Eine Einführung in MATLAB

Prof. Dr. Günter Gramlich
Fachhochschule Ulm
Fachbereich Grundlagen
http://www.rz.fh-ulm.de/~gramlich
Ulm, 8. September 2004

Inhaltsverzeichnis

1. Einleitung

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.

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, das NATS-Projekt. 1976 lag als Ergebnis dieser Bemühungen das Softwarepaket EISPACK zur Lösung algebraischer Eigenwertprobleme vor [8]. Im Jahr 1975 begannen die Arbeiten an einem effizienten und portablen Softwarepaket zur Lösung linearer Gleichungssysteme. Das Ergebnis war das Softwarepaket LINPACK [1]. 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.

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 `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 die verschiedenste Weise darzustellen.
Programmierung. MATLAB verfügt über eine eigene höhere Programmiersprache. Dadurch stellt MATLAB ein offenes System dar. 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.

Die grundlegenden Datenelemente von MATLAB sind Matrizen bzw. allgemeiner mehrdimensionale Felder (Arrays), 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 FORTRAN oder C/C++ erforderlich wäre.

Im Verlauf mehrerer Jahre und durch Beiträge vieler Benutzer hat sich MATLAB zu seinem heutigen Umfang entwickelt. In Hochschulen ist MATLAB das bevorzugte Lehrmittel für Grund- und Aufbaukurse in Mathematik, Ingenieurtechnik und Wissenschaft. 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 [2].
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 (`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 *(Extended) 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 41 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, die in MATLAB eingebunden werden können.

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 oder physikalischen Systems, das als Blockdiagramm 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 Differentialgleichungen (ODE-Löser).

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

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.

2. 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 Visialisierungsmöglichkeiten.
- 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) verweise ich Sie auf unser Buch [3] oder auf [4].

3. 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` verlassen Sie MATLAB. Weitere Hinweise finden Sie in den MATLAB-Handbüchern. Gegebenenfalls müssen Sie Ihren Systemmanager nach lokalen Installationseigenschaften befragen.

4. MATLAB unterbrechen

Mit `ctrl-c` können Sie MATLAB jederzeit unterbrechen.

5. 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
```

6. Das `help`-Kommando

Das `help`-Kommando ist eine einfache Möglichkeit, Hilfe über eine MATLAB-Funktion im Kommandofenster 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 Einteilung. Gibt man `help` alleine ein, so wird diese Gruppierung angezeigt.

```
>> help
HELP topics

matlab\general - General ...
matlab\ops     - Operators ...
usw.
```

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

7. 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` erhalten Sie eine Liste aller elementaren mathematischen Funktionen, die in MATLAB realisiert sind.

8. Demos

Durch den Aufruf

```
>> demo
```

erhalten wird das Hilfe-Fenster geöffnet und Sie können sich Demonstrationen über MATLAB und seine Toolboxen anschauen. Weitere Info erhalten Sie mit `doc demo` (`help demo`) oder `doc demos` (`help demos`).

9. 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 dimensions.
    ACOS   Inverse cosine.
    ACOSH  Inverse hyperbolic cosine.
    ACOT   Inverse cotangent.
    ACOTH  Inverse hyperbolic cotangent.
    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 tangent.
    ATANH  Inverse hyperbolic tangent.
    ERFCINV Inverse complementary error function.
    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
```

10. Einfaches Rechnen

Zusammen mit den Klammern `()` sind `+ - * / ^` und die grundlegenden Rechenoperationen.

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

Wie wird hier gerechnet: So `3+4/(5*6)` oder so `3+(4/5)*6`? Die Antwort geben die Vorrang-Regeln:

1. Größen in Klammern `()`,

2. Potenzen `^`,

3. `* /` von links nach rechts,

4. `+ -` von links nach rechts.

Folglich gilt $3 + 4/5 \cdot 6 = 3 + (4/5) \cdot 6 = 3 + (0.8 \cdot 6) = 3 + 4.8 = 7.8$.

11. Welche aritmetische 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 1 aufgezeigt (Eine komplette Tabelle für alle MATLAB-Operationen zeigt die Tabelle 2). Für Operatoren, die auf einer Ebene stehen, ist der Vorrang von links nach rechts geregelt. Klammern können immer verwendet weerden, 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
```

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

Tabelle 1: Vorrang-Tabelle arithmetischer Operationen

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 2: Vorrang-Tabelle der MATLAB-Operationen

12. 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 10^3 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`. Die Tabelle 3 gibt mögliche numerische Zahlenformate an.

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

Tabelle 3: Zahlenausgabe in MATLAB

13. Variablen und Konstanten

Ein Variablenamen muss mit einem Buchstaben beginnen und darf aus maximal 31 Buchstaben, Zahlen und Unterstrichen bestehen. Ist ein Name länge, so sind nur die ersten 31 Stellen signifikant. Umlaute sind nicht erlaubt! Erlaubt sind zum Beispiel

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

Nicht erlaubt sind

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

Aufgabe 1 (Variablennamen)
Wieviel verschiedene MATLAB Variablenamen stehen in folgender Zeile?

```
anna ANNA anNa aNna_anna
```

Lösung: Vier verschiedene Variablennamen.

Darüber hinaus gibt es vordefinierte Variablen, siehe Tabelle 4. Achtung! Sie können diese spezielle Variablen überschreiben; vermeiden Sie dies aber, wenn immer möglich.

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

14. Eine Sitzung aufzeichnen

Das Kommando

```
>> diary MeineSitzung
```

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

Tabelle 4: Spezielle Variablen

sorgt dafür, dass der nachfolgende 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.

15. Mathematische Funktionen

MATLAB verfügt über viele mathematische Funktionen, zum Beispiel über die trigonometrischen Funktionen `sin`, `cos` oder `tan`. Ihre Arumente werden im Bogenmaß (Radian) erwartet. Liegen die Werte im Gradmaß (Grad) vor, so sind die Funktionen `sind`, `cosd` und `tand` zu verwenden. Für die anderen trogonometrischen Funktionen gilt entsprechendes, sieh `doc elfun`.

Die nachfolgenden Zeilen bestätigen die Ergebnisse $\cos(\pi/2) = 0$ und $\cos(90°) = 0$.

```
>> cos(pi/2)
ans =
  6.1232e-017
>> cosd(90)
ans =
     0
```

Aufgabe 2 (Trogonometrische F.)
Bestätigen Sie die Tabelle

x	0	$1/2\pi$	π	$3/2\pi$	2π
$\cos x$	1	0	-1	0	1

und die Tabelle

ϕ (in Grad)	0	90	180	270	360
$\cos\phi$	1	0	-1	0	1

Lösung: Hier die Bestätigung:

```
>> 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`.

```
>> sqrt(4), exp(4), log(x^2+1)
ans =
     2
ans =
   54.5982
ans =
    2.8332
```

Ein komplette Übersicht über die grundlegenden und speziellen mathematischen

Funktionen von MATLAB findet man mit Hilfe der Kommandos `doc elfun` (`help elfun`) und `doc specfun` (`help specfun`).

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 28. Zum Beispiel erklärt man die quadratische Funktion $f(x) = x^2$, $x \in \mathbf{R}$ 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. Zum Beispiel definiert man die quadratische Funktion $f(x) = x^2$, $x \in \mathbf{R}$ 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 37) dies auch so verlangen. Funktionen mit mehreren Variablen können ebenso erklärt werden.

Aufgabe 3 (Funktionen)
Definieren Sie in MATLAB die Funktion $f(x, y) = -xye^{-2(x^2+y^2)}$, $(x, y) \in \mathbf{R}^2$ 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 =
     0
ans =
   -0.0183
```

Weitere Informationen finden Sie mit `doc function` (`help function`) und `doc function_handle` (`help function_handle`). Für symbolische Funktionen, siehe Abschnitt 41.

16. Visualisieren in MATLAB

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.

16.1. 2D-Grafik

Ein einfaches Beispiel soll die erste Situation erläutern. Hierzu nehmen wir an,

dass die Messung des zeitlichen Verlaufs der Abkühlung einer Flüssigkeit die Werte aus der Tabelle 5 ergab. Wir wollen dieses

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 5: Abkühlung einer Flüssigkeit

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

Der Befehl

```
>> plot(x,y)
```

erzeugt ein Grafikfenster und zeichnet die Elemente von `x` gegen die Elemente von `y` und verbindet diese Punkte geradlinig. Die Abbildung 1 zeigt das Ergebnis.

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:

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

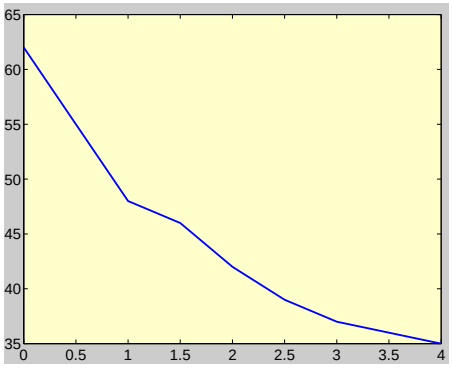

Abbildung 1: Abkühlung einer Flüssigkeit

2. Die Funktion muß an jedem Diskretisierungspunkt ausgewertet werden:

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

3. Ein Polygonzug muß gezeichnet werden, der die Punkte $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ 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 2 zeigt das Ergebnis.

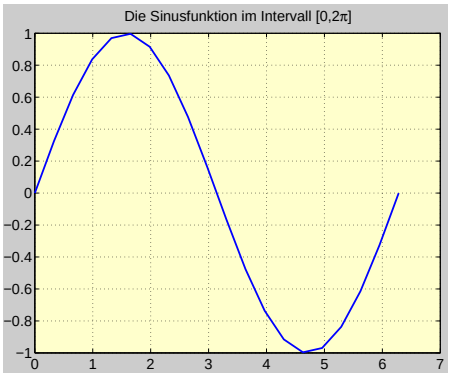

Abbildung 2: Ein einfacher Plot

Aufgabe 4 (Grafik)

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:

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

Aufgabe 5 (Grafik)

Erzeugen Sie mit einem einzigen `plot`-Befehl die Graphen der Funktionsterme $\sin(kx)$ über dem Intervall $[0, 2\pi]$ für $k = 1 : 5$.
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);
plot(x,sin(x))
hold on
for k=2:5
   plot(x,sin(k*x),'-')
end
hold off
```

Aufgabe 6 (Grafik)

Zeichnen Sie den Graph der Funktion

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

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);
```

Die folgenden Befehle zeichnen den Graph.

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

16.2. 3D-Grafik

Bevor Sie weiterlesen: Beachten Sie, dass der Operator `.*` elementweise arbeitet.

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 3 zeigt das Ergebnis.

Aufgabe 7 (3D-Grafik)
Zeichnen Sie den Graph der Funktion

$$\begin{cases} -\frac{1}{2}x_2 + 2 & \text{für } x_1 \geq 0 \text{ und } x_2 \geq 0 \\ 2 & \text{sonst.} \end{cases}$$

im Bereich $(x_1, x_2) \in [-2, 2]^2$.
Lösung: Den Graph kann man wie folgt plotten.

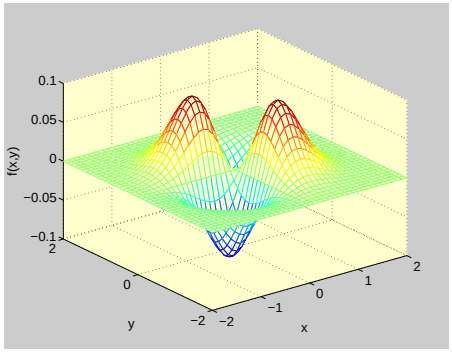

Abbildung 3: $f(x, y) = -xye^{-2(x^2+y^2)}$

```
>> [X,Y] = meshgrid(-2:0.1:2,...
-2:0.1:2);
>> Z = 2*ones(size(X));
>> Z = -0.5*Y.*(X>=0 & Y>=0)+2;
>> mesh(X,Y,Z)
>> axis([-2,2,-2,2,1,3])
```

Die Funktionen `ezplot` und `ezmesh` erlauben es, explizit gegebene Funktionen einfacher zu zeichnen. Weitere Grafikfunktionen findet man in der Tabelle 6

Name	*Beschreibung*
`loglog`	Logarithmisches KO-System
`semilogx`	x-Achse logarithmisch
`semilogy`	y-Achse logarithmisch
`polar`	polare Darstellung
`hist`	Histogramm
`bar`	Balkendiagramm
`stem`	Punkte mit Linien

Tabelle 6: Weitere Grafikfunktionen

Mehr Informationen über Visualisierungsmöglichkeiten findet man mit `doc graph2d` (`help graph2d`), `doc graph3d`

(`help graph3d`) und `doc specgraph` (`help specgraph`).

Aufgabe 8 (Grafik, 3D)
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:

```
[X,Y] = meshgrid(-3:0.1:3);
Z = 1/5*cos(X)+Y.*exp(-X.^2-Y.^2);
mesh(X,Y,Z)
```

oder als Einzeiler mit `ezmesh`.

```
ezmesh('1/5*cos(x)+y*exp(-x^2-y^2)',
[-3,3])
```

Aufgabe 9 (Grafik, 3D)
Gegeben sei der Funktionterm

$$f(x,y) = x^2 - 8x + y^2 - 6y - 0.1xy + 50.$$

Benutzen Sie die `mesh`-Funktion, um das Minimum und den minimalen Wert der Funktion f im Bereich $0 < x < 5$, $0 < y < 5$ zu schätzen.

16.3. Funktionsdarstellungen

Kennt man den Funktionsterm einer reellwertigen Funktion einer reellen Variablen, so kann man mit der Funktion `fplot` den Graph darstellen. Der Aufruf

```
>> fplot(@(x)exp(-x^2),[-3,3])
```

plottet die Funktion

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

im Intervall $[-3,3]$. Mit

```
>> fplot(@humps,[-2,2])
```

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

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

Die Abbildung 20 zeigt den Graph von `humps` im Intervall $[-2,2]$.

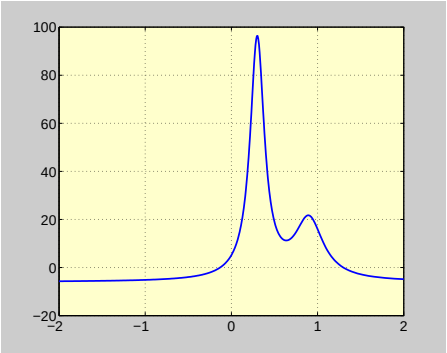

Abbildung 4: Graph der `humps`-Funktion

Die Abbildung 5 zeigt vier verschiedene Darstellungen einer Funktion mit zwei Variablen. Es handelt sich hier um die sogenannte `peaks`-Funktion, die in MATLAB be-

reits vordefiniert ist. Es ist die Funktion

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

Die `peaks`-Funktion geht durch Translationen und Skalierungen aus der GAUSSschen Normalverteiluungsfunktion hervor. Das Bild links oben in Abbildung 5 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),
```

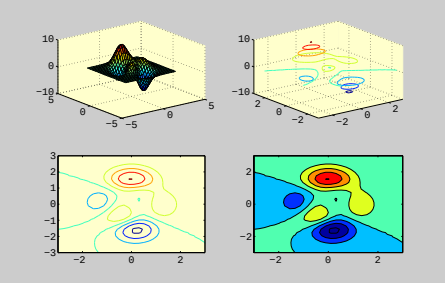

Abbildung 5: Darstellungen der `peaks`-Funktion

Die Abbildung 6 zeigt den Graph der Funktion $f(x,y) = xe^{x^2+y^2}$, $(x,y) \in \mathbf{R}^2$ Die Figure wurde mit Hilfe der Anweisungen erzeugt:

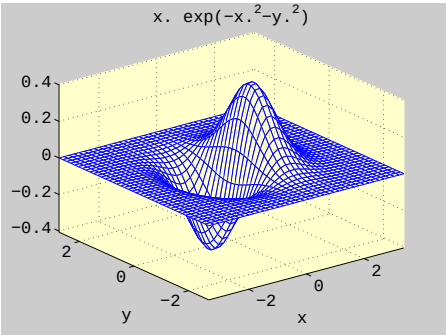

Abbildung 6: Graph

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

Die Abbildung 7 zeigt Höhenlinien der `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

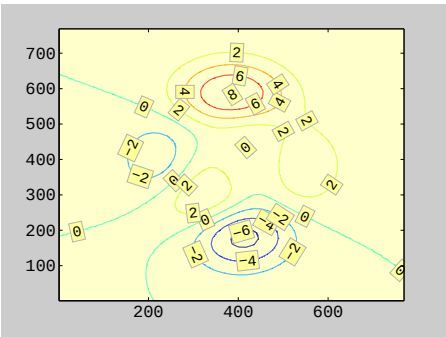

Abbildung 7: Geglättete Höhenlinien der `peaks`-Funktion

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])
```

16.4. Parametrisierte Kurven

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

```
ezplot('cos(3*t)*cos(t)',...
'cos(3*t)*sin(t)',[0,2*pi]), grid;
```

erzeugt man die Abbildung 8.

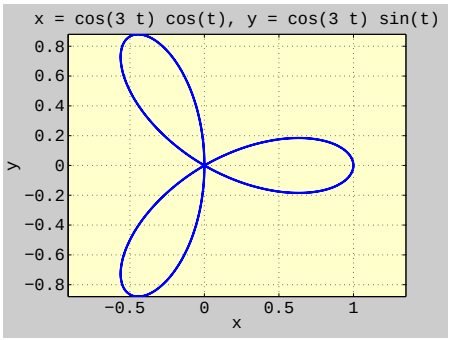

Abbildung 8: Blütenblattkurve

Die Anweisung

```
ezplot3('exp(-0.2*t)*cos(t)',...
'exp(-0.2*t)*sin(t)',...
't',[0,20],'animate')
```

erzeugt die räumliche Kurve in Abbildung 9. Durch das zusätzlich Argument `animate` im Funktionsaufruf erhält man eine Animation der räumlichen Kurven in dem Sinn, dass ein roter Punkte vom Anfagang- zum Endpunt läuft.

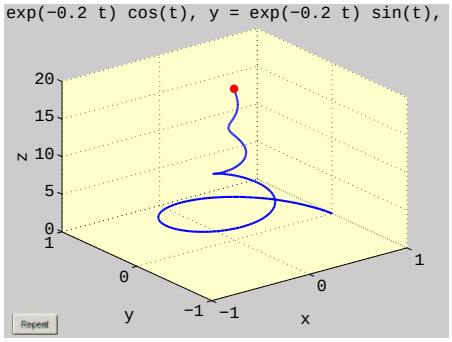

Abbildung 9: Räumliche Kurve

Aufgabe 10 (Räumliche Kurve)
Plotten Sie die räumliche Kurve

$$x(t) = (1 + t^2)\sin(20t)$$
$$y(t) = (1 + t^2)\cos(20t)$$
$$z(t) = t$$

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 11 (Ebene Kurve)
Plotten Sie die Zykloide

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

für $t \in [0, 4\pi]$.
Lösung: Die Kurve kann zum Beispiel wie folgt geplottet werden.

```
ezplot('t-sin(t)','1-cos(t)',...
[0,4*pi])
grid on, xlabel('x(t)'),
ylabel('y(t)')
```

Aufgabe 12 (Ebene Kurve)
Zeichnen Sie die Kurve

$$\begin{aligned} x &= \sin(-t) + t \\ y &= 1 - \cos(-t) \end{aligned}$$

in der x, y-Ebene für $0 \leq t \leq 4\pi$.
Lösung: Es handelt sich um eine Zykloide.

```
ezplot('sin(-t)+t','1-cos(-t)',...
[0,4*pi])
```

16.5. Parametrisierte Flächen

Mit den MATLAB-Funktion `ezmesh` und `ezsurf` können parametrisierte Flächen im $\mathbf{R}^3$ 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 10.

```
>> ezmesh('(10+4*cos(theta))*cos(phi)',...
'(10+4*cos(theta))*sin(phi)',...
'4*sin(theta)')
```

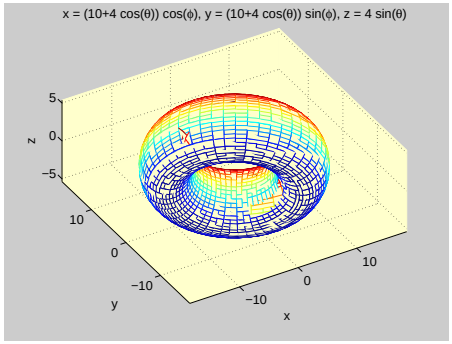

Abbildung 10: Ein Torus

16.6. Koordinatenachsen skalieren

Beachten Sie, dass MATLAB die x- und y-Achse (und natürlich auch die z-Achse im 3D Fall) automatisch skaliert. Wenn Sie möchten, dass die x- und y-Achse gleich lang, also quadratisch sind, dann müssen Sie `axis square` eingeben. Wollen Sie dagegen, dass die x- und y-Achse die gleiche Skalierung haben, so geht das mit dem Befeht `axis equal`. Die Abbildung 11 zeigt dies anhand des Bereichs $[-4, 4] \times [-2, 2]$.

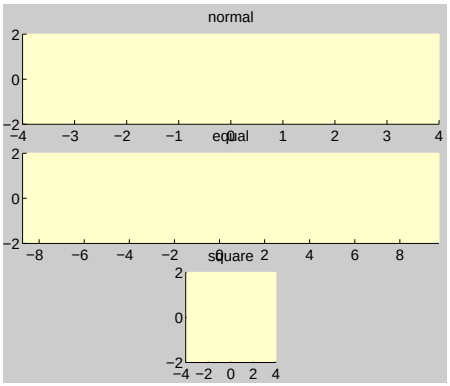

Abbildung 11: Skalierungen

17. Grafiken exportieren

Hat man eine Grafik erzeugt, so will man diese häufig in einem bestimmten Grafikformat abspeichern und gegebenenfalls in ein Satz- oder Textverarbeitungssystem einbinden. Das geht zum Beispiel so:

```
%-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:

1. 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 Unterpunkt *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.

18. 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, jedes Grafikobjekt anzusprechen. Mit `doc graphics` (`help graphics`) erhalten Sie eine komplette Übersicht über alle zur Verfügung stehenden Kommandos und Funktionen.

19. 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`).

20. Matrizen

Ein rechteckiges Zahlenschema mit m Zeilen und n Spalten heißt (m, n)-Matrix. Es ist üblich, eine Matrix in eckige oder runde Klammern zu setzen; wir wählen eckige. Ist $m = 2$ und $n = 3$, so liegt eine $(2, 3)$-Matrix vor, zum Beispiel

$$\boldsymbol{A} = \begin{bmatrix} 1 & \sqrt{2} & -2 \\ 7 & -3 & \pi \end{bmatrix}.$$

Die Matrix $\boldsymbol{A}$ können wir nun Zeile für Zeile wie folgt in MATLAB eingeben

```
>> A = [1 sqrt(2) -2; 7 -3 pi]
A =
    1.0000    1.4142   -2.0000
    7.0000   -3.0000    3.1416
```

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 $\boldsymbol{A}$ ändern, so kann dies auf zwei Arten geschehen. Die Anweisung `A(1,3) = 5` ändert zum Beispiel das Elemente $a_{13} = -2$ der Matrix $\boldsymbol{A}$ 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

```
>> x = 2:6
x =
     2     3     4     5     6
```

Die Schrittweite muss nicht notwendigerweise Eins sein

```
>> 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 $\boldsymbol{A}$ aus den Zeilen Spalten und aus den Spalten Zeile, so entsteht die transponierte Matrix $\boldsymbol{A}^{\mathrm{T}}$. In MATLAB erreicht man dies mit dem '-Operator.

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

Nützliche und häufig verwendete Matrizen stellt MATLAB als eingebaute Funktionen zur Verfügung; man muss nur die Größe angeben. Die Funktion `ones` erzeugt eine Matrix mit lauter Einsen.

```
>> ones(2,3)
ans =
     1     1     1
     1     1     1
```

Die Nullmatrix wird mit der Funktion `zeros` erzeugt.

```
>> Z1 = zeros(3,2)
Z1 =
     0     0
     0     0
     0     0
>> Z2 = zeros(size(A))
Z2 =
     0     0     0
     0     0     0
```

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

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

Diagonalmatrizen werden mit der Funktion `diag` erzeugt.

```
>> D = diag([1 2 3])
D =
     1     0     0
     0     2     0
     0     0     3
```

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

```
>> A = [1 -2; 3 2]; x = [3; 1];
>> A*x
ans =
     1
    11
```

Die Operatoren `+` und `-` können eingesetzt werden, um Matrizen miteinander zu addieren bzw. zu subtrahieren.

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

Aufgabe 13 (Diagonalmatrizen)

Erzeugen Sie eine Diagonalmatrix mit 1, 2, 3, 4 und 5 auf der Diagonalen!

Lösung:

```
>> diag([1 2 3 4 5])
ans =
     1     0     0     0     0
     0     2     0     0     0
     0     0     3     0     0
     0     0     0     4     0
     0     0     0     0     5
```

Aufgabe 14 (Dreiecksmatrizen)

Erzeugen Sie eine 6×6 obere (untere) Dreiecksmatrix mit Zufallszahlen zwischen 0 und 1!

Lösung:

```
>> triu(rand(6))
```

Aufgabe 15 (Symmetrische Matrizen)

Erzeugen Sie eine 10×10 symmetrische Matrix $\boldsymbol{S}$ mit Zufallswerten zwischen 0 und 8!

Lösung:

```
>> A = 4*rand(8);
>> A+A';
```

Aufgabe 16 (Matrizen)

Wir betrachten die folgende Matrix $\boldsymbol{A}$:

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

Geben Sie die Ordnung (Größe) dieser Matrix an! Wie kann man in MATLAB die Größe von $\boldsymbol{A}$ bestimmen? Geben Sie alle Indizes an, deren Matrixelemente 5.5 sind.

Lösung: Die Ordnung der Matrix $\boldsymbol{A}$ ist $(5, 3)$. Mit der Funktion `size` kann man die Ordnung in MATLAB bestimmen. Die Indizes sind: $1, 1$, $2, 2$ und $5, 2$.

Aufgabe 17 (Matrizen)

Gegeben sei die folgende Matrix $\boldsymbol{A}$:

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

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

$$\begin{bmatrix} -4.6 & 9.0 \\ 5.5 & 7.7 \end{bmatrix}.$$

Aufgabe 18 (Matrizen, Vektoren)

Geben Sie jeweils den Vektor `c` an, nachdem Sie die folgenden Operationen ausgeführt haben. Überprüfen Sie Ihre Ergebnisse in MATLAB.

```
a = [2 -1 5 0];
b = [3 2 -1 4];
```

(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;`

Aufgabe 19 (Matrizen)
Erzeugen Sie mit der MATLAB-Funktion `rand` eine 5×5-Zufallsmatrix $\boldsymbol{A}$. Welches sind die Werte der folgenden Ausdrücke? Überlegen Sie sich die Resultate, bevor Sie die Rechnung am Computer durchführen.

```
A(2,:)     A(:,1)
A(:,5)     A(1,1:2:5)
A([1,5])   A(4:-1:1,5:-1:1)
```

Aufgabe 20 (Matrizen)
Gegeben seien die folgenden Matrizen:

$$\boldsymbol{A} = \begin{bmatrix} 0 & -1 & 0 & 3 \\ 4 & 3 & 5 & 0 \\ 1 & 2 & 3 & 0 \end{bmatrix}$$

und

$$\boldsymbol{B} = \begin{bmatrix} 1 & 3 & 5 & 0 \\ 3 & 6 & 9 & 12 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}.$$

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))`

21. Matrizenoperationen

Beachten Sie im folgenden stets, dass Vektoren und Skalare spezielle Matrizen sind!

Die Tabelle 7 zeigt Matrizenoperationen in MATLAB.

Symbol	*Operation*
`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`^`	Potenzieren

Tabelle 7: Matrizenoperationen in MATLAB

Wenn Sie zwei Matrizen multiplizieren, so geht geht das natürlich nur dann, wenn die Mulitplikation definiert ist, das heißt die Matrizen die entsprechenden Größen haben.

```
>> A = [1 2 4; 2 6 0];
B = [4  1  4  3; 0 -1 3 1; ...
2 7 5 2];
A*B
ans =
    12    27    30    13
     8    -4    26    12
```

Analog für die anderen Operationen. Manchmal ist es notwendig, Matrizen ele-

mentweise zu multiplizieren (HADAMARD-Produkt), dann hilt der `.*`-Operator.

```
>> C = [1 2; 3 4];
>> D = [5 6; 7 8];
>> C.*D
ans =
     5    12
    21    32
```

Andere Operatoren, die elementweise arbeiten, sind `./` (rechte Division), `.\` (rechte Division) und `.^` (Potenzieren).

Weitere Infos mit `doc ops` (`help ops`).

Aufgabe 21 (Direktes Produkt)

Berechnen Sie in MATLAB das direkte Produkt von

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & -2 \\ 5 & -1 & 0 \end{bmatrix}$$

und

$$\boldsymbol{B} = \begin{bmatrix} 1 & 3 \\ -1 & 5 \\ 2 & -2 \end{bmatrix}.$$

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

```
>> A = [1 2 3; 3 1 -2; 5 -1 0];
>> B = [1 3; -1 5; 2 -2];
>> kron(A,B)
ans =
  1     3     2     6     3     9
 -1     5    -2    10    -3    15
  2    -2     4    -4     6    -6
  3     9     1     3    -2    -6
 -3    15    -1     5     2   -10
  6    -6     2    -2    -4     4
  5    15    -1    -3     0     0
 -5    25     1    -5     0     0
 10   -10    -2     2     0     0
```

Aufgabe 22 (Matrizenoperationen)

Es seien $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ und $\boldsymbol{D}$ nachfolgend definierte Matrizen.

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

$$\boldsymbol{C} = \begin{bmatrix} 1 & 5 \\ -5 & 3 \end{bmatrix}$$

und

$$\boldsymbol{D} = \begin{bmatrix} 4 & 3 & -2 \\ 1 & 0 & 5 \\ 2 & -1 & 6 \end{bmatrix}.$$

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) $\boldsymbol{B} + \boldsymbol{C}$

(c) $\boldsymbol{D}\boldsymbol{A}$

(d) $2\boldsymbol{A} - 3\boldsymbol{B}$

(e) $\boldsymbol{A}^{\mathrm{T}}$

(f) $\boldsymbol{C}^2$

Lösung:

(a)
```
>> A+B
ans =
     0     5
     6     2
    10     0
```

(b) $\boldsymbol{B}+\boldsymbol{C}$ ist nicht definiert.

(c)
```
>> D*A
ans =
     4    22
    16     8
    18     8
```

(d)
```
>> 2*A-3*B
ans =
     5     0
    -8    14
   -15     5
```

(e)
```
>> A'
ans =
     1     2     3
     3     4     1
```

(f)
```
>> C^2
ans =
   -24    20
   -20   -16
```

Aufgabe 23 (Matrizenoperationen)

Die nachfolgenden Regeln der Matrizenalgebra sehen so aus wie die der reellen Zahlen. Trotzdem sind einige davon falsch. Benutzen Sie MATLAB, um die falschen herauszufinden, und geben Sie für jede falsche Regel ein Gegenbeispiel.

(a) $\boldsymbol{A}+\boldsymbol{B}=\boldsymbol{B}+\boldsymbol{A}$

(b) $\boldsymbol{AB}=\boldsymbol{BA}$

(c) Falls $\boldsymbol{AB}=\boldsymbol{O}$ ist, dann ist $\boldsymbol{A}=\boldsymbol{O}$ oder $\boldsymbol{B}=\boldsymbol{O}$.

(d) Falls $\boldsymbol{A}^2=\boldsymbol{O}$ ist, dann ist $\boldsymbol{A}=\boldsymbol{O}$.

(e) $(\boldsymbol{A}+\boldsymbol{B})^2=\boldsymbol{A}^2+2\boldsymbol{AB}+\boldsymbol{B}^2$

(f) $(\boldsymbol{A}-\boldsymbol{B})(\boldsymbol{A}+\boldsymbol{B})=\boldsymbol{A}^2-\boldsymbol{B}^2$

(g) $\boldsymbol{A}(\boldsymbol{B}+\boldsymbol{C})=\boldsymbol{AB}+\boldsymbol{AC}$

(h) $(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{C}=\boldsymbol{CA}+\boldsymbol{CB}$

(i) $(\boldsymbol{AB})^2=\boldsymbol{A}^2\boldsymbol{B}^2$

Aufgabe 24 (Matrizenoperationen)

Warum gibt es in MATLAB kein `.+` Operator?

Lösung: Die Matrizenaddition ist bereits elementweise definiert.

Aufgabe 25 (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?

```
a + b      a' + b
a + b'     a' + b'
a - b      a' - b
a - b'     a' - b'
a * b      a' * b
a * b'     a' * b'
```

```
a \ b      a' \ b
a \ b'     a' \ b'
a .* b     a' .* b
a .* b'    a' .* b'
a .\ b     a' .\ b
a .\ b'    a' .\ b'
```

Aufgabe 26 (Matrizenoperationen)
Was ist 0^0 in MATLAB?
Lösung: Es ist $0^0 = 1$ in MATLAB.

22. Lineare Gleichungssysteme

In vielen Anwendungen muss man lineare Gleichungen lösen. Daher ist es in MATLAB besonders einfach, solche zu lösen. Hierzu dient der \-Operator (Backslash-Operator).

22.1. Quadratische Systeme

Ist $\boldsymbol{A}$ eine reguläre (quadratische) Matrix, so gibt es genau eine Lösung des linearen Systems $\boldsymbol{Ax} = \boldsymbol{b}$ und zwar für jede rechte Seite $\boldsymbol{b}$. Zum Beispiel ist die Matrix

$$\boldsymbol{A} = \begin{bmatrix} 1 & -2 \\ 3 & 2 \end{bmatrix}$$

regulär und daher gibt es für $\boldsymbol{b} = (1, 11)$ genau eine Lösung; diese ist $\boldsymbol{x} = (3, 1)$. In MATLAB löst man dies in einem Einzeiler, nachdem man $\boldsymbol{A}$ und $\boldsymbol{b}$ eingegeben hat.

```
>> A = [1 -2; 3 2];
>> b = [1; 11];
>> x = A\b
x =
     3
     1
```

Ist die Koeffizientenmatrix $\boldsymbol{A}$ singulär, so erhält man eine Fehlermeldung, auch dann wenn $\boldsymbol{b}$ im Spaltenraum von $\boldsymbol{A}$ liegt und es nicht nur eine, sondern unendlich viele Lösungen gibt.

Aufgabe 27 (Lineare Systeme)
Berechnen Sie die allgemeine Lösung des linearen Gleichungssystems

$$\underbrace{\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}}_{\boldsymbol{b}}.$$

Lösung: Die allgemeine Lösung ist die eindeutige Lösung $\boldsymbol{x} = (-1, 2, 2)$. Dies zeigen die nachfolgenden MATLAB-Zeilen.

```
>> A = [2 4 -2; 4 9 -3; -2 -3 7];
>> b = [2; 8; 10];
>> x = A\b
x =
   -1.0000
    2.0000
    2.0000
```

Die Lösung ist eindeutig, sonst wäre die Matrix $\boldsymbol{A}$ singulär und MATLAB hätte mit einer Fehlermeldung geantwortet. Hier noch alternativ die symbolische Lösung.

```
>> x = sym(A)\b
x =
```

```
[ -1]
[  2]
[  2]
```

22.2. Überbestimmte Systeme

Gibt es mehr Gleichungen als Unbekannte, so nennt man $\boldsymbol{Ax} = \boldsymbol{b}$ überbestimmt. Das System $\boldsymbol{Ax} = \boldsymbol{b}$ hat in der Regel keine Lösung, aber das Ersatzproblem $\boldsymbol{Ax} \cong \boldsymbol{b}$ (die lineare Ausgleichsaufgabe) ist stets lösbar. Entweder hat das Ersatzproblem genau eine oder aber unendlich viele Lösungen, je nachdem ob die Spalten von $\boldsymbol{A}$ lineare unabhängig sind oder nicht. Im Fall, dass es unendlich viele Lösungen gibt erhält man durch `A\b` eine Basislösung.

22.3. Unterbestimmte Systeme

Ein lineares System $\boldsymbol{Ax} = \boldsymbol{b}$ heißt unterbestimmt, wenn weniger Gleichungen als Unbekannte vorliegen; in der Regel hat $\boldsymbol{Ax} = \boldsymbol{b}$ dann unendlich viele Lösungen. In diesem Fall wird durch `A\b` die Lösung kleinster Länge berechnet, das heißt von allen Lösungen $\boldsymbol{x}$ wird diejenige ermittelt, wo die Länge von $\boldsymbol{x}$ am kleinsten ist. Hat das System $\boldsymbol{Ax} = \boldsymbol{b}$ keine Lösung, so wird eine Basislösung des Ersatzproblems $\boldsymbol{Ax} \cong \boldsymbol{b}$ berechnet.

Weitere Informationen siehe `doc slash` (`help slash`).

23. Weitere Funktionen

Um die Eigenwerte einer quadratischen Matrix `A` zu berechnen, braucht man nur `eig(A)` einzutippen. Eine Eigenvektormatrix `X` zusammen mit den Eigenwerten (in der Diagonalmatrix `D`) erhält man durch die Anweisung `[X,D] = eig(A)`.

Analog lassen sich die Konditionszahl, die Determinate, die Inverse, die Norm, der Rang und die Spur einer Matrix mit einem Funktionsnamen ausrechnen. Die Tabelle 8 listet die entsprechenden MATLAB-Funktionen auf.

Name	*Aktion*
`cond`	Konditionszahl
`det`	Determinante einer Matrix
`inv`	Inverse einer Matrix
`norm`	Norm einer Matrix
`rank`	Rang einer Matrix
`trace`	Spur einer Matrix

Tabelle 8: MATLAB-Funktionen

Auch Matrizenfaktorisierungen wie die LU-, QR-, SCHUR- oder Singulärwertzerlegung sind über einen einzigen Funktionsnamen berechenbar. Zum Beispiel berechnet `[U,S,V] = svd(A)` eine Singulärwertzerlegung der Matrix `A`. Die Tabelle 9 gibt einen Überlick.

Aufgabe 28 (Rundungsfunktionen)
In MATLAB findet man folgende eingebaute Funktionen:

`ceil(x)` Rundet x zur nächsten ganzen Zahl auf.

Name	*Aktion*
`eig`	Berechnet Eigensystem
`lu`	LU-Zerlegung
`qr`	QR-Zerlegung
`schur`	SCHUR-Zerlegung
`svd`	Singulärwertzerlegung

Tabelle 9: Faktorisierungen

`fix(x)` Wählt von x den ganzzahligen Anteil.

`floor(x)` Rundet x zur nächsten ganzen Zahl ab.

`round(x)` Rundet x zur nächsten ganzen Zahl.

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(10.8))`

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)
```

Aufgabe 29 (Funktionen)
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 =
```

```
     7
>> min(A)
ans =
     1    -1    -2
>> min(x,y)
ans =
     0    -1    -2     7
>> mean(A)
ans =
    3.0000    3.3333    3.0000
>> median(x)
ans =
    1.5000
>> cumprod(A)
ans =
     1     3     7
     2    24    28
    12   -24   -56
>> sort(2*x+y)
ans =
     1     3     5    21
>> sort(A)
ans =
     1    -1    -2
     2     3     4
     6     8     7
```

Aufgabe 30 (Matlab-Funktionen)
Bestimmen Sie die Werte der folgenden Ausdrücke. Überprüfen Sie Ihre Ergebnisse dann in MATLAB.

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 0 & 3 \\ 8 & 7 & 0 \end{bmatrix}$$

(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 =
     0
>> finite(B(:,3))
ans =
     1
     1
     1
>> any(B(1:2,1:3))
ans =
     1     0     1
```

24. Zeichen und Zeichenketten

Außer dem `double`-Datentyp (Klasse) kennt MATLAB auch noch andere, zum Bei-

spiel Zeichen bzw. Zeichenketten (`char`). Zeichenketten werden durch Hochkommata begrenzt

```
>> s = 'Ich bin eine Zeichenkette'
s =
Ich bin eine Zeichenkette
```

Die Variable `s` ist eine Matrix (Array, Feld) der Größe $(1, 25)$ vom Typ `char`. Mit dieser Variablen können Sie nun mit allen erlaubten Feldoperationen manipulieren. Sie können zum Beispiel Zeichenketten zusammenfügen oder trennen. Mit den Funktionen `str2num` oder `num2str` können Sie Zeichen in Zahlen oder umgekehrt konvertieren.

Für weitere Informationen siehe `doc strfun` (`help strfun`).

25. 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.

26. 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.

26.1. Logische Operatoren

Werden Matrizen mit logischen Operatoren verknüpft, so geschieht dies komponentenweise. Die Tabelle 10 zeigt die logischen Operatoren.

Logische Operatoren	*Beschreibung*
`&`	logisches UND
`|`	logisches ODER
`~`	logisches NICHT

Tabelle 10: 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
```

26.2. 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. Die logische Funktion `isieee` testet Ihren Computer, ob er den IEEE-Standard erfüllt.

```
>> isieee
ans =
     1
```

Mein Rechner unterstützt diesen Standard und deswegen wird 1 (WAHR) zurückgegeben.

Weitere Informationen über logische Operatoren und Funktionen findet man mit `doc ops` (`help ops`).

Aufgabe 31 (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 | k<b-a`

27. Steuerstrukturen

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`

27.1. `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.

1. 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 MATLAB, 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 ihn zuvor initialisieren.*

27.2. `while`-Schleife

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

```
>> n = 1; Summe = 0;
>> while n <= 100
Summe = Summe+n;
n = n+1;
end
```

```
>> Summe
Summe =
        5050
```

Der nachfolgende MATLAB-Code berechnet die ersten sieben Vektoren $\boldsymbol{Au}$ mit Startvektor $\boldsymbol{u} = (1, 0)$ und der MARKOV-Matrix

$$\boldsymbol{A} = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}.$$

Anschließend werden die Vektoren gezeichnet.

```
u = [1;0]; A = [0.8 0.3; 0.2 0.7];
x = u; k = 0:1:7;
while length(x) <= 7
    u = A*u;
    x = [x u];
end
plot(k,x)
```

27.3. 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
```

27.4. 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`).

28. 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*.

28.1. Script-Files

Ein Script-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. Auch kann ein Script-File einen anderen m-File aufrufen.

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 komplex-konjugierten 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')
```

Dieser Script-File erzeugt 1000 Zufallszahlen und zeichnet ein Histogramm der Verteilung der Anzahl der reellen Eigenwerte. Die Abbildung 12 zeigt ein mögliches Resultat. Wollen Sie sehen, wie Ihr Script-

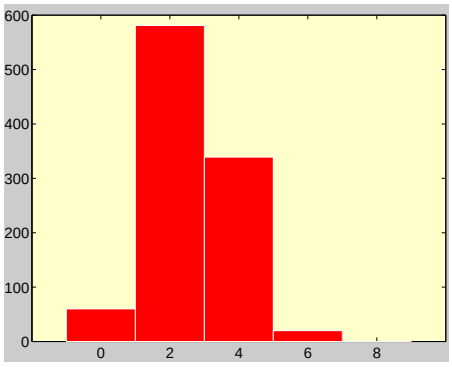

Abbildung 12: Histogramm zum Script

File den von Ihnen geschriebenen Code abarbeitet, so geben Sie `echo on` ein. Mit `echo off` können Sie den Vorgang wieder rückgängig machen.

28.2. 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 MATLAB erweitern. Mit Hilfe von Function-Files können Sie den MATLAB-Funktionsvorrat erweitern. Variablen in Function-Files sind lokale Varia-

blen. 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 Erzeugen 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, der Funktionsname und schließlich die Eingabeargumente. 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. Es ist möglich, dass keine Ausgabe- oder Eingabeargumente vorhanden sind.

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 erhalten Sie in der Matrix `V` die Eigenvektoren, in `D` die Eigenwerte und in `r` den Rang der Matrix `A`. `V,D,r` sind die Ausgabe- und `A` die Eingabeargumente der Funktion `MatrixEig`.

Aufgabe 32 (Function-File)

Schreiben Sie einen Function-File, der die Sprungfunktion (Einheitssprungfuktion, HEAVISIDE-Funktion)

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

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 );
```

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

definiert und mit den Anweisungen

```
fplot(@h,[-3,3,-0.5,1.5]), grid
```

erhält man die Abbildung 13.

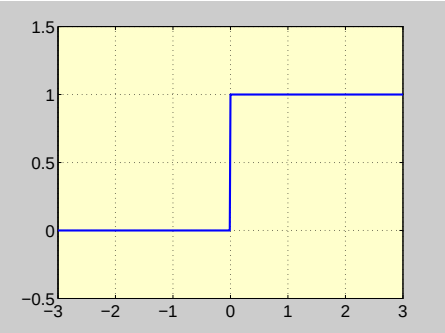

Abbildung 13: Sprungfunktion

Die Funktion `heaviside` aus MATLAB ist an der Stelle $t = 0$ undefiniert, das heißt MATLAB ordnet dem Nullpunkt `NaN` zu.

Aufgabe 33 (Function-File)
Zeichnen Sie die Graphen der verschobenen Sprungfunktionen $h(t+T)$, $t \in \mathbf{R}$ und $h(t-T)$, $t \in \mathbf{R}$ für $T = 2$ im Intervall $[-4, 4]$.
Lösung: Die Funktionen können mit dem Function-File 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 14.

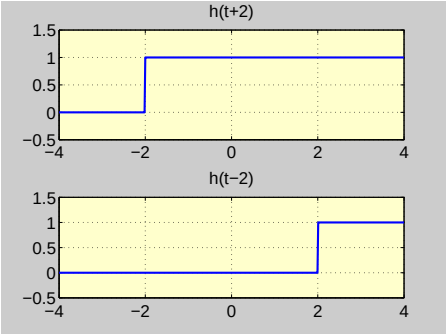

Abbildung 14: Verschobene Sprungfunktionen

Aufgabe 34 (Function-File)
Schreiben Sie jeweils einen Function-File, um die folgenden stückweise definierten Funktionen zu berechnen:

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

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

(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 $\text{rect}(t) = h(t + 0.5) - h(t - 0.5)$, $t \in \mathbf{R}$, $\text{ramp}(t) = th(t)$, $t \in \mathbf{R}$ und $g(t) = \sin(\frac{\pi t}{2})(h(t) - h(t - 1))$, $t \in \mathbf{R}$. 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 15.

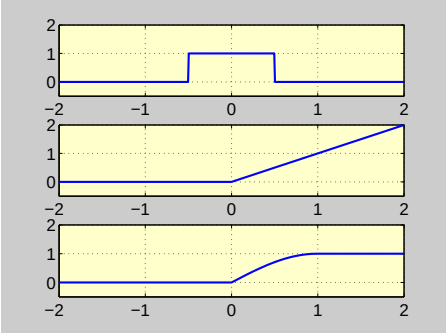

Abbildung 15: Graph der Funktionen

Aufgabe 35 (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 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;
```

Der Aufruf `fplot(@f,[-3,3])` zeichnet den Graph im Intervall $[-3, 3]$.

Aufgabe 36 (Function-File)
Schreiben Sie einen Function-File, um die Funktion $f(t) = t^{\frac{1}{3}}$, $t \in \mathbf{R}$ 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);
```

kann man den Graph mit `fplot('f',[0,1])` zeichnen.

Aufgabe 37 (Function-File)
Zeichnen Sie den Funktionsterm $f(x) = 5$ im Intervall $[0, 2]$.
Bemerkung: Zunächst empfindet man diese Übung schwieriger als sie tatsächlich ist. In MATLAB ist es zwar leichter, eine nicht konstante Funktion zu zeichnen als eine Konstante. Es gibt aber eine Reihe von Tricks, dies zu tun. Hilfreich sind hierbei die MATLAB-Funktionen `size` und `ones`.

29. 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);
```

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 nachfolgende Script zeigt, wie man die Auswertung dieser Funktion in einer skalaren Programmiersprache wie zum Beispiel FORTRAN 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. MATLAB 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 38 (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

$$A = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$$

gegeben und wir wollen die Eigenwerte der Matrizen $\boldsymbol{A}^k$ 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 längere 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 39 (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);
```

30. 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 41.13.

30.1. 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.

```
>> p1 = [1 -8 2 1 -12]
p1 =
     1    -8     2     1   -12
>> p2 = [2 0 1]
p2 =
     2     0     1
```

30.2. 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 40 (Polynomfunktionen)
Bestimmen Sie die reellen Nullstellen nachfolgender Polynome. Zeichnen Sie diese Polynome dann in einem geeigneten Intervall, um diese Nullstellen geometrisch überprüfen zu können ($x \in \mathbf{R}$).

(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
```

30.3. 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
```

30.4. 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.

```
function p1undp2 = addpoly(p1,p2)
%----------------------------
%-Addiert zwei Polynome
%----------------------------
if nargin < 2
   error('Zu wenig Argumente.')
end
%-Stellt sicher, dass p1 und p2
%-Zeilenvektoren sind.
p1 = p1(:)';
p2 = p2(:)';
lp1 = length(p1); %Länge von p1.
lp2 = length(p2); %Länge von p2.
p1undp2 = [zeros(1,lp2-lp1) p1]+
[zeros(1,lp1-lp2) p2];
```

Um die Funktion `addpoly` zu verstehen, betrachte man die nachfolgenden Zeilen:

```
>> addpoly(p1,p2)
ans =
     2     0     5    -7
```

Das Ergebnis ist das gleiche wie oben. Diese MATLAB-Funktion `addpoly` kann auch dazu verwendet werden, um Polynome zu subtrahieren. Dies zeigen die nachfolgenden MATLAB-Zeilen:

```
>> p7 = addpoly(p1,-p2)
p7 =
    -2     2    -1     1
```

$p_1(x) = x^2 + 2x - 3$ minus $p_2(x) = 2x^3 - x^2 + 3x - 4$ ergibt das Polynom $p_7(x) = -2x^3 + 2x^2 - x + 1$.

30.5. 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 $r(x) = 19x - 19$.

```
>> p2 = [2 -1 3 -4];
>> p1 = [1 2 -3];
>> [q,r] = deconv(p2,p1)
q =
     2    -5
r =
     0     0    19   -19
```

30.6. Ableiten von Polynomen

Die MATLAB-Funktion `polyder` differenziert ein Polynom. Die Ableitung des Polynoms $p_2(x) = 2x^3 - x^2 + 3x - 4$ ist das Polynom $p_2'(x) = 6x^2 - 2x + 3$.

```
>> p2 = [2 -1 3 -4];
>> dp2 = polyder(p2)
dp2 =
     6    -2     3
```

30.7. Auswerten von Polynomen

Polynome können mit der Funktion `polyval` ausgewertet werden. Die nachfolgenden Kommandos berechnen $p(2.5)$ für das Polynom $p(x) = 4x^3 - 2x^2 + x - 7$.

```
>> p = [4 -2 1 -7];
>> px = polyval(p,2.5)
px =
   45.5000
```

Die nachfolgenden 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);
>> px = polyval(p,x);
```

Aufgabe 41 (Polynomfunktionen)
Gegeben seien die folgenden Polynomfunk-

tionsterme

$$f_1(x) = x^3 - 3x^2 - x + 3$$
$$f_2(x) = x^3 - 6x^2 + 12x - 8$$
$$f_3(x) = x^3 - 8x^2 + 20x - 16$$
$$f_4(x) = x^3 - 5x^2 + 7x - 3$$
$$f_5(x) = x - 2$$

Zeichnen Sie die Polynome im Intervall $[0, 4]$. Benutzen Sie eingebaute MATLAB-Funktionen, um nachfolgende 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);
plot(x,ypa,x,ypb,x,ypc,x,ypd)
```

30.8. 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 Interpolations- und Approximationsaufgaben behandelt werden. Die Tabelle 12 fasst die Funktionen nochmals zusammen.

Funktion	*Beschreibung*
`conv`	Multipliziert Polynome
`deconv`	Dividiert Polynome
`poly`	Polynom aus Nullstellen
`polyder`	Berechnet Ableitung
`polyval`	Berechnet Polynomwerte
`roots`	Berechnet Nullstellen

Tabelle 11: Polynome in MATLAB

31. Polynominterpolation

Gegeben sind n Punkte in der Ebene $\mathbf{R}^2$, finde ein Polynom minimalen Grades, das durch alle Punkte geht. Ein solches Polynom hat höchstens den Grad $n - 1$ und ist eindeutig bestimmt; wir nennen 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 ist also ein quadratisches Polynom $p_2(t, \boldsymbol{x}) = x_1 + x_2 t + x_3 t^2$, das die drei gegebene Punkte interpoliert. Die nachfolgenden MATLAB-Zeilen berechnen das Poylnom 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
```

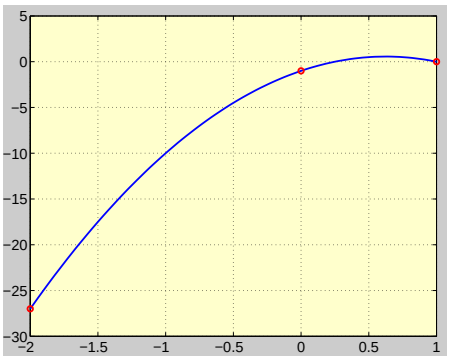

Abbildung 16: Polynominterpolation

Lässt man sich den Zeilenvektor `x` ausgeben, so findet man darin die gewünschten Koeffizienten des quadratischen Polynom; man erhält $p_2(t) = -t^2 + 5t - 4$.

32. Polynomapproximation

Die Funktion `polyfit` kann auch verwendet werden, um Daten durch Polynome zu approximieren. Gegeben sind m Punkte (t_i, y_i), $i = 1, 2, \ldots, m$ und gesucht ist ein Parametervektor $\boldsymbol{x} \in \mathbf{R}^n$, sodass die polynomiale Modellfunktion $p(t, \boldsymbol{x}) = x_1 + x_2 t + x_3 t^2 + \cdots + x_n t^{n-1}$ die gegebenen Punkte „bestmöglichst“ approximiert, wobei bestmöglichst im Sinne kleinster Fehlerquadrate zu verstehen ist:

$$\begin{array}{ll} \text{Minimiere} & \sum_{i=1}^{m} (y_i - p(t_i, \boldsymbol{x}))^2. \\ \boldsymbol{x} \in \mathbf{R}^n & \end{array}$$

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

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

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
```

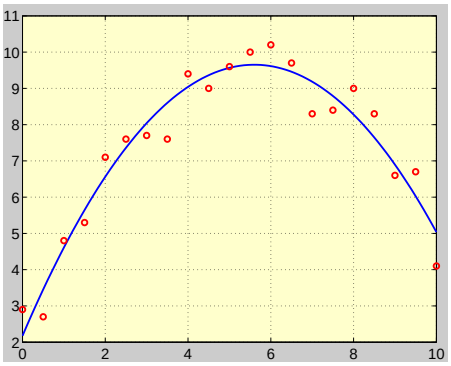

Abbildung 17: Polynomapproximation

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.

Die Polynomapproximation bzw. Ausgleichsaufgabe ist in der Statistik von besonderer Bedeutung und wird dort unter dem Namen Regressionsrechnung (*Engl.:data fitting*) studiert.

33. 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 nachfolgende 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 geeignet 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')
```

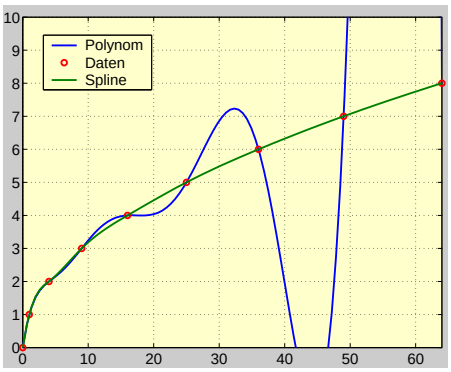

Abbildung 18: Kubische Splineinterpolation

34. Stückweise lineare Interpolation

Angenommen es sind m Punkte (t_i, y_i) gegeben. Setzt man $\boldsymbol{t} = (t_1, t_2, \ldots, t_m)$ und $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$ 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 $\text{hums}(t) = 1/\big((t-0.3)^2 + 0.01\big) + 1/\big((t-0.9)^2 + 0.04\big) - 6$.

```
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'),
```

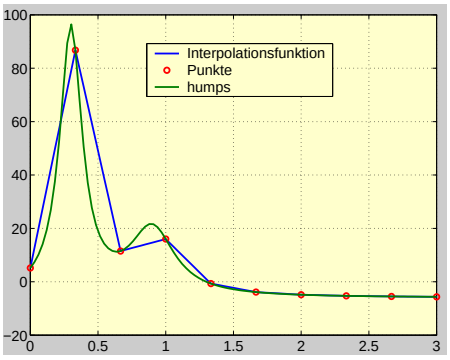

Abbildung 19: Lineare Interpolation

35. Nichtlineare Gleichungen

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 [3]. Es ist ein guter Algorithmus, aber

leider nur für skalarwertige Funktionen mit einer skalaren Variablen einsetzbar. Den m-File `fzero` findet man im Verzeichnis `Toolbox/MATLAB/funfun`.

Die Buckelfunktion `humps` hat zwei Nullstellen, siehe Abbildung 4. Wir berechnen diese mit `fzero`.

```
>> fzero(@humps,1)
ans =
    1.2995
>> fzero(@humps,0)
ans =
   -0.1316
```

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
```

Das zweite Argument des Funktionsaufrufes gibt jeweils den Startpunkt des iterativen Verfahren an, $x_0 = 1$ bzw. $x_0 = 3$. Sie können sich die einzelnen Iterationsschritte auch ausgeben lassen, indem Sie die entsprechende Option mit der `optimset`-Funktion setzen. Dies geht wie folgt:

```
>> options = optimset('Display',...
'iter');
>> x2 = fzero(f,1,options);
```

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. Diese Funktion gehört aber zur *Optimization Toolbox* und deshalb nur dann verwendbar, wenn diese installiert ist.

Aufgabe 42 (Nichtlineare Gleichungen)
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 43 (Polynomgleichungen)
Berechnen Sie mit `roots` die Lösung der Polynomgleichung

$$x^3 + 6x^2 - 11x - 6 = 0$$

Lösung: Die drei Lösungen findet man wie folgt:

```
>> roots([1 6 -11 -6])
ans =
   -7.3803
    1.8256
   -0.4453
```

Alle drei Lösungen sind reell.

36. Optimierung

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. Als Beispiel betrachten wir:

```
>> 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.

Wollen Sie eine Funktion f maximieren statt minimieren, dann können Sie $-f$ minimieren, da $\mathrm{Max}_x f(x) = -\mathrm{Min}_x(-f(x))$ ist.

Mit der Funktion `fminsearch` ist es möglich, ein lokales Minimum einer reellwertigen Funktion von $n \geq 1$ Variablen zu berechnen. Die Syntax ist ähnlich wie die von `fminbnd`: `x = fminsearch(fun,x0,options)`, außer dass das zweite Argument `x0` ein Startvektor ist anstatt eines Intervalls. Die quadratische Funktion $f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2$, $(x_1, x_2) \in \mathbf{R}^2$ hat in $\boldsymbol{x}_* = (0, 0)$ 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 Algortihmen finden Sie in der *Optimization Toolbox*.

Aufgabe 44 (Optimierung)
Lösen Sie das Problem

$$\begin{array}{ll} \text{Minimiere} & e^{x_1-x_2}+x_1^2+x_2^2. \\ \boldsymbol{x}\in\mathbf{R}^2 & \end{array}$$

Geben Sie Ihre Lösung auf vier Nachkommastellen an.
Lösung: Die Lösung ist $x_1 \approx -0.2836$ und $x_2 \approx 0.2836$, sowie $f(x_1,x_2) \approx 0.7281$. 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
```

37. 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 ein bestimmtes 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

```
quad(fun,a,b)
```

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 20 zeigt die Geometrie in Form 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

$$\int_1^2 x^x dx$$

kann von MATLAB nicht exakt (symbolisch) berechnet werden.

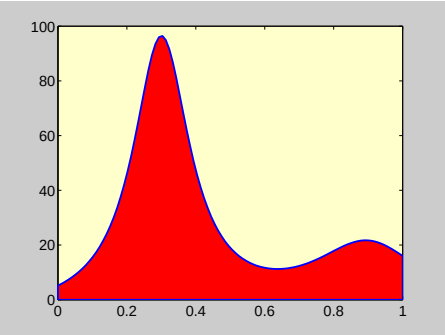

Abbildung 20: Graph und Flächeninhalt der `humps`-Funktion in $[0,1]$

```
>> syms x
>> f = x^x;
>> int(f,1,2)
Warning: Explicit integral could not
be found.
> In sym.int at 58
ans =
int(x^x,x = 1 .. 2)
```

Das Integral wird unverändert als nichtberechenbar ausgegeben. Wir können das Integral aber numerisch berechnen. Es ist:

```
>> quad(@(x)x.^x,1,2)
ans =
    2.0504
```

Aufgabe 45 (Numerische Integration)
Berechnen Sie die Länge der Zykloide

$$\begin{aligned} \boldsymbol{f} : \mathbf{R} &\to \mathbf{R}^2 \\ t &\mapsto (t - \sin t, 1 - \cos t) \end{aligned}$$

von $t = 0$ bis $t = 2\pi$.
Lösung: Es ist $\boldsymbol{f}'(t) = (1 - \cos t, \sin t)$ und somit gilt

$$\begin{aligned} L_{\boldsymbol{f}} &= \int_0^{2\pi} |\boldsymbol{f}'(t)| dt \\ &= \int_0^{2\pi} \sqrt{(1-\cos t)^2 + \sin^2 t}\, dt \end{aligned}$$

und damit in MATLAB

```
>> f = @(t)sqrt((1-cos(t)).^2+...
sin(t).^2);
>> quad(f,0,2*pi)
ans =
    8.0000
```

Aufgabe 46 (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.

Beachten Sie die Array-Multiplikation `.*` in obiger Funktion `fxlog`, damit die Funktion für vektorwertige Eingabeargumente korrekt funktioniert.

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 `quadl` 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.

Man spricht von einem zweifachen Integral (Doppelintegral) wenn es die Form

$$\int\int_D f(x,y)dxdy$$

hat. Hierbei ist D ein beschränktes Gebiet im zweidimensionalen Raum $\mathbf{R}^2$. Ein dreifaches Integral (Dreifachintegral) liegt vor, wenn die Gestalt

$$\int\int\int_G f(x,y,z)dxdydz$$

hat, wobei G ein beschränktes Gebiet im dreidimensionalen Raum $\mathbf{R}^3$ ist. Analog sind vier- und 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

$$\int_{y=4}^{6}\int_{x=0}^{1}(y^2e^x + x\cos y)dxdy$$

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 $\boldsymbol{x}$ 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 47 (Doppelintegral)
Berechnen Sie das Integral

$$\int_{y=0}^{2}\int_{x=0}^{1}(x^2)dxdy$$

Lösung: Es ist

```
>> dblquad(@(x,y) x.^2,0,1,0,2)
ans =
    0.6667
```

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 48 (Dreifachintegral)
Berechnen Sie das Integral

$$\int_{z=-1}^{1}\int_{y=0}^{1}\int_{x=0}^{\pi}(x^3)dxdydz$$

Lösung: Es ist

```
>> triplequad(@(x,y,z) x.^3,...
0, pi, 0, 1, -1, 1)
ans =
   48.7045
```

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 Besipiel. 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 `quadl` vorzuziehen, wenn der Integrand analytisch verfügbar ist.

Steht der Integrand f in tabellarischer Form $(t_i, f(t_i))$, $i = 1,2,\ldots,m$ 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 $i = 1,2,\ldots,m$ 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 21 zeigt den Graph der `humps`-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.

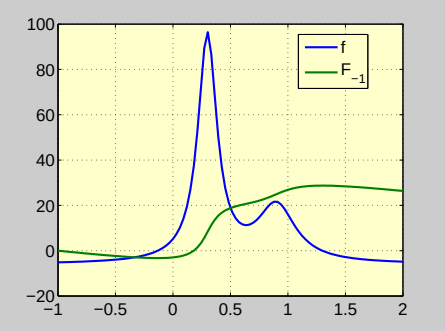

Abbildung 21: `humps`-Funktion und Integralfunktion im Intervall $[-1, 2]$

Aufgabe 49 (Integration)
Zeichnen Sie den Graph der Integralfunktion F_{-3} von $f(t) = 1/\sqrt{2\pi}e^{-t^2/2}$, $t \in \mathbf{R}$ 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 22 zeigt die Graphen. f ist die Dichtefunktion und $F_{-\infty}$ ist die Verteilungsfunktion der Standardnormalverteilung 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}')
```

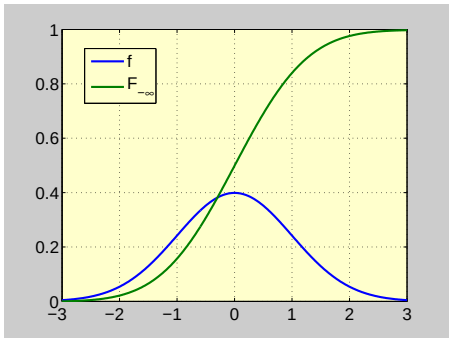

Abbildung 22: Die Graphen zu Aufgabe 49

Zusammenfassend können wir sagen: MATLAB 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 38, 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 41.9 und 41.15.

38. 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

$$u_{tt}(x,t) = c^2 u_{xx}(x,t)$$

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 41.15 für symbolisches Lösen von Differenzialgleichungen.

38.1. Anfangswertaufgaben

In MATLAB gibt es mehrere Funktionen, um Anfangswertaufgaben von gewöhnlichen Differentialgleichungen numerisch zu lösen. Anfangswertaufgaben haben folgende Form

$$\text{AWA}: \begin{cases} \frac{d}{dt}\boldsymbol{y}(t) = \mathbf{f}(t, \boldsymbol{y}) \\ \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \end{cases} \qquad t_0 \leq t \leq t_f$$

wobei t eine reelle Variable, $\boldsymbol{y}$ eine unbekannte vektorwertige Funktion und die gegebene Funktion $\mathbf{f}$ ebensfalls vektorwertig ist. Konkret kann man sich t als die Zeit vorstellen. Die Funktion $\boldsymbol{f}$ bestimmt die gewöhnliche Differenzialgleichung und zusammen mit der Anfangsbedingung $\boldsymbol{y}(t_0) = \boldsymbol{y}_0$ wird die Anfangswertaufgabe definiert. Der einfachste Weg solch ein Problem in MATLAB zu lösen, besteht darin, eine Funktion zu schreiben, die $\boldsymbol{f}$ auswertet und dann einen der MATLAB-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 $\boldsymbol{y}_0$. 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 Situatio-

nen 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 effizient für die meisten Probleme.

Um das skalare ($m = 1$) Anfangswertproblem ($0 \le t \le 3$)

$$\text{AWA}: \begin{cases} \frac{d}{dt}y(t) = -y(t) - 5e^{-t}\sin(5t) \\ y(0) = 1 \end{cases}$$

mit `ode45` zu lösen, kreieren wir als erstes den m-File `rSeite.m`, der die rechte Seite der Differentialgleichung 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 23. (Beachten Sie, dass wir beim Setzen der x- und y-Labels die Kommando/Funktionsdualität ausgenutzt haben.)

Die Eingabeargumente von `ode45` sind die Funktion `rSeite`, der 2-Vektor `tspan`, der die Zeitspanne der Simulation spzifiziert und der Anfangswert `y0`. Zwei Argumente

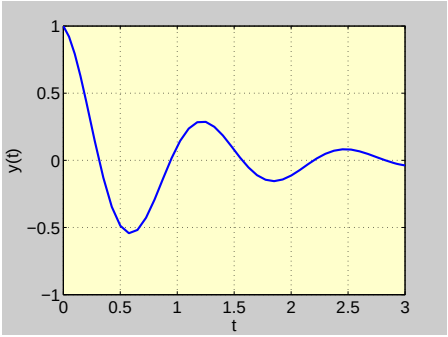

Abbildung 23: Skalares Anfangswertproblem

`t` und `y` werden zurückgegeben. Die `t` Werte sind im Intervall $[0, 3]$ geordnet 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 sind mehr Punkte angeordnet.

39. Zufallszahlen

Zufallszahlen sind im wissenschaftlichen Rechnen ein nützliches Hilfsmittel. In vielen Fällen werden Zufallszahlen in einer rechnergestützten Simulation eines komplexen Problems eingesetzt. 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 Zu-

fallszahlen generieren.

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.

39.1. 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 $[\texttt{eps}/2, (1-\texttt{eps}/2)]$):

```
>> rand(10,1)
ans =
    0.9501
    0.2311
    0.6068
    0.4860
    0.8913
    0.7621
    0.4565
    0.0185
    0.8214
    0.4447
```

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 $-\infty$). 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 );
```

Aufgabe 50 (Zufallsmatrizen)
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.

```
function A = randganz1bisk(m,n,k)
if nargin == 1, n=m; end;
if nargin < 3, k=9; end;
A = ceil( k*rand(m,n) );
```

39.2. 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)
```

39.3. 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 24. 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 24 zeigt, wie sich 10000 Zufallszahlen mit Mittelwert 0 und Varianz 1 ähnlich einer Gaußschen Glockenkurve verteilen.

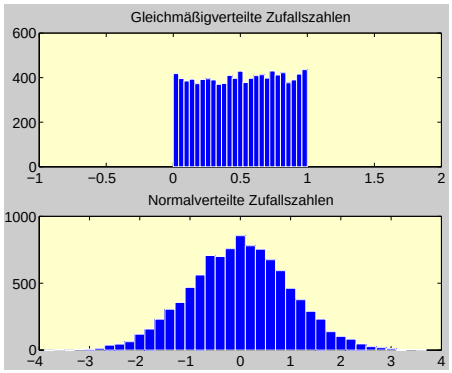

Abbildung 24: Zufallszahlen

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 25, um Unterschiede zwischen `rand` und `randn` klar zu machen.

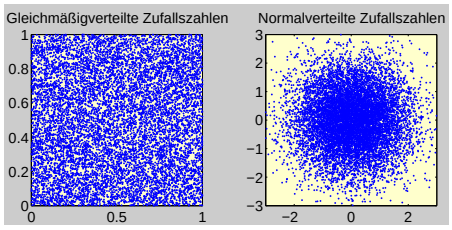

Abbildung 25: Zufallszahlen

39.4. 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 26 zeigt eine Exponentialverteilung mit Erwartungwert 1.

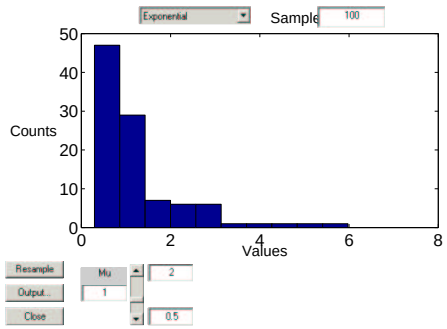

Abbildung 26: Exponentialverteilung

Aufgabe 51 (Zufallsmatrizen)
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)`.

(d) `-pi+2*pi*rand(1,10)`.

Aufgabe 52 (Zufallsmatrizen)
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)`

mit den Funktionen `mean` und `var` können Sie das überprüfen; am Besten mit noch mehr Zahlen.

Aufgabe 53 (Zufallsmatrizen)
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 haben Mittelwert null und Standardabweichung (Varianz) gleich eins haben.

40. Kombinatorik

Mit der Funktion `nchoosek` kann man Binomialzahlen berechnen.

Aufgabe 54 (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 $1/13983816 \approx 7.1511 \cdot 10^{-8} = 0.000000071511$.

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
     1     3     4
     1     3     5
     1     4     5
     2     3     4
     2     3     5
     2     4     5
     3     4     5
```

Aufgabe 55 (Binomialzahlen)
Geben Sie alle Teilmengen der Menge $\{1, 2, 3, 4, 5, 6\}$ mit genau vier Elementen an. Wieviel gibt es?
Lösung: Es ist:

```
>> nchoosek(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
```

Möglichkeiten.

Mit Hilfe der Funktion `factorial` kann man die Fakultät berechnen.

Aufgabe 56 (Fakultät)
Gegeben sind $n = 4$ verschiedene Objekte. Wie viele Anordnungsmöglichkeiten ohne Wiederholung gibt es?
Lösung: Es gibt genau

```
>> factorial(4)
ans =
    24
```

Möglichkeiten. Alternativ zur Funktion `factorial` können Sie die Funktion `prod` verwenden.

```
>> prod(1:4)
ans =
    24
```

Die Funktion `perms` gibt alle Anordungsmöglichkeiten an.

Aufgabe 57 (Fakultät)
Gegeben sind $n = 3$ verschiedene Objekte. Geben Sie alle Anordnungsmöglichkeiten ohne Wiederholung an.
Lösung: Davon gibt es genau $3! = 6$ Möglichkeiten; diese sind:

```
>> perms(1:3)
ans =
     3     2     1
     3     1     2
     2     3     1
     2     1     3
     1     2     3
     1     3     2
```

Siehe auch `doc specfun` (`help specfun`) für weitere Funktionen.

41. 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 können, 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, subtrahie-

ren 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.

41.1. Erste Schritte

Durch

```
>> syms x t
```

werden die beiden symbolischen Variablen `x` und `t` erzeugt, mit denen man durch

```
>> 3*x^2+t-1
ans =
3*x^2+t-1
```

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.

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 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
```

41.2. Algebraische Gleichungen

Mit der Funktion `solve` kann man Gleichungen lösen. Das nachfolgende 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
```

41.3. Grenzwerte

Die Folge

$$a_n = \frac{2n+1}{4n}, \qquad n \geq 1$$

hat den Grenzwert 1/2. Der nachfolgende MATLAB-Code bestätigt dies:

```
>> limit((2*n+1)/(4*n),inf)
ans =
1/2
```

Endliche oder unendliche Reihen lassen sich mit der MATLAB-Funktion `symsum` berechnen, die symbolisches Summieren erlaubt. Die Summe der endlichen Reihe

$$\sum_{k=1}^{10} 2^k = 2 + 4 + 8 + \cdots + 1024$$

ist 2046, was man durch folgende Zeilen bestätigen kann:

```
>> symsum(2^k,1,10)
ans =
2046
```

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 man[1]:

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}.$$

Mit dem Kommando `symsum` kann man dies überprüfen:

[1] Zum Beispiel mit Hilfe der Theorie der FOURIER-Reihen.

```
>> syms n
>> symsum(1/n^2,1,inf)
ans =
1/6*pi^2
```

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 $f'(2)$.

```
>> syms x
>> limit( (3/x-3/2)/(x-2) ,x,2 )
ans =
-3/4
```

Somit ist $f'(2) = -3/4$ oder

$$\lim_{x\to 2} \frac{3/x - 3/2}{x-2} = -\frac{3}{4}$$

Aufgabe 58 (Grenzwerte)

Berechnen Sie den Grenzwert

$$\lim_{x\to 0} \frac{\sin x}{x}$$

Lösung: Man berechnet den Grenzwert mit

```
>> syms x
>> limit( sin(x)/x )
ans =
1
```

Somit ist der Grenzwert also

$$\lim_{x\to 0} \frac{\sin(x)}{x} = 1\,.$$

Aufgabe 59 (Endliche Summen)

Berechnen Sie

$$\sum_{k=1}^{10} \frac{1}{k}.$$

Lösung: Mit

```
>> syms k
>> symsum(1/k,1,10)
```

erhält man

```
ans =
7381/2520
```

Es ist somit also

$$\sum_{k=1}^{10} \frac{1}{k} = \frac{7381}{2520}\,.$$

Aufgabe 60 (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

```
>> syms k n
```

so erhält man

```
a) >> simple(symsum(k,1,n))
   ans =
   1/2*n*(n+1)
```

```
b) >> simple(symsum(k^2,1,n))
   ans =
   1/6*n*(n+1)*(2*n+1)

c) >> simple(symsum(k^3,1,n))
   ans =
   1/4*n^2*(n+1)^2
```

Aufgabe 61 (Unendliche Summen)
Berechnen Sie die Summe der Reihe

$$\sum_{k=1}^{\infty} \frac{1}{4k^2-1}.$$

Lösung: Mit

```
>> syms k
>> symsum( 1/(4*k^2-1),1,inf)
```

erhält man

```
ans =
1/2
```

Also ist die Summe der Reihe

$$\sum_{k=1}^{\infty} \frac{1}{4k^2-1} = \frac{1}{2}.$$

Aufgabe 62 (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

```
ans =
log(2)
```

Aufgabe 63 (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

```
ans =
-1+1/4*pi
```

Also ist die LEIBNIZSCHE Reihe

$$\sum_{k=1}^{\infty} (-1)^k \frac{1}{2k+1} = \frac{\pi}{4} - 1.$$

41.4. 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, unnd zwar genauso, wie man es mathematisch tun würde. Die folgenden Anweisungen

```
>> syms x
>> f = x^2*exp(x);
>> diff(f)
```

erzeugen eine symbolische Variable x, bilden den symbolischen Ausdruck $x^2 \exp(x)$ und berechnen die Ableitung von f nach x. Das Ergebnis ist:

```
ans =
2*x*exp(x)+x^2*exp(x)
```

Aufgabe 64 (Ableitungen)
Berechnen Sie

$$\frac{d}{dx}(x^2 e^x)$$

Lösung: Mit

```
>> syms x
>> diff(x^2*exp(x))
ans =
2*x*exp(x)+x^2*exp(x)
```

sieht man, dass

$$\frac{d}{dx}(x^2 e^x) \;=\; 2xe^x + x^2 e^x$$

ist.

41.5. Partielle Differentiation

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.

41.6. Der Gradient

Mit der Funktion `jacobian` ist es möglich, den Gradient einer reellwertigen Funktion zu berechnen. Für $f : \mathbf{R}^3 \to \mathbf{R}$ mit $f(x, y, z) = e^{x+2y} + 2x \sin(z) + z^2 xy$ ist

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} e^{x+2y} + 2\sin(z) + z^2 y \\ 2e^{x+2y} + z^2 x \\ 2x\cos(z) + 2xyz \end{bmatrix}.$$

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]
[         2*exp(x+2*y)+z^2*x]
[         2*x*cos(z)+2*z*x*y]
```

41.7. Die Hesse-Matrix

Der Gradient der Funktion $f(x,y) = 2x^4 + 3x^2y + 2xy^3 + 4y^2$, $(x,y) \in \mathbf{R}^2$ ist

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 8x^3 + 6xy + 2y^3 \\ 3x^2 + 6xy^2 + 8y \end{bmatrix}$$

und die Hesse-Matrix ergibt sich zu

$$H_f(\boldsymbol{x}) = \begin{bmatrix} 24x^2 + 6y & 6x + 6y^2 \\ 6x + 6y^2 & 12xy + 8 \end{bmatrix}.$$

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

$$\nabla f(-2,3) = \begin{bmatrix} -46 \\ -72 \end{bmatrix}$$

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
```

41.8. Jacobi-Matrizen

Bei Funktionen mehrerer Veränderlicher berechnet man die Funktionalmatrix oder Jacobi-Matrix mittels `jacobian`. Will man die Jacobi-Matrix der Funktion

$$\begin{array}{rccc} \mathbf{f}: & \mathbf{R}^3 & \to & \mathbf{R}^3 \\ & (x,y,z) & \mapsto & (xy,x,z) \end{array}$$

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]
```

41.9. Integration

Die Funktion `int` erlaubt symbolische Integration. Als erstes Beispiel wollen wir die Gleichung $\int \sin(x)dx = -\cos(x) + c$ 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 $\int (x-t)^2 dx = 1/3(x-t)^3 + c$

```
>> syms x t
>> pretty(int((x-t)^2))

                3
   1/3 (x - t)
```

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*.

```
>> int('sin(x)',0,pi)
ans =
2
```

Aufgabe 65 (Integration)
Berechnen Sie das unbestimmte Integral

$$\int x\sin(x)dx.$$

Lösung: Mit

```
>> syms x
>> int(x*sin(x))
ans =
sin(x)-x*cos(x)
```

sieht man, dass

$$\int x\sin(x)dx = \sin(x) - x\cos(x) + c$$

ist.

Aufgabe 66 (Integration)
Berechnen Sie das bestimmte Integral

$$\int_0^\pi x\sin(x)dx$$

symbolisch.
Lösung: Mit

```
>> syms x
>> int(x*sin(x),0,pi)
ans =
pi
```

folgt also, dass

$$\int_0^\pi x\sin(x)dx \;=\; \pi$$

ist.

Aufgabe 67 (Integration)
Berechnen Sie das uneigentliche Integral

$$\int_1^\infty \frac{1}{x^5} dx.$$

Lösung: Mit

```
>> syms x
>> int(x^(-5),1,inf)
ans =
1/4
```

folgt also, dass

$$\int_1^\infty \frac{1}{x^5} dx = \frac{1}{4}$$

ist.

41.10. TAYLOR-Polynome

Die Funktion `taylor` erlaubt das symbolische Berechnen des TAYLOR-Polynoms einer Funktion. Zum Beispiel liefert

```
>> taylor(sin(x))
```

das TAYLOR-Polynom bis zur fünften Ordnung:

```
ans =
x-1/6*x^3+1/120*x^5
```

Aufgabe 68 (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

$$f(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24}.$$

41.11. 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. Mit Hilfe des Kommandos `taylortool` können Sie TAYLOR-Approximationen studieren.

41.12. Multivariate TAYLOR-Polynome

Zur Berechnung des TAYLOR-Polynoms einer reellwertigen Funtkion mehrerer Variablen kann man die Funktion `mtaylor` aus MAPLE verwenden.

Das TAYLOR-Polynom zweiten Grades der Funktion $f(x,y) = e^{x+y}$, $(x,y) \in \mathbf{R}^2$ 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:

```
>> syms x y
>> T2 = maple('mtaylor',exp(x+y),'[x,y]',3)
T2 =
1+x+y+1/2*x^2+x*y+1/2*y^2
```

Aufgabe 69 (Taylor)
Approximieren Sie die Funktion $f(x,y) = xe^y - x^3y$, $(x,y) \in \mathbf{R}^2$ 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%.

41.13. Polynome

Wir zeigen nun Beispiele für den Umgang mit „symbolischen Polynomen“, siehe Abschnitt 30 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 12. Hilfe zu den MAPLE-Funktionen erhalten Sie mit `mhelp <Funktionsname>`.

41.14. Lineare Algebra

Die folgenden Anweisungen erzeugen zwei symbolische Matrizen $\boldsymbol{K}$ und $\boldsymbol{G}$.

Funktion	*Bedeutung*
`coeff`	Koeffizient (MAPLE)
`convert`	Konvertiert (MAPLE)
`degree`	Grad des Polynoms (MAPLE)
`factor`	Linearfaktoren (MAPLE)
`horner`	HORNER-Darstellung
`solve`	Nullstellen
`subs`	Ersetzen

Tabelle 12: Zum symbolischen Rechnen mit Polynomen

```
>> syms a b t
>> K = [a+b,a-b ; b-a,a+b]
K =
[ a+b, a-b]
[ b-a, a+b]
>> G = [cos(t),sin(t) ;
        -sin(t),cos(t)]
G =
[cos(t), sin(t)]
[-sin(t), cos(t)]
```

Um die Inverse der Matrix

$$\boldsymbol{A} = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$$

zu berechnen, kann man wie folgt vorgehen:

```
>> A = [3 -1;-1 3]
A =
     3    -1
    -1     3
>> inv(sym(A))
ans =
[ 3/8, 1/8]
[ 1/8, 3/8]
```

Die Funktion `rref` transformiert eine gegebene Matrix unter elementaren Zeilenumformungen in reduzierte Zeilenstufenform. Als Beispiel betrachten wir das folgende quadratische lineare Gleichungssystem

$$\begin{aligned} x + y + 2z &= 9 \\ 2x + 4y - 3z &= 1 \\ 3x + 6y - 5z &= 0 \end{aligned}$$

Dieses Gleichungssystem besitzt die erweiterte Matrix

$$\begin{bmatrix} 1 & 1 & 2 & 9 \\ 2 & 4 & -3 & 1 \\ 3 & 6 & -5 & 0 \end{bmatrix}.$$

Die folgende Zeile berechnet die reduzierte Zeilenstufenform der Matrix.

```
>> rref(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 eindeutige Lösung des linearen Systems kann somit direkt abgelesen werden. Sie ergibt sich zu $x = 1$, $y = 2$ und $z = 3$.

Ist $\boldsymbol{A}$ eine Matrix, so kann man mit den Funktionen `colspace` und `null` eine Basis für den Bildraum und für den Nullraum von $\boldsymbol{A}$ berechnen.

```
>> A = sym([1 2;2 4]);
>> null(A)
ans =
[ -2]
```

```
[  1]
>> colspace(A)
ans =
[ 1]
[ 2]
>> colspace(A)'*null(A)
ans =
0
```

Die Matrix $\boldsymbol{A}$ hat einen eindimensionalen Nullraum und einen eindimensionalen Bildraum. Der Vektor $(-2, 1)$ ist eine Basis für den Nullraum und $(1, 2)$ ist eine Basis für den Bildraum von $\boldsymbol{A}$. Die letzte Anweisung bestätigt, dass diese senkrecht aufeinander stehen.

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

$$\boldsymbol{A} = \begin{bmatrix} -6 & 12 & 4 \\ 8 & -21 & -8 \\ -29 & 72 & 27 \end{bmatrix}$$

gegeben. Wir wollen MATLAB verwenden, um folgende Frage zu beantworten: Ist $\boldsymbol{A}$ reell diagonalisierbar, das heißt, können wir eine Matrix $\boldsymbol{X}$ finden, so dass $\boldsymbol{X}^{-1}\boldsymbol{A}\boldsymbol{X}$ diagonalisierbar ist? Die Eigenwerte von $\boldsymbol{A}$ finden wir mit `eig(sym(A))`. MATLAB antwortet mit:

```
ans =
[  3]
[ -2]
[ -1]
```

Da die Eigenwerte von $\boldsymbol{A}$ alle reell und voneinander verschieden sind, kann $\boldsymbol{A}$ diagonalisiert werden, das heißt, es gibt eine Matrix $\boldsymbol{X}$ mit

$$\boldsymbol{X}^{-1}\boldsymbol{A}\boldsymbol{X} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 3 \end{bmatrix}.$$

Wie findet man $\boldsymbol{X}$? Die Spalten von $\boldsymbol{X}$ können als Eigenvektoren zu den Eigenwerten -1,-2 und 3 gewählt werden. Gibt man

```
>> [X,D] = eig(sym(A))
```

ein, so gibt MATLAB in der Diagonalmatrix $\boldsymbol{D}$ die Eigenwerte zurück und die Spalten von $\boldsymbol{X}$ sind dazugehörige Eigenvektoren:

```
X =
[  1,  0, -4]
[  0,  1, -2]
[  1, -3,  1]
D =
[ -2,  0,  0]
[  0,  3,  0]
[  0,  0, -1]
```

Schließlich kann man mit `inv(X)*A*X` überprüfen, ob $\boldsymbol{X}^{-1}\boldsymbol{A}\boldsymbol{X}$ die gewünschte Diagonalmatrix ist. Wir erhalten das gewünschte Ergebnis:

```
ans =
[ -2,  0,  0]
[  0,  3,  0]
[  0,  0, -1]
```

Aufgabe 70 (Eigensysteme)
Gegeben ist die 4×4 Matrix

$$A = \begin{bmatrix} 12 & 48 & 68 & 88 \\ -19 & -54 & -57 & -68 \\ 22 & 52 & 66 & 96 \\ -11 & -26 & -41 & -64 \end{bmatrix}.$$

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:

```
>> A = [ 12  48  68  88;
        -19 -54 -57 -68;
         22  52  66  96;
        -11 -26 -41 -64];
>> [X,D] = eig(sym(A))
X =
[      1, -12/11,      0,     -2]
[     -2,      1,      1,      2]
[      1, -14/11,     -2,     -2]
[      0,   7/11,      1,      1]
D =
[ -16,   0,   0,   0]
[   0,  -4,   0,   0]
[   0,   0,  -8,   0]
[   0,   0,   0, -12]
>> inv(X)*A*X
ans =
[ -16,   0,   0,   0]
[   0,  -4,   0,   0]
[   0,   0,  -8,   0]
[   0,   0,   0, -12]
```

Die Anweisung `J = jordan(A)` berechnet die JORDAN-Matrix von $\boldsymbol{A}$ und `[V,J] = jordan(A)` zusätzlich die Ähnlichkeitstransformation $\boldsymbol{V}$. Die Spalten von $\boldsymbol{V}$ sind die verallgemeinerten Eigenvektoren von $\boldsymbol{A}$.

```
>> 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]
```

41.15. 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. Zum Beispiel hat die Differenzialgleichung $y' = ay$ die allgemeine Lösung $y(t) = ce^{at}$. In MATLAB erhält man diese wie folgt:

```
>> dsolve('Dy = a*y')
ans =
exp(a*t)*C1
```

Mit der MAPLE-Funktion `pdsolve` kann man einfache partielle Differenzialgleichungen lösen. Als Beispiel betrachten wir die eindimensionale homogene Wellengleichung

$$u_{tt} - c^2 u_{xx} = 0 \qquad (c \neq 0).$$

Dies ist eine lineare partielle 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)
u(x,t) = _F1(t*c+x)+_F2(t*c-x)
```

Hierbei sind `_F1` und `_F2` zwei beliebige (differenzierbare) Funktionen.

41.16. 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 periodisch 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

$$\hat{f}(\omega) = \int_{\mathbf{R}} f(t) e^{-i\omega t} dt.$$

Die Abbildung $f \to \hat{f}$ 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 Zeitbereich oder Originalbereich in den Frequenzbereich oder Bildbereich 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-Transformation $\hat{f}(\omega)$ berechnet werden. Es ergibt sich

$$\begin{aligned} \hat{f}(\omega) &= \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt = \int_{-T}^{T} e^{-i\omega t} dt \\ &= \left[\frac{e^{-i\omega t}}{-i\omega} \right]_{-T}^{T} = \frac{1}{-i\omega} (e^{-iT\omega} - e^{iT\omega}) \\ &= \frac{2}{\omega} \frac{1}{2i} \left(e^{iT\omega} - e^{-iT\omega} \right) \\ &= \frac{2\sin(\omega T)}{\omega}. \end{aligned}$$

Die nachfolgenden Anweisungen bestätigen die Rechnung.

```
>> syms t T
>> Rechteck = heaviside(t+T)-...
heaviside(t-T);
>> fourier(Rechteck)
ans =
2/w*sin(T*w)
```

Die nachfolgenden Anweisungen geben eine zweite Möglichkeit; 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 nachfolgende Beispiel zeigt, dass die FOURIER-Transformierte $\hat{f}(\omega)$ einer Funktion $f(t)$ im allgmeinen eine komplexwertige Funktion ist. Der Graph einer komplexwertigen Funktion liegt im $\mathbf{R}^4$ und ist somit nicht direkt darstellbar. Entweder man zeichnet den Real- und Imaginärteil getrennt oder man zerlegt die komplexe Funktion in Betrag und Phase.

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-Transformation $\hat{f}(\omega)$ berechnet werden. Es ergibt sich

$$\begin{aligned} \hat{f}(\omega) &= \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt \\ &= \int_0^{\infty} ae^{-bt}e^{-i\omega t}dt = a\int_0^{\infty} e^{-(b+i\omega)}dt \\ &= a\left[\frac{e^{-(b+i\omega)t}}{-(b+i\omega)}\right]_{t=0}^{t=\infty} = \frac{a}{b+i\omega}. \end{aligned}$$

Die nachfolgenden 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 71 (Fourier-Transformation) 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

$$\begin{aligned}
\hat{f}(\xi) &= \int_{\mathbf{R}} e^{-|x|} e^{-ix\xi} dx \\
&= \int_0^\infty e^{-|x|} e^{-ix\xi} dx + \int_{-\infty}^0 e^{-|x|} e^{-ix\xi} dx \\
&= \int_0^\infty e^{-x} e^{-ix\xi} dx + \int_0^\infty e^{-x} e^{ix\xi} dx \\
&= \int_0^\infty e^{x(-1-i\xi)} dx + \int_0^\infty e^{x(-1+i\xi)} dx \\
&= \lim_{r\to\infty} \left[\frac{e^{-x(1+i\xi)}}{-1-i\xi} + \frac{e^{-x(1-i\xi)}}{-1+i\xi} \right]_{x=0}^{x=r} \\
&= -\frac{1}{-1-i\xi} - \frac{1}{-1+i\xi} \\
&= \frac{2}{1+\xi^2}.
\end{aligned}$$

Wir bestätigen dies symbolisch

```
>> syms x
>> f = exp(-abs(x));
>> fhat = fourier(f)
fhat =
2/(1+w^2)
```

Beachten Sie, dass die unabhängige Variable der FOURIER-Transformierten w ist. Die Abbildung 27 zeigt die Funktion f und ihre FOURIER-Transformierte $\hat{f}$.

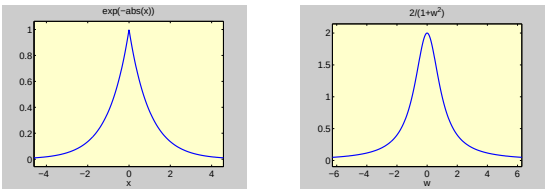

Abbildung 27: Zur stetigen FOURIER-Transformation

41.17. 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`).

41.18. Überblick über alle symbolischen Funktionen

Mit `help symbolic` (`doc symbolic`) erhalten Sie einen Überblick über alle Funktionen der *Symbolic Math Toolbox*.

42. WWW-Seiten

Die Firma *MathWorks* kann über das World Wide Web (WWW) erreicht werden. Die URL-Adresse[2] 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 findet man 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

`www.mathworks.de/matlabcentral`

zur Verfügung.

Programmiertipps stehen unter

`www.mathworks.de/gallery/tips/`.

In dieser Beschreibung findet Sie auf der letzten Seite weitere Adressen, um weitere Infos über Realease Notes, Function Reference, Newsgroup-Adressen, Suchadressen nach Online-Quellen, Digest-Infos, News & Notes, Dokumentationen (Handbücher), Beispielsammlung, usw zu bekommen. 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

`www.mit.edu/~pwb/cssm`.

[2] URL: Uniform Resource Locator.

43. Handbücher

Die Handbücher [5], [6] und [7] sind als PDF-Files unter

`www.mathworks.de/access/helpdesk/help/techdoc`

erhältlich.

44. Weitere Übungsaufgaben

Aufgabe 72 (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 73 (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
```

(a)
```
>> simplify(a^m*a^n)
ans =
a^(m+n)
```

(b)
```
>> simplify(a^m/a^n)
ans =
a^(m-n)
```

Aufgabe 74 (Rechnen)

Berechnen Sie den Ausdruck

$$\frac{\sqrt{16} + \cos(\pi/3)}{\sqrt[3]{8} + 4}$$

symbolisch und numerisch.
Lösung: Symbolische Rechnung:
Sowohl

```
>> sym((sqrt(16)+cos(pi/3))...
/(8^(1/3)+4))
```

also auch

```
>> (sym(sqrt(16)) + sym(cos(pi/...
3)))/sym(8^(1/3)+4)
```

liefert

```
ans =
3/4
```

Numerische Rechnung:

```
>> (sqrt(16)+cos(pi/3))/(8^(1/3)+4)
ans =
    0.7500
```

Aufgabe 75 (Vereinfachen)

Vereinfachen Sie den Ausdruck

$$\frac{x^2 + 2xy + y^2}{x^2 - y^2}$$

Lösung: Mit

```
>> syms x y
>> simplify((x^2-2*x*y+y^2)/...
(x^2-y^2))
```

erhält man

```
ans =
(-y+x)/(x+y)
```

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 76 (Multiplikation)
Multiplizieren Sie

$$(x^2 + x + 1) \cdot (x^3 - x^2 + 1).$$

Lösung: Mit

```
>> syms x
>> collect( (x^2+x+1)*(x^3-x^2+1))
ans =
x^5+x+1
```

sieht man, dass

$$(x^2 + x + 1) \cdot (x^3 - x^2 + 1) \; = \; x^5 + x + 1$$

ist.

Aufgabe 77 (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

$$(1 + x)^4 \; = \; x^4 + 4x^3 + 6x^2 + 4x + 1$$

ist.

Aufgabe 78 (Faktorisieren)
Faktorisieren Sie das Polynom

$$x^6 + x^4 - x^2 - 1.$$

Lösung: Mit

```
>> syms x
>> factor(x^6+x^4-x^2-1)
ans =
(x-1)*(1+x)*(1+x^2)^2
```

sieht man, dass

$$(x^6+x^4-x^2-1) \; = \; (x-1)\cdot(x+1)\cdot(x^2+1)^2$$

ist.

Aufgabe 79 (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 80 (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
```

(d)
```
>> diff(a^x)
ans =
a^x*log(a)

>> int(a^x)
ans =
1/log(a)*a^x
```

Damit gilt:

$f(x)$	$f'(x)$	$\int f(x)dx$
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	$\ln(a)a^x$	$a^x/\ln(a)$

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 MATLAB 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 MATLAB 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.

Literatur

Die Literaturangaben sind alphabetisch nach den Namen der Autoren sortiert.

[1] Dongarra, J., Bunch, J., Moler, C., Stewart, G.: *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.

[2] Golub, G., van Loan, C.: *Matrix Computations*. The Johns Hopkins University Press, 3. Auflage, 1996.

[3] Gramlich, G., Werner, W.: *Numerische Mathematik mit* Matlab. dpunkt.verlag, 2000.

[4] Higham, D., Higham, N.: Matlab *Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[5] Matlab: *Graphics*. The MathWorks, Natick, MA, USA, online verfügbar, 2004.

[6] Matlab: *Mathematics*. The MathWorks, Natick, MA, USA, online verfügbar, 2004.

[7] Matlab: *Programming*. The MathWorks, Natick, MA, USA, online verfügbar, 2004.

[8] Smith, B., Boyle, J., Dongarra, J., Garbow, B., Ikebe, Y., Kleme, V., Moler, C.: *Matrix Eigensystem Routines*-Eispack *Guide*. Springer-Verlag, 1976.

Stichwortverzeichnis

E

F

G

H

I

