

Im Sparsemodus werden die von Null verschiedenen Matrixelemente als eindimensionales (lineares) Feld mit ihren Zeilen- und Spaltenindizes abgespeichert. Die Anweisung

```
>> F = -triu(tril(ones(6),1),-1)
+3*eye(6)
```

erzeugt die (6,6)-Matrix

$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	F =					
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	2	-1	0	0	0	0
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	-1	2	-1	0	0	0
0 0 0 -1 2 -1	0	-1	2			0
	0	0	-1	2	-1	0
	0	0	0	-1	2	-1
0 0 0 0 -1 2	0	0	0	0	-1	2

Mit S = sparse(F) wird F im Sparsemodus abgespeichert:

8-1		
S =		
(1,1)	2	
(2,1)	-1	
(1,2)	-1	
(2,2)	2	
(3,2)	-1	
(2,3)	-1	
(3,3)	2	
(4,3)	-1	
(3,4)	-1	
(4,4)	2	
(5,4)	-1	
(4,5)	-1	
(5,5)	2	
(6,5)	-1	
(5,6)	-1	
(6,6)	2	

Die von Null verschiedenen Elemente werden zusammen mit ihren Indizes spaltenweise angeordnet. Mit

```
>> F = full(S)
```

lässt sich dies rückgängig machen. Der Funktionsaufruf nnz(F) zeigt Ihnen die Anzahl der von Null verschiedenen Elemente der Matrix F.

72.1. Sparsematrizen erzeugen

Sparsematrizen können auch direkt erzeugt werden. Mit Hilfe der Funktion spdiags lässt sich zum Beispiel eine dünn besetzte Bandmatrix erzeugen. Obige Bandmatrix F kann somit direkt wie folgt erstellt werden:

```
>> m = 6; n = 6;
>> e = ones(n,1); d = 2*e;
>> A = spdiags([-e,d,-e],...
[-1,0,1],m,n);
```

Der Vektor [-1,0,1] gibt an, in welcher Diagonalen die Spalten von [-e,d,-e] stehen. Führen Sie die Anweisungen aus und sehen Sie sich die volle Matrix mit full(A) an. Alternativ dazu können Sie sich mit spy(A) die Struktur der Matrix A grafisch betrachten. Die zu eye, zeros, ones, rand und randn analogen Sparsefunktionen sind: speye, sparse, spones, sprand, und sprandn.

72.2. Mit Sparsematrizen rechnen

Unabhängig vom Speichermodus können Sie die arithmetischen Operationen und viele Funktionen verwenden. Welchen Speichermodus haben dann die Ergebnismatrizen? Operationen mit vollen Matrizen ergeben volle Matrixresultate. Ist S eine Sparsematrix und F eine vollbesetzte Matrix, so ist:

```
Sparse: S+S, S*S, S.*S, S.*F, S^n, S.^n, S\S
```

Sparse: inv(S), chol(S), lu(S),

diag(S), max(S), sum(S)

Full: S+F, S*F, $S\setminus F$, $F\setminus S$

Ist a ein Skalar, so sind die Ergebnisse von a*S und a\S vom Sparsemodus.

Um Eigenwerte oder singuläre Werte einer Sparsematrix S zu berechnen, muss man S in den vollen Speichermodus konvertieren und die Funktionen eig bzw. svd verwenden: eig(full(S)) bzw. svd(full(S)). Will man nur einige Eigenwerte oder singuläre Werte berechnen, so kann man die Funktionen eigs und svds benutzen. Diese akzeptieren Sparsematrizen als Eingabeargument: eigs(S) bzw. svds(S).

Bildet man eine Blockmatrix und ist eine Untermatrix eine Sparsematrix, so ist auch die Blockmatrix vom Speichermodus sparse.

Weitere Informationen erhalten Sie mit doc sparfun (help sparfun).

Aufgabe 192 (Sparsematrizen) Erzeugen Sie eine (100, 100)-Sparsematrix *A*, die in der Diagonalen lauter 4-en hat und die obere und untere Nebendiagonale besteht aus lauter Zahlen -1. Plotten Sie die Sparsematrix *A* und bestimmen Sie die Anzahl der von Null verschiedenen Elementen.

Lösung: Man definiert zunächst

```
>> n = 100;
```

```
>> e = ones(n,1);
>> d = 4*e;
```

Dann erzeugt man die Matrix A mittels

```
A = spdiags([-e,d,e],[-1,0,1],n,n);
```

Mit spy(A) kann ein Plot der Matrix A erzeugt werden, siehe Abbildung 82. In der Grafik er-

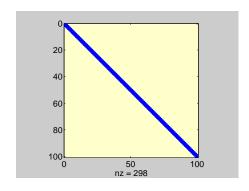


Abbildung 82: Sparsematrix

scheint die Anzeige nz = 298 (nz: **n**on **z**eros). Dies gibt die Anzahl der von Null verschiedenen Elemente an. Diese Information kann man auch mittels nz = length(nonzeros(A)) erhalten. \odot \odot

Aufgabe 193 (Sparsematrizen) Wie sieht die Matrix S nach den folgenden Anweisungen aus? Überlegen Sie zuerst und überprüfen Sie danach Ihre Antwort mit full(S).

Lösung: Die Matrix S hat die folgende Form:

>> f	ull(S)				
ans	=				
2	-1	0	0	0	0
-1	2	-1	0	0	0
0	-1	2	-1	0	0
0	0	-1	2	-1	0
0	0	0	-1	2	-1
0	0	0	0	-1	2

Aufgabe 194 (Sparsematrizen, fill-in) In vielen Fällen, zum Beispiel bei der Diskretisierung von partiellen Differenzialgleichungen mit Differenzenverfahren und nach der Methode der finiten Elemente, entstehen Matrizen, deren n^2 Elemente nicht in den Hauptspeicher passen, sehr viele Nullelemente haben, aber ihre nichtverschwindenden Elemente sehr wohl in den Arbeitsspeicher passen, wenn man spezielle Speichertechniken verwendet. Im allgemeinen sind dann Eliminationsverfahren (LU- bzw. QR-Zerlegungen) nicht anwendbar, weil in Ihrem Verlauf Nichtnullemente an vorher noch nicht belegten Positionen entstehen können. Dieses Phänomen der Erzeugung von Nichtnullelementen bezeichnet man als fill-in. Das folgende Beispiel zeigt solch ein fill-in. Definieren Sie hierzu die Matrix A durch die Anweisung A =gallery('redheff', 20). Schauen Sie sich mit help private/redheff an, wie die Matrix definiert ist. Mit Hilfe der Funktion 1u enthält man in Matlab eine LU-Zerlegung (Siehe zum Beispiel [7]). Plotten Sie sodann die Matrizen A, L und U mit der Funktion spy.

Lösung: Die LU-Zerlegung der Matrix A er-

hält man mit

Die grafische Darstellung der Matrizen A, L und U (Siehe Abbildung 83 und 84) erzeugt

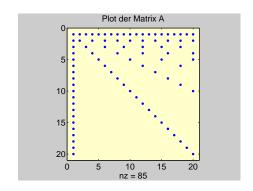


Abbildung 83: Plot der Matrix A

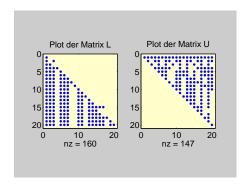


Abbildung 84: Plot der Matrizen L und U

man mit

```
>> figure
>> plot(3,1,1)
>> spy(A)
>> title('Plot der Matrix A')
>> subplot(1,2,1)
>> spy(L)
```

```
>> title('Plot der Matrix L')
>> subplot(1,2,2)
>> spy(U)
>> title('Plot der Matrix U')
```

73. Mehrdimensionale Arrays

Arrays der Datentypen (Abschnitt 74) double, char, cell und struct können mehr als zwei Dimensionen haben. Solche mehrdimensionale Arrays werden in natürlicher Verallgemeinerung zu Techniken, wie man sie für Matrizen kennt, definiert und manipuliert.

Mit den Funktionen zeros, rand und randn können mehrdimensionale Arrays erzeugt werden. Zum Beispiel wird durch

ein (3, 4, 5)-Array mit insgesamt $3 \cdot 4 \cdot 5 = 60$ normalverteilten Zufallszahlen erzeugt.

Ein dreidimensionales Array könnte zum Beispiel dreidimensionale physikalische Daten darstellen, etwa die Temperatur in einem Raum über den Punkten eines rechteckigen Gitters. Es könnte auch eine Folge von Matrizen A_k repräsentieren bzw. Abtastwerte einer zeitabhängigen Matrix A(t) bezeichnen. Das (i, j)-te Element der k-ten Matrix ist dann $A(i, j, k) = a_{ijk}$. Mit Hilfe der cat-Funktion kann man ein mehrdimensionales Array durch Angabe der Elemente in einer Anweisung definieren. Das erste Argument gibt die Dimension des Arrays. Die folgende Anweisung erzeugt ein dreidimensionales Array der Größe $3 \times 2 \times 2$.

Funktionen, die elementweise arbeiten, können auch auf mehrdimensionale Arrays angewendet werden, zum Beispiel arithmetische, logische und relationale Operatoren. zum Beispiel geben die Ausdrücke A.*A, exp(A), 3.^A und A>0 die erwarteten Ergebnisse. Der (konjugiert) tranponierte Operator und die Funktionen diag, inv, eig und \ aus der Linearen Algebra sind für Arrays der Dimension größer als zwei nicht definiert, also nur für Matrizen (zweidimensionale Arrays).

Die Tabelle 38 gibt weitere Matlab-Funktionen an, die speziell zum Manipulieren von mehrdimensionalen Arrays geeignet sind.

Funktion	Bedeutung
cat	Verkettet Arrays
ndims	Größe, Ordnung
ndgrid	Erzeugt Arrays
permute	Permutiert Dimension
ipermute	Inverse Permutation
shiftdim	Verschiebt Dimension
squeeze	Verschiebt Dimension

Tabelle 38: Array-Funktionen

Aufgabe 195 (Mehrdim. Arrays) Verwenden Sie dreidimensionale Arrays, um die folgende symmetrische Matrix für verschiedene reelle Parameter t zu studieren:

$$A_t = \left[\begin{array}{ccccc} t & -1 & 0 & 0 & 0 \\ -1 & t & -1 & 0 & 0 \\ 0 & -1 & t & -1 & 0 \\ 0 & 0 & -1 & t & -1 \\ 0 & 0 & 0 & -1 & t \end{array} \right].$$

- (a) Brechnen Sie die Inverse von A_2 und von
- (b) Versuchen Sie die Inverse von A_0 zu berechnen.
- (c) Lassen Sie den Parameter t von 2 an langsam immer kleiner werden. Berechne die Inversen von A_t und gib den ersten Wert an, wo A_t singulär wird.

Die $n \times n$ -Matrix A_t ist beim numerischen Lösen von Differenzialgleichungen von Bedeutung, insbesondere für große n.

Lösung: Zunächst definieren wir die Matrix A0 für A_0 .

Nun definieren wir für verschiedene Parameterwerte den Vektor T. Zum Beispiel:

$$T = [0 \ 1 \ 2 \ 3.7 \ 5];$$

Dann erzeugen wir A_t durch:

(a) Die Inverse der Matrix A_5 erhält man durch inv(A(:,:,5)).

- (b) Die Inverse der Matrix A_2 ist durch inv(A(:,:,3)) zu berechnen.
- (c) Der Befehl inv(A(:,:,1)) zeigt, dass die Matrix A_0 singulär ist.
- (d) Der erste Parameterwert, für den die Matrix A_t singulär wird, ist t = 1.

74. Datentypen (Klassen)

Bei der Realisierung numerischer und hauptsächlich nichtnumerischer Algorithmen sind Datentypen von besonderer Bedeutung. Spricht man von einfachen Datentypen, so meint man ganze Zahlen, Gleitpunktzahlen, Zeichenketten usw. Unter einem zusammengesetzten Datentyp oder einer Datenstruktur versteht man ein Array (Feld) oder einen Verbund. Datenstrukturen können statisch und dynamisch sein. Ein statischer Datentyp ist zum Beispiel ein Array und eine verkettete Liste ist ein dynamischer Datentyp. Durch die Definition von Datentypen werden die möglichen Operationen mit denen Daten und die Größe des Speicherplatzes festgelegt. In Matlab wird stets der Begriff Datentyp oder Klasse verwendet und man meint damit zum Beispiel einen der folgenden Datentypen:

- double (numerische Arrays)
- sym (Symbolisches Objekt)
- char (Zeichenketten)
- cell (Zellenarrays)
- struct (Strukturarrays)

Bisher haben wir hauptsächlich mit dem Datentyp double und sym (aus der *Symbolic Toolbox*) gearbeitet; gelegentlich auch mit der Klasse function_handle.

Um festzustellen von welchem Datentyp ein Objekt ist, kann man die Funktion class verwenden:

```
>> class(pi)
ans =
double
>> f = @(x) x^2;
>> class(f)
ans =
function_handle
```

Auch mit der Funktion isa kann man den Datentyp einer Variablen herausfinden.

Mit den folgenden Anweisungen werden die Variablen A bis N definiert, die alle einen anderen Datentyp besitzen. Den Datentyp können Sie aus dem MATLAB-Workspace ablesen.

```
>> A = 'Z';
>> B = magic(3);
>> C = @(x) x^2;
>> D = cell(3,2);
>> E.field = 1;
>> F = int8(123);
>> G = uint8(123);
>> H = int16(123);
```

```
>> I = uint16(123);
>> J = int32(123);
>> K = uint32(123);
>> L = single(123);
>> M = sym(123);
>> N = ss(1,1,1,1);
```

Die vorletzte Anweisung M = sym(123) ist mit der *Symbolic Math Toolbox* und die letzte Anweisung N = ss(1,1,1,1) mit *Control Toolbox* möglich. Erstere erzeugt eine symbolische Variable und Letztere ein Zustandsraummodell. Weitere Matlab-Datentypen werden von Toolboxen erzeugt. Für weitere Informationen siehe auch doc datatypes (help datatypes). Die drei Datentypen char, cell und struct wollen wir etwas genauer anschauen.

74.1. Zeichenketten (char)

Zeichenketten (Strings) werden durch Hochkommata begrenzt

```
>> s = 'Ich bin eine Zeichenkette'
s =
Ich bin eine Zeichenkette
```

Die Variable s ist ein Array der Größe (1,25) vom Typ char. Mit dieser Variablen können Sie nun mit allen erlaubten Arrayoperationen manipulieren. Sie können zum Beispiel Zeichenketten zusammenfügen oder trennen. Mit den Funktionen str2num oder num2str können Sie Zeichen in Zahlen konvertieren oder umgekehrt.

Für weitere Informationen siehe doc strfun (help strfun).

Aufgabe 196 (Zeichenketten) Die druckbaren Zeichen des Ascıı-Zeichensatzes werden durch die ganzen Zahlen 32:127 dargestellt. Zahlen kleiner als 32 entsprechen nicht druckbaren Steuerzeichen. Geben Sie diese druckbaren Zeichen auf dem Bildschirm aus. Pro Zeile sollen 16 Zeichen ausgegeben werden. Welcher Zahl entspricht das @-Zeichen, das in jeder E-Mail-Adresse vorkommt?

Lösung: Zunächst definiert man eine (6, 16)-Matrix Z, die die Werte 32 bis 127 enthält.

```
N = 32:127;
Z = reshape(N,16,6)';
```

Dann kann man diese mit Hilfe des Befehls char in den ASCII-Zeichensatz umwandeln. Nach der Eingabe

```
char(Z)
```

sieht die Ausgabe wie folgt aus:

```
ans =
!"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
'abcdefghijklmno
pqrstuvwxyz{|}~
```

Um herauszufinden welcher Zahl ein Zeichen entspricht kann man Befehl double verwenden. double ('@') liefert dann die Zahl 64. ⊚

74.2. Zellen- und Strukturenarrays

Die Datentypen cell und struct erlauben es, unterschiedliche aber logisch zusammengehörige Daten (Felder) zu einer Variablen zusammenzufassen. Zum Beispiel können Zeichenketten und numerische Felder unterschiedlicher Größe in einer Zelle gespeichert werden. Mathematische Operationen sind darauf aber nicht definiert. Hierzu muss man gegebenenfalls auf den Inhalt zugreifen. Strukturen und Zellen sind ähnlich. Ein Unterschied besteht jedoch darin, dass Strukturen durch Namen und nicht durch Zahlen identifiziert werden.

Strukturenarrays werden innerhalb von Mat-LAB an verschiedenen Stellen verwendet, zum Beispiel von den Funktionen spline, solve und von Optimierungs- und Differenzialgleichungslösern. Außerdem spielen sie beim objektorientierten Programmieren in Matlab eine große Rolle (Darauf sind wir bisher nicht eingegangen). Zellenarrays finden zum Beispiel in den Funktionen varargin und varargout Verwendung, sowie in Grafikkommandos, um Text zu spezifizieren.

Eine Struktur ist ein Datentyp, welcher verschiedene Werte von möglicherweise verschiedenen Datentypen beinhaltet. Eine MATLAB-Struktur ist mit dem Datentyp RECORD in der Sprache PASCAL vergleichbar. Im einfachsten Fall lässt sich eine Struktur durch einfache Anweisungen erzeugen. Die folgenden Anweisungen erzeugen die Variable M vom Datentyp struct. M ist ein (1, 1)-Array.

```
n = 2;
M.Name = 'Hilbert';
M.Matrix = hilb(n);
M.Eigenwerte = eig(sym(hilb(n)));
```

Gibt man M ein, so erhält man die Feldnamen, aber nicht deren Inhalt:

```
>> M
M =
          Name: 'Hilbert'
          Matrix: [2x2 double]
          Eigenwerte: [2x1 sym]
```

Das erste Feld M. Name ist vom Typ char, das zweite vom Typ double und das dritte vom Typ sym. Den Inhalt des zweiten Feldes erhält man zum Beispiel durch

```
>> M.Matrix
ans =
1.0000 0.5000
0.5000 0.3333
```

Ein Zellenarray ist ein Matlab-Array, das als Elemente Matlab-Arrays haben kann. Ein Element heißt Zelle oder Zellenelement. Im Gegensatz zu Strukturenarrays werden Zellenarrays nicht durch Namen, sondern durch Zahlen identifiziert. Ein Weg, um ein Zellenarray zu definieren besteht darin, geschweifte Klammern zu verwenden. Die folgende Anweisung erzeugt das (2, 2)-Zellenarray Z und gibt Informationen über die einzelnen Zellen am Bildschirm aus.

```
>> Z = {sym(2), pi; pascal(3),...
'ABC'}
Z =
[1x1 sym ] [3.1416]
[3x3 double] 'ABC'
```

Ein Zellenarray erlaubt es somit, Arrays verschiedener Datentypen zu einem Objekt zusammenzufassen. Im obigen Beispiel besteht das Zellenarray Z aus Arrays des Typs double und char. Sind die Zellen nicht zu groß, so werden sie am Bildschirm angezeigt. Lässt es der Raum aber nicht zu, so erscheint nur die Größe der jeweiligen Zelle. Eine Zelle kann auch eine mehrdimensionale Struktur sein. Die Inhalte von einzelnen Zellen sind zur Ausgabe mit Indizes in geschweiften Klammern anzugeben. Beispiel:

Mit der Funktion cellplot kann man sich die grafische Struktur eines Zellenarrays in einem Grafikfenster anschauen. Der Aufruf

```
cellplot(A)
```

erzeugt die Abbildung 85.

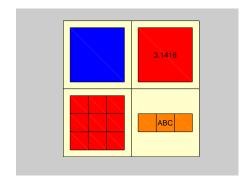


Abbildung 85: Die cellplot-Funktion

Aufgabe 197 (Zellenarrays) Speichern Sie die

Blockmatrix

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ \hline 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{bmatrix},$$

durch Indexadressierung in einem (2, 2)-Zellenarray A ab.

Lösung: Die entsprechenden Zeilen lauten:

Aufgabe 198 (Zellenarrays) Speichern Sie im (2, 2)-Zellenarray Z orthonormale Basen für alle vier fundamentalen Unterräume von

$$A = \left[\begin{array}{rrr} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{array} \right].$$

In der (1,1)-Zelle soll eine orthonormale Basis von Bild (A^T) stehen, in der (1,2)-Zelle eine Basis von Null(A), in der (2,1)-Zelle von Bild(A) und in der verbleibenden Zelle eine orthonormale Basis von Null $(A)^T$.

Lösung: Die Singulärwertzerlegung

$$A = USV^{\mathrm{T}}$$

liefert orthonormale Basen für alle vier fundamentalen Unterräume von A.

```
>> A = [1 -1 1;1 0 0;1 1 1;1 2 4];
>> [U,S,V] = svd(A);
>> Z = cell(2,2);
```

```
>> Z(1,1) = {V};
>> Z(1,2) = {[]};
>> Z(2,1) = {U(:,1:3)};
>> Z(2,2) = {U(:,4)};
```

Alternativ kann eine Lösung wie folgt aussehen:

```
>> Z{1,1} = orth(A');
>> Z{1,2} = null(A);
>> Z{2,1} = orth(A);
>> Z{2,2} = null(A');
```

75. Audiosignale (Töne, Musik)

Für die folgende Beispiele mit Audiosignalen ist eine funktionsfähige "PC Sound Card" erforderlich.

Analoge Signale, also wert- und zeitkontinuierliche Signale, können mit Hilfe eines Analog-Digital-Umsetzers (A/D-Umsetzer) durch Abtastung und Quantisierung in ein digitales (wert- und zeitdiskretes) Signal überführt werden. Umgekehrt lassen sich aus digitalen Signalen mit einem Digital-Analog-Umsetzer (D/A-Umsetzer) analoge Signale erzeugen. Wichtige Parameter dabei sind die Abtastfrequenz f_S , das heißt die Häufigkeit der Abtastungen pro Sekunde, und die Wortlänge w, das heißt die Zahlendarstellung der Amplituden des digitalen Signals.

Eine moderne "PC Sound Card" besitzt A/Dund D/A-Umsetzer mit einer typischerweise von 5 bis 44.1 kHZ einstellbaren Abtastfrequenz und einer Wortlänge von 8 oder 16 Bit. Sie erreicht dann theoretisch eine Hörqualität vergleichbar zur Audio-CD.

Der folgende Script läd die Datei handel in den Workspace. Darin befindet sich das Audiosignal y für das *Hallelujah aus* HÄNDELS *Messias*. Mit der Funktion soundsc kann das Signal abgespielt werden. Die Abspieldauer beträgt 8.9 s. Die Abtastfrequenz Fs ist 8192 Hz.

```
>> load handel
>> soundsc(y,Fs)
```

Weiter Tonbeispiele (Zugpfeife, Gong, Gelächter, usw.) finden Sie unter doc audiovideo (help audiovideo).

Am PC mit Betriebssystem Windows liegen Audiosignale oft in Dateien im wav-Format vor. Matlab kann derartige Dateien lesen und schreiben, sowie digitale Signale direkt an die "PC Sound Card" ausgeben. Siehe doc audiovideo (help audiovideo).

Aufgabe 199 (Audiosignale) Suchen Sie auf Ihrem PC eine wav-Datei und laden Sie sie mit der Funktion wavread. Bestimmen Sie die Abtastfrequenz und die Wortlänge und geben Sie das Signal grafisch aus.

Lösung: Hier mein Beispiel, siehe Abbildung 86. Dazu habe ich folgende Skript verwendet.

```
>> [y,fs] = wavread('Beispielwav');
>> soundview(y,fs)
```

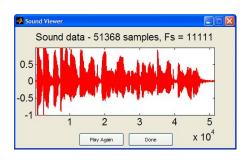


Abbildung 86: wav-Datei

76. WWW-Seiten

Die Firma MathWorks kann über das World Wide Web (WWW) erreicht werden. Die URL-Adresse³ lautet: http://www.mathworks.com Der Deutsche Web Mirror (Spiegel) ist http://www.mathworks.de Von hier aus findet man verschiedene Informationen über Matlab und Simulink sowie deren Toolboxen. Außerdem gibt es Hinweise über Blocksets, eine Liste von Büchern über Matlab sowie m-Files von anderen Benutzern. Auch Informationen zur Studentenversion sind dort zu haben. Es empfiehlt sich, hin und wieder die Homepage aufzusuchen, da dort die aktuellen Informationen über Matlab zu finden sind.

Um sich m-Files von der User Community zu besorgen, steht die Adresse http://www.mathworks.de/matlabcentral zur Verfügung.

Programmiertipps stehen unter http: //www.mathworks.de/access/helpdesk/ help/techdoc/matlab_prog. In dieser Beschreibung findet Sie auf der letzten Seite weitere Adressen, um weitere Infos über Rea-

³URL: Uniform Resource Locator.

lease Notes, Function Reference, Newsgroup-Adressen, Suchadressen nach Online-Quellen, Digest-Infos, News & Notes, Dokumentationen (Handbücher), Beispielsammlung, usw. Übrigens, diese Programmiertips (mit den Adressen auf der letzten Seite) werden auch als PDF-File mit Matlab mitgeliefert. Über das Menü help in Matlab help unter Printable Dokumentation (PDF) kommen Sie an das PDF-Dokument.

Ein FAQ-Text (FAQ = Frequently Asked Questions) zu MATLAB finden Sie unter http://www.mit.edu/~pwb/cssm.

77. Das Matlab-Logo

Das Matlab-Logo (Abbildung 87) ist eine Darstellung des ersten Schwingungsmodus eines dünnen, fest in einen L-förmigen Rahmen eingespannten Tuches⁴, dessen Schwingungsverhalten durch eine Wellengleichung beschrieben wird. Zur Geschichte des Matlab-Logos siehe http://www.mathworks.com/company/newsletters/news_notes/clevescorner/win03_cleve.html. Das Matlab-Logo erhält man durch das Kommando logo (doc logo).

78. Studentenversion

Eine Studentenversion von Matlab erhalten Sie unter https://www.academic-center.de/cgi-bin/home.

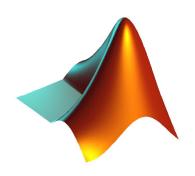


Abbildung 87: Das Matlab-Logo

79. Cleve's Corner

Besonders interessant finde ich *Cleve's Corner*. Darin befinden sich interessante Artikel rund um MATLAB von CLEVE MOLER, siehe www.mathworks.com/company/newsletters/news_notes/clevescorner/index.html.

80. Handbücher

Die Handbücher [14], [15], [16] und [17] sind als PDF-Files unter http://www.mathworks.de/access/helpdesk/help/helpdesk.html erhältlich.

81. Progammiertips

Tips zur effizienten Programmierung in Matlab finden Sie unter http://www.mathworks.de/access/helpdesk/help/pdf_doc/matlab/programming_tips.pdf.

⁴etwa wie ein Tambourin (eine längliche, zylindrische Trommel, die mit zwei Fellen bespannt ist).

82. Literatur

Bücher zu und über Matlab findet man unter http://www.mathworks.de/support/books. Darunter auch das sehr empfehlenswerte Buch von Cleve Moler, siehe [19].

83. Ähnliche Systeme

Diejenigen, die sich Matlab nicht leisten können, sei auf die Public Domain Software Scilab (http://www-rocq. inria.fr/scilab/), **OCTAVE** (http: //www.gnu.org/software/octave/, http://www.che.wisc.edu/octave), (http://rlab.sourceforge.net) und **EULER** (http://mathsrv. ku-eichstaett.de/MGF/homes/ grothmann/euler/german/index.html) verwiesen. Diese Systeme sind bezüglich Syntax und Kapazität dem Matlab-System ähnlich. Andere kommerzielle Systeme sind zum Beispiel Gauss, HiQ, IDL, Mathcad und PV-WAVE.

Andere interaktive Systeme, die ihre Stärken hauptsächlich im Bereich symbolischer Rechnungen haben, sind: Axiom, Derive, Macsyma, Maple, Mathematica, MuPAD und Reduce. Man nennt diese Systeme *Computeralgebra-Systeme* (*CA-Systeme*).

A. Glossar

- **Array (Feld).** Unter einem Array (Feld) versteht man eine Reihe (Ansammlung) von Daten eines bestimmten Typs. Vektoren und Matrizen sind die bekanntesten Beispiele.
- **Array Editor.** Ein Tool, das es erlaubt, den Inhalt von Arrays anzuzeigen und zu verändern.
- Class (Klasse oder Datentyp). Ein Datentyp in Matlab.
- Command History (Kommando-Historie).

 Ein Tool, das früher eingebaute MatLAB Kommandos anzeigt, sowohl für
 die gegenwärtige als auch für frühere
 Sitzungen.
- Command Window (Kommando-Fenster).

 Das Fenster, in dem Matlab den Prompt

 » anzeigt und man Kommandos eingeben kann. Es ist Teil der Matlab

 Arbeitsoberfläche.
- Current Directory Browser. Aktueller Verzeichnis-Browser. Ein Browser, in dem m-Files und andere Files angezeigt werden. Es können auch Operationen angewendet werden.
- **Editor/Debugger.** Ein Tool zum Erzeugen, Editieren und zur Fehlersuche von Files.
- FIG-file. Ein File mit der Endung .fig, der eine Matlab-Figur speichert und in Matlab eingeladen werden kann.
- **Figure.** Ein Matlab-Fenster zur Anzeige von Grafik.
- Function M-File. Ein Typ von m-File, der

- Ein- und Ausgabeargumente akzeptiert. Variablen sind dort lokal definiert.
- Handle Graphics. Ein objekt-orientiertes Grafiksystem, dem die Grafik von Mat-LAB unterliegt. Objekte sind hierachisch organisiert und werden durch Handles manipuliert.
- **Help Browser.** Ein Browser, mit dem man die Dokumentation von Matlab und anderen *MathWorks* Produkten anschauen und suchen kann.
- IEEE arithmetic (IEEE-Arithmetik). Ein Standard-Gleitpunktsystem, das in Matlab realisiert ist.
- LAPACK. Eine Bibliothek von FORTRAN
 77 Programmen zur Lösung linearer
 Gleichungen, Ausgleichsaufgaben,
 Eigenwert- und Singulärwertberechnnungen. Viele MATLAB-Funktionen zur
 linearen Algebra basieren auf LAPACK.
- **Launchpad.** Ein Fenster für den Zugang zu Tools, Demos und Dokumentationen von *MathWorks* Produkten.
- **M-File.** Ein File mit der Endung .m, der Mat-LAB Kommandos beinhaltet. Ein m-File ist entweder ein Function oder Script-File.
- MAT-File. Ein File mit der Endung .mat, der MATLAB Variablen beinhaltet. Es wird mit den Kommandos save und load erzeugt bzw. eingeladen.
- Matlab desktop (Arbeitsoberfläche). Eine Benutzer-Arbeitsoberfläche, um Files, Tools und Anwendungen mit Matlab zu bearbeiten.
- MEX-File. Ein Unterprogramm mit C oder

- FORTRAN-Code, das plattformabgängige Endungen hat. Es verhält sich wie ein m-File oder eine eingebaute Funktion.
- **Script M-File.** Ein Typ von m-File, das kein Ein- und Ausgabeargument hat und auf Daten im Workspace (Arbeitsspeicher) operiert.
- **Toolbox.** Eine Sammlung von m-Files, die den Funktionsumfang von Matlab erweitert, gewöhnlich im Hinblick auf ein spezielles Anwendungsfeld.
- Workspace. Arbeitsspeicher, der über die Matlab-Befehlszeilen erreichbar ist. Beim Beenden werden die Variablen gelöscht.
- **Workspace Browser.** Ein Browser, der alle Variablen im Workspace auflistet und Operationen auf diesen erlaubt.

B. Die Top Matlab-Funktionen

In diesem Anhang liste ich Matlab-Funktionen auf, von denen ich glaube, dass sie vom typischen Matlab-User am Häufigsten eingesetzt werden. Informationen über diese Funktionen erhalten Sie mit den Online-Hilfen von Matlab, insbesondere mit doc und help. Siehe auch die Abschnitte 17 und 18.

Vektoren/Matrizen	
zeros	Nullmatrix
ones	Einsmatrix
eye	Einheitsmatrix
rand	Zufallsmatrix
randn	Zufallsmatrix
linspace	Gleicher Abstand

Spezielle Variablen/Konstanten	
ans	Nullarray
eps	Maschinengenauigkeit
i	Imaginäre Einheit
j	Imaginäre Einheit
inf	∞
NaN	Not a Number
pi	Kreiszahl π

Logische Operatoren	
all	Test auf Nichtnullen
any	Test für ein Nichtnullelement
find	Findet Indizes
isempty	Test auf leeres Array
isequal	Test auf Gleichheit

Arrays	
size	Arraygröße
length	Vektorlänge
reshape	Ändert Form
:	Doppelpunkt
end	Letzter Index
diag	Diagonalmatrizen
tril	Dreiecksmatrizen
triu	Dreiecksmatrizen
repmat	Blockmatrix

Kontrollstrukturen	
error	Fehlermeldung
for	For-Schleife
if	If-Abfrage
switch, case	Fallunterscheidung
while	Wiederholungen

Datenanalysis	
max	Maximum
min	Minimum
mean	arithm. Mittelwert
std	Standardabweichung
sum	Summe
prod	Produkt
sort	Sortieren

Lineare Algebra	
norm	Norm
cond	Kondition
\	Löst LGS
eig	Eigensysteme
lu	LU-Faktorisierung
qr	QR-Faktorisierung
svd	Singulärwertzerlegung

m-Files	
edit	Editor
lookfor	Suche
nargin	Anzahl Input-Argumente
nargout	Anzahl Output-Argumente
type	Listet File
which	Pfadname

Gemischtes	
clc	Löscht
demo	Demonstrationen
diary	Aufzeichnungen
dir	Verzeichnis
doc	Hilfe
help	Hilfe
tic, toc	Zeitmessung
what	Listet Files

Datentypen	
double	Double Precision
char	Zeichen
cell	Zelle
num2str	Zahl2Zeichen
sparse	Sparsematrix
struct	Struktur

Workspace	
clear	Löscht
who	Listet Variablen
load	Läd Variablen
save	Speichert
exit, quit	Beendet

Ein- und Ausgabe	
disp	Zeigt Text oder Array
fprintf	Schreibt Daten
sprintf	Schreibt Daten
input	Prompt für Eingabe

Numerische Methoden		
bvp4c	Randwertaufgabe	
fft	FFT	
fminbnd	Minimierung	
interp1	Interpolation	
ode45	Löst ODE	
polyfit	Polynomfit	
quadl	Numerische Integration	
roots	Nullstellen	
spline	Splines	

Grafik		
plot	x, y-Plot	
fplot	Funktionsplot	
ezplot	Funktionsplot	
semilog	Halblogarithmisch	
bar	Barplot	
hist	Histogramm	
axis	Axen	
xlim	<i>x</i> -Achse	
ylim	y-Achse	
grid	Gitter	
xlabel	x-Label	
ylabel	y-Label	
title	Titel	
legend	Legende	
text	Text	
subplot	Subplot	
hold	Einfrieren	
contour	Höhenlinien	
mesh	3D-Netz	
surf	3D-Flächengraph	
spy	Sparsestruktur	
print	Drucken	
clf	Löscht Figur	
close	Schließt Figur	

Literatur

Die Literaturangaben sind alphabetisch nach den Namen der Autoren sortiert.

- [1] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: *LA-PACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 3. Auflage, 1999.
- [2] DONGARRA, J., BUNCH, J., MOLER, C., STE-WART, G.: LINPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.
- [3] GOLUB, G., VAN LOAN, C.: *Matrix Computations*. The Johns Hopkins University Press, 3. Auflage, 1996.
- [4] Gramlich, G.: Lineare Algebra. Mathematik-Studienhilfen. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2003.
- [5] Gramlich, G.: Anwendungen der Linearen Algebra. Mathematik-Studienhilfen. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2004.
- [6] Gramlich, G.: Zur Linearen Algebra: Zusammenfassungen, Ergänzungen, Erweiterungen und Übersichten. Skript, März 2007. http://www.hs-ulm.de/gramlich.
- [7] Gramlich, G., Werner, W.: Numerische Mathematik mit Matlab. dpunkt.verlag, 2000.

- [8] GREENBAUM, A.: Iterative Methods for Solving Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.
- [9] HIGHAM, D., HIGHAM, N.: MATLAB *Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
- [10] Kanzow, C.: Numerik linearer Gleichungssysteme. Springer Verlag, 2005.
- [11] Kelley, C.: *Iterative Methods for Linear ans Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1995.
- [12] Knorrenschild, M.: Numerische Mathematik. Mathematik-Studienhilfen. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2003.
- [13] Lehoucq, R., Sorensen, D., Yang, C.: AR-PACK User's Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.
- [14] Matlab: *Data Analysis*. The MathWorks, Natick, MA, USA, 2006. http://www.mathworks.de.
- [15] Matlab: *Graphics*. The MathWorks, Natick, MA, USA, 2006. http://www.mathworks.de.
- [16] Matlab: *Mathematics*. The MathWorks, Natick, MA, USA, 2006. http://www.mathworks.de.
- [17] Matlab: *Programming*. The MathWorks, Natick, MA, USA, 2006. http://www.mathworks.de.

- [18] Meister, A.: Numerik linearer Gleichungssysteme. Vieweg-Verlag, 1999.
- [19] Moler, C.: Numerical Computing with Matlab. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004.
- [20] NITSCHKE, M.: *Geometrie*. Mathematik-Studienhilfen. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2005.
- [21] Rieder, A.: Keine Probleme mit Inversen Problemen. Vieweg Verlag, 2003.
- [22] SAAD, Y.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2. Auflage, 2003.
- [23] Shampine, L., Gladwell, I., Thompson, S.: *ODEs with* Matlab. Cambridge, 2003.
- [24] Shampine, L. F., Reichelt, M. W.: The MATLAB ODE suite. SIAM J. SCI. Computing, (18):1–22, 1997.
- [25] Simulink: Simulation and Model-Based Design. The MathWorks, Natick, MA, USA, 2006. http://www.mathworks.de.
- [26] SMITH, B., BOYLE, J., DONGARRA, J., GARBOW, B., IKEBE, Y., KLEME, V., MOLER, C.: *Matrix Eigensystem Routines*-Eispack *Guide*. Springer-Verlag, 1976.
- [27] Weigand, H., Weth, T.: *Computer im Mathematikunterrricht*. Spektrum Akademischer Verlag, 2002.

Stichwortverzeichnis

', 44	Ахіом, 224
*, 10, 15, 29, 33, 48	axis, 70, 228
+, 29, 33, 48, 50	axis equal,70
-, 29, 33, 48, 50	axis square,70
-all, 27	
-realmin, 34	backsub, 110, 111
.*, 50	bar, 72, 228
.+, 50	bar3, 72
./, 15, 50	bar3h, 72
.^, 50	barh, 72
.50	Basis, 200
/, 15, 29, 33	bench, 29
:, 42, 43, 227	Bessel, 39
^, 29, 48	bicg, 122
92, 99, 107, 110, 113, 129, 216, 227	bicgstab, 122
	binomial, 200
abs, 33	binopdf, 157
acos, 39	blkdiag, 52
acosd, 39	box off, 63
acsc, 39	box on, 63
Albrecht Dürer, 12	bvp4c, 152–155, 228
all, 227	bvpinit, 153
all(any(B)),55	
angle, 33	C/C++, 17, 18, 38, 78, 83
ans, 29, 32, 227	cart2pol,72
any, 227	cart2sph,72
any(all(B)), 55	cat, 216
any(B(1:2,1:3)), 55	cd, 28
any(B), 55	ceil, 162
Ascii, 25	ceil(x), 40
asec, 39	cell, 216–218, 228
asin, 39	cellplot, 220
asind, 39	cgs, 122
atan, 39	char, 115, 216–218, 220, 228
atan2, 39	chol, 118
,	, -

Cholesky, 118	degree, 189	
cholinc, 122	delete, 28	
Cholseky, 118	demo, 228	
circshift, 53	demos, 201	
class, 218	Derive, 224	
clc, 27, 228	det, 101, 106, 107	
clear, 228	diag, 45, 52, 106, 107, 216, 227	
clear global, 88	diag(rot90(B)), 53	
Cleve Moler, 2, 223, 224	diary, 228	
clf, 228	diary off, 25	
close, 228	diff, 54, 184	
coeff, 189	digits, 198	
colorbar, 66	dir, 28, 228	
colormap, 66	disp, 57, 228	
colspace, 97, 106	doc, 22, 26, 27, 227, 228	
combinat, 160	doc audiovideo, 222	
compan, 46	doc datatypes, 37, 218	
compose, 175	doc demo, 26	
computer, 28, 32, 35	doc demos, 26	
cond, 116, 227	doc elfun, 26, 33, 38	
condest, 117	doc elmat, 32, 43	
contour, 228	doc funfun, 59, 150	
conv, 131, 133	doc gallery, 122	
convert, 189	doc general, 84	
corrcoef, 157	doc graph2d, 64	
cos, 39, 126	doc graph3d, 64	
cosd, 39	doc graphics, 76	
cot, 39	doc iofun, 58	
cov, 129, 157	doc lang, 32, 81, 84	
cross, 51, 96, 107	doc 1ogo, 223	
csc, 39	doc ops, 42, 43, 79	
cumprod, 54	doc optim, 141, 201, 204	
cumsum, 54	doc polyfun, 133	
cumtrapz, 147, 148	doc sim, 212	
cylinder, 76, 77	doc sparfun, 123, 214	
-	doc specfun, 38	
dblquad, 146, 148	doc specgraph, 64	
deconv, 132, 133	doc stats, 156	

doc strfun, 218	feval, 127
doc symbolic, 200	fft, 142, 228
doc uicontrol, 76	fftw, 143
dot, 96, 107	Fibonacci, 170
double, 3, 9, 31, 33–37, 115, 216–218, 220,	filebrowser, 22
228	find, 227
dsolve, 148, 150, 192	find(B), 55
	finite(B(:,3)),55
echo off,82	finverse, 176
echo on, 82	fix(x), 40
edit, 84, 228	fliplr, 52, 53
eig, 84, 102, 104-107, 121, 123, 124, 214,	fliplr(A), 53
216, 227	flipud, 11, 52
eigs, 2, 104, 123, 124, 214	floor, 161
eigshow, 103, 104	floor(x), 40
Eispack, 16	fminbnd, 129, 140, 141, 228
ellipsoid, 76	fminsearch, 140, 141
end, 128, 227	fopen, 58
eps, 32, 34, 113, 120, 227	for, 79, 80, 227
eps*realmin, 35	format, 31, 57
error, 227	format bank, 31
Euklid, 38	format long, 31
Euler, 178, 198, 199, 224	format short, 31
Excel, 23, 25	Fortran, 17, 18, 83, 114
exist, 28, 88	Fourier, 141–143, 195–197
exit, 19	fourier, 196
exit, quit, 228	fplot, 59, 64, 87, 228
exp, 39, 126	fprintf, 57, 58, 228
expand, 170	fread, 58
expm, 52, 106, 107, 124	Frobenius, 115, 116
eye, 45, 92, 107, 213, 227	fscanf, 58, 59
ezmesh, 64, 68	fsolve, 137, 201
ezplot, 64, 67, 70, 126, 228	full, 123, 212
ezplot3,67	function, 40, 41, 83, 84
ezsurf,68	function_handle, 217, 218
	funm, 52
factor, 189	funtool, 190
factorial, 156, 157, 159	fwrite,58

fzero, 59, 136, 137, 141, 177	Horner, 87
	horner, 189
Galiei, 9	hostid, 28
gallery, 46	humps, 64, 137, 144, 147
Gauss, 112, 191	
Gauss-Jordan, 110	i, 32, 227
Gauss, 224	IDL, 224
gausselim, 110, 111, 113	IEEE, 31, 33, 34
gaussjord, 110, 113	if, 79, 81, 227
gcd, 173	ifft, 142
gemres, 122	imag, 33
geomean, 157	image, 74, 76
geopdf, 157	imread, 74, 76
get, 76	Inf, 32, 34, 35
global, 88	inf, 32, 227
gmres, 122, 123	input, 56, 228
grid, 61, 228	inputname, 84
gsvd, 129	int, 148, 186, 187
guide, 76	int16, 35, 36
	int32, 35, 36
Hadamard, 46	int8, 35, 36
hadamard, 46, 129	interp1, 136, 228
handel, 222	interp2,66
Hankel, 39, 46	intmax, 36
hankel, 46	intmin, 36
Hdf, 25	inv, 92, 106, 107, 113, 216
Heaviside, 85	inverse, 27
heaviside, 85	invhilb, 46
help, 26, 27, 227, 228	ipermute, 216
Hermite, 118, 121, 122, 124	isa,218
Hesse, 185	iscount, 177
Ніднам, 122	isempty, 227
Highams, 46	isequal, 227
hilb, 46	iskeyword,88
Hilbert, 46	isvarname, 32
HıQ, 224	
hist, 72, 228	j, 32, 227
hold, 228	Jасові, 186

jacobian, 185, 186 mad, 157 Java, 17, 18, 38 magic, 10, 46 JORDAN, 106 Maple, 12, 110, 157-160, 173, 177, 190, jordan, 106 191, 193, 198–200, 224 maple, 110, 173, 200 kron, 51 maple('with(combinat)'), 200 Kronecker, 51 MATHCAD, 224 MATHEMATICA, 224 LAPACK, 114 Matlab, 87 Laplace, 188, 197, 208 matlabrc, 29 laplace, 188, 197 matlabroot, 28 legend, 228 max, 38, 54, 227 Leibniz, 183 mean, 54, 82, 157, 164, 227 length, 227 median, 54, 82, 157 license, 28 mesh, 63, 64, 156, 228 limit, 179, 180 meshgrid, 63 LinearAlgebra, 200 mfun, 198 LINPACK, 16 mhelp, 189 linprog, 202, 203 mhelp Basis, 200 linsolve, 107 mhelp binomial, 200 linspace, 42, 43, 156, 227 mhelp combinat, 160, 200 load, 21, 46, 228 mhelp LinearAlgebra[Basis], 201 log, 39, 40 mhelp('LinearAlgebra, Basis'), 201 log10, 40 min, 54, 227 log2, 40 minres, 122 loglog, 71 mode, 157 logm, 52 movie, 76 logo, 223 mtaylor, 190 logspace, 43 multinomial, 159 lookfor, 27, 84, 228 MuPAD, 224 lookfor complex, 33 1s, 28 namelengthmax, 32 1sqcurvefit, 204, 205 NaN, 32, 34, 85, 227 1sqnonlin, 204 nargchk, 84 1sqr, 122 nargin, 84, 228 lu, 92, 107, 117, 215, 227 nargout, 84, 228 luinc, 122 nchoosek, 158 Macsyma, 224 ndgrid, 216

ndims, 216	plot3, 62, 63
norm, 38, 114, 115, 227	plotyy, 72
normest, 116	pol2cart, 72
normpdf, 128, 157	polar, 72
null, 97, 106, 107, 120	poly, 102, 106, 107, 130, 133
num2str, 218, 228	polyder, 132, 133
Tuni25C1, 210, 220	polyfit, 129, 133, 134, 228
Octave, 224	polyint, 132, 133, 134, 228
ode113, 151	polyval, 132, 133
ode15i, 150, 151	pretty, 170
ode15s, 151, 152	-
ode23, 151	print, 74, 76, 228
ode23s, 151	prod, 54, 157, 159, 227
ode23t, 151	PV-WAVE, 224
ode23tb, 151	pwd, 28
ode45, 59, 129, 149–152, 209, 210, 228	qmr, 122
ones, 44, 45, 213, 227	qr, 107, 227
optimset, 137, 141	quad, 41, 59, 143, 145–148
orth, 97, 107, 120	quad, 41, 52, 143, 143–146 quad, quad1, 129
of til, 97, 107, 120	quad1, 143, 145–148, 228
Pascal, 18, 46, 83, 219	quadr, 143, 143–148, 228 quadv, 148
pascal, 46	quauv, 146 quit, 19
patch, 77	_
path, 28	quiver, 72, 73
Path Browser, 28	quiver3,73
pathtool, 28	rand, 10, 45, 48, 161–163, 213, 216, 227
pcg, 122, 123	rand(n), 161
pdepe, 154, 155	randn, 45, 161–163, 216, 227
pdsolve, 192–194	randtool, 164
peaks, 65, 66	range, 157
perms, 158, 159	rank, 98, 106, 107, 120
permute, 216	rcond, 113, 117
pi, 31, 32, 227	real, 33
pie, 72	realmax, 34
piecewise, 177	realmin, 34
pinv, 121	Record, 219
pinv, 121 pinv(A)*b, 108	REDUCE, 224
-	
plot, 15, 60-64, 228	REICHELT, 151

repmat, 227	soundsc, 222
reshape, 52, 227	sparse, 212, 213, 228
reshape(A,4,3),53	spdiags, 213
RLaB, 224	speye, 213
roots, 133, 137, 140, 228	sph2cart,72
Rosenbrock, 151	sphere, 76, 77
rosser, 46	spline, 135, 219, 228
rot90, 52	spones, 213
rot90(A,3),53	sprand, 213
rot90(B),53	sprandn, 213
round, 40	sprintf, 57, 228
round(x), 40	spy, 215, 228
rref, 92, 106, 107, 110, 112, 113	sqrt, 26, 33, 39, 42
rsolve, 191	sqrtm, 52
Runge-Kutta, 149, 151	squeeze, 216
Runge-Kutte, 151	startup, 29
	std, 54, 82, 157, 227
save, 21, 228	stem, 72
saveas, 76	str2num, 218
Scilab, 224	struct, 216-219, 228
sdvs, 124	subplot, 62, 228
sec, 39	subs, 171, 175, 176, 189
semilog, 228	sum, 54, 227
semilogx,71	surf, 63, 228
semilogy, 71	svd, 106, 107, 214, 227
set, 76	svds, 124, 214
Shampine, 151	switch, 79, 81
shiftdim, 216	switch, case, 227
sim, 212	sym, 217, 218, 220
simplify, 171	symmlq, 122
simulink, 206	symsum, 181, 182
sin, 26, 39, 59, 126	
sind, 39	Töplitz, 46
single, 35-37	tabulate, 157
size, 44, 46, 84, 227	tan, 39
solve, 110, 112, 113, 139, 140, 177, 189,	Taylor, 190, 191
219	taylor, 189
sort, 54, 82, 227	taylortool, 190

text, 228
tic, 124
tic, toc, 228
title, 61, 63, 228
toc, 124
toeplitz, 46
trapz, 146, 148
tril, 52, 106, 107, 227
triplequad, 146, 148
triu, 52, 106, 107, 227
triu(B), 53
type, 28, 176, 228

uiimport, 25 uint16, 35, 36 uint32, 35, 36 uint8, 35, 36 unidpdf, 157 unifpdf, 157

vander, 46 Vandermonde, 46 var, 54, 129, 157, 164 varargin, 129, 219 varargout, 129, 219 vectorize, 56, 90 ver, 29 version, 29 vpa, 198

Wathen, 122 wav, 222 wavread, 222 what, 28, 228 whatsnew, 29 which, 28, 228 while, 79, 80, 227 who, 21, 228 whos, 9, 21 whos global, 88 why, 129 WILKINSON, 46 wilkinson, 46

xlabel, 63, 228
xlim, 228

ylabel, 63, 228
ylim, 228

zeros, 45, 80, 92, 107, 213, 216, 227 zlabel, 63