Projektpraktikum zu der Vorlesung Computergestützte Ingenieurmathematik

Dr.-Ing. S. Bieder

(Prof. Dr.-Ing. A. Czylwik)

Dr. Stefan Bieder
Raum: BA 245, Tel: +49-203-37-92944
Email: bieder@nts.uni-due.de
Fachgebiet Nachrichtentechnische Systeme
Universität Duisburg-Essen

Stand: 20.06.2023

Inhaltsverzeichnis

Chapter 1

Einleitung in Matlab

1.1 Was ist Matlab?

Matlab[1] ist eine leistungsstarke Sprache für technische Berechnungen. Es bindet die Berechnung, Programmierung und Visualisierung in eine benutzerfreundliche Umgebung ein, in der Probleme und Lösungen in einer einfach zu verstehenden mathematischen Schreibweise ausgedrückt werden.
Matlab ist ein interaktives System, dessen grundlegendes Datenelement ein *Array* ist, welches keinerlei Dimensionierung benötigt. Das erlaubt dem Benutzer eine Menge technischer Probleme, insbesondere diese mit Matrix und Vektoroperationen, in kürzerer Zeit zu lösen, als in einer skalaren nicht-interaktiven Sprache wie C oder Fortan.
Matlab unterstützt eine Reihe von anwendungsspezifischen Lösungen welche *Toolboxen* genannt werden. Für die meisten Matlab-User ist es sehr wichtig, dass Toolboxen das Lernen und das Anwenden spezialisierter Technologien erlauben. Diese Toolboxen sind umfassende Sammlungen von Matlab-Functionen, sogenannte *M-files*, welche die Matlab-Umgebung erweitern, um bestimmte Arten von Problemen zu lösen.
Matlab ist ein Matrix-basiertes Programier-Tool. Obwohl Matrizen häufig nicht explizit Dimensioniert werden müssen, hat der Nutzer sorgfältig auf die Matrix-Dimensionen zu achten. Wenn nicht anders definiert, weist die Standard-Matrix zwei Dimensionen `n x m` auf. Spaltenvektoren und Reihenvektoren werden konsequenterweise durch `n x 1` und `1 x n` dargestellt. Matlab Operationen können in folgende Arten unterteilt werden:

- arithmetische und logische Operationen,
- mathematische Funktionen,
- grafische Funktionen, und
- Eingabe/Ausgabe Operationen

In den folgenden Kapiteln werden einzelne Elemente der Matlab-Operationen im Detail erklärt.

[1] Matlab is a registered trademark of The MathWorks, Inc.

1.2 Ausdrücke

Wie die meisten anderen Programmiersprachen unterstützt MATLAB mathematische Ausdrücke, aber im Gegensatz zu den meisten Programmiersprachen beinhalten diese vollständige Matrizen. Die Bausteine aus denen Ausdrücke bestehen sind:

- Variablen
- Zahlen
- Operatoren
- Funktionen

1.2.1 Variablen

MATLAB benötigt keine Typen- oder Dimensionsdeklaration. Wenn ein neuer Variablenname eingeführt wird, wird automatisch die Variable erstellt und entsprechender Speicher zugeteilt. Existiert die Variable bereits, ändert MATLAB dessen Inhalt und teilt, falls nötig, neuen Speicher zu. Zum Beispiel erstellt

```
>> books = 10
```

eine 1-mal-1 Matrix mit dem Namen `books` und speichert den Wert 10 in deren einziges Element. In dem obigen Ausdruck stellt `>>` die MATLAB-Eingabe dar, in welcher der Befehl eingegeben werden kann.
Variablennamen bestehen aus einem String, welcher mit einem Buchstaben, gefolgt von einer beliebigen Anzahl von Buchstaben, Ziffern oder Unterstrichen, beginnt. MATLAB beachtet hierbei die Groß- und Kleinschreibung. `A` und `a` sind nicht dieselbe Variable. Um die einer Variablen zugewiesenen Matrix zu sehen, geben sie einfach den Variablennamen ein.

1.2.2 Zahlen

MATLAB verwendet die konventionelle Dezimalschreibweise. Ein Dezimalpunkt oder ein vorangestelltes Plus- oder Minuszeichen sind optional. Die wissenschaftliche Notation verwendet den Buchstaben e, um den Zehnerpotenz Skalierungsfaktor anzugeben. Imaginäre Zahlen verwenden entweder `i` oder `j` als Zusatz. Einige Beispiele von erlaubten Zahlen sind:

```
7 -55 0.0041 9.657838 6.10220e-10 7.03352e21 2i -2.71828j 2e3i 2.5+1.7j
```

1.2.3 Operatoren

Ausdrücke verwenden die bekannten arithmetischen Operatoren und Präzedenzregeln. Einige Beispiele sind:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
'	komplex konjugiert transponiert
( )	Klammern zur Bestimmung der Berechnungsreihenfolge.

1.2.4 Funktionen

MATLAB unterstützt eine große Anzahl von elementaren mathematischen Funktionen, einschließlich `sin`, `sqrt`, `exp` und `abs`. Das Lösen der Quadratwurzel oder des Logarithmus einer negativen Zahl führt nicht zu einer Fehlermeldung; das entsprechende *komplexe* Ergebnis wird automatisch berechnet. MATLAB verfügt weiterhin auch über eine Menge weitergehender mathematischer Funktionen, wie Bessel und Gamma Funktionen. Viele dieser Funktionen akzeptieren komplexe Argumente. Für eine Liste der elementaren mathematischen Funktionen schreiben sie:

```
>> help elfun
```

Einige der Funktionen, wie `sqrt` und `sin` sind "build-in". Das bedeutet, dass sie ein fester Teil des MATLAB-Kern und damit sehr effizient sind. Der Nachteil hierbei ist, dass Einzelheiten der Berechnung nicht leicht einzusehen sind. Andere Funktionen, wie `gamma`, sind in sogenannten *M-files* implementiert. Man kann den Code einsehen und, wenn gewünscht, modifizieren.

1.3 Umgang mit Matrizen

MATLAB wurde hauptsächlich für den Umgang mit Matrizen ausgelegt. In MATLAB ist eine Matrix eine rechteckiges Array von Zahlen. Ein Skalar kann als eine 1-mal-1 Matrix und Vektoren als Matrizen mit nur einer Spalte oder Zeile interpretiert werden. MATLAB verfügt noch über weitere Möglichkeiten numerische und nicht-numerische Daten zu speichen, aber zu Beginn im Umgang mit MATLAB ist es üblicherweise am besten sich alles als Matrix vorzustellen. Die Operationen in MATLAB sind so natürlich wie möglich ausgelegt. Wo andere Programmiersprachen nur das Arbeiten mit einzelnen Zahlen erlaubt, erlaubt MATLAB das schnelle und einfache Arbeiten mit Matrizen.

1.3.1 Eingabe von Matrizen und Adressierung von Elementen

Die Elemente einer Matrix müssen einzeln in eine Liste eingetragen werden, in welcher die Elemente einer Reihe durch Kommas oder Leerzeichen und die Reihen selbst durch Semikolons voneinander zu trennen sind. Die gesamte Liste muss mit eckigen Klammern umgeben sein. Zum Beispiel:

```
>> A = [1 2 3; 8 6 4; 3 6 9]
```

Nach dem Drücken von „Enter“ zeigt MATLAB die Zahlen in der Befehlszeile

```
>> A =  1 2 3
        8 6 4
        3 6 9
```

Die Adressierung eines Elements einer Matrix ist sehr einfach. Das `n`-te Element von der `m`-ten Spalte in der Matrix `A` ist `A(n,m)`. Die Eingabe von

```
>> A(1,3) + A(2,1) + A(3,2)
```

berechntet die Antwort (*ans*wer)

```
ans = 17
```

Das `k`-te bis `1`-te Element von der `m`-ten bis zur `n`-ten Spalte kann mit `A(k:1,m:n)` adressiert werden, z.B.

```
>> A(2:3,1:2)

ans = 8 6
      3 6
```

Weitere Beispiele:

```
>> A(1,1:2)
```

adressiert die ersten zwei Elemente von der ersten Reihe.

```
ans = 1 2

>> A(:,2)
```

adressiert alle Elemente von der zweiten Spalte.

```
ans = 2
      6
      6
```

1.3.2 Matrizen erstellen

Es gibt mehrere Möglichkeiten Matrizen zu erstellen. Die explizite Zuordnung von Elementen wurde in dem Abschnitt oben erläutert. Um einen Reihenvektor mit 101 äquidistanten Werten von 0 bis π zu erstellen, wäre diese Methode sehr ineffizient. Deshalb werden nachfolgend zwei weitere Methoden vorgestellt:

```
>> x = linspace(0, pi, 101)
```

oder

```
>> x = (0:0.01:1)*pi
```

Im ersten Fall wird die MATLAB *Funktion* `linspace` verwendet, um `x` zu erstellen. Die Argumente der Funktion werden wie folgt beschrieben:

```
linspace(erster_wert, letzer_wert, anzahl_an_werten)
```

mit dem Standardwert `anzahl_an_werten = 100`.
Im zweiten Fall erstellt die Doppelpunkt-Notation `(0:0.01:1)` ein Array, welches mit 0 beginnt und mit einer Schrittweite von 0.01 bei 1 endet. Anschließend wird jedes Element des Arrays mit π multipliziert, um die gewünschten Werte in `x` zu erstellen.
Beide Methoden sind übliche Formen der Array-Erstellung in MATLAB. Während die Doppelpunkt-Noten erlaubt die Schrittweite zwischen zwei Elementen direkt zu bestimmen, jedoch nicht die Anzahl von Elementen, erlaubt die MATLAB *Funktion* `linspace` die Definition der Elementanzahl, aber nicht die Schrittweite zwischen den Elementen.
Die Doppelpunkt-Notation wird sehr of in MATLAB verwendet, weshalb sie näher betrachtet werden sollte.
`(erster_wert:schrittweite:letzer_wert)` erstellt ein Array das bei `erster_wert` beginnt und mit einer Schrittweite von `schrittweite`, welche auch negativ sein kann, bei `letzer_wert` endet. Z.B.:

```
>> v = (10:-2:0)

v =  10  8  6  4  2  0
```

Ist die Schrittweite gleich 1, dann ist die Verwendung optional:

```
>> w = (5:10)

w =  5  6  7  8  9  10
```

MATLAB bietet zusätzlich vier Funktionen, welche grundlegende Matrizen generieren: `zeros`, `ones`, `rand` and `randn`. Diese Funktion werden in Kapitel 5 behandelt.

1.3.3 Verkettung von Matrizen

Die Verkettung ist ein Prozess kleine Matrizen zu einer größeren Matrix zu vereinigen. Tatsache ist, die oben erstellte Matrix `A` wurde durch die Verkettung ihrer individuellen Elemente erstellt. Das paar eckiger Klammern [ ] ist der Verkettungsoperator. Für ein Beispiel beginnen wir mit der 3-mal-3 Matrix `A` und bilden

```
>> F = [A A+10; A*2 A*4].
```

Das Ergebnis ist eine 6-mal-6 Matrix, welches die vier Untermatrizen verbindet.

```
F = 1  2  3  11  12  13
    8  6  4  18  16  14
    3  6  9  13  16  19
    2  4  6   4   8  12
   16 12  8  32  24  16
    6 12 18  12  24  36
```

1.3.4 Löschen von Reihen und Spalten

Um Reihen oder Spalten einer Matrix zu löschen, wird ein paar eckiger Klammern verwendet. Zum Beispiel löscht

```
>> A(2,:) = []
```

die zweite Reihe von `A`.

```
A =  1  2  3
     3  6  9
```

Es ist nicht möglich ein einzelnes Element einer Matrix zu löschen, weil die Matrix anschließend keine Matrix mehr wäre. (Ausnahme: Vektoren, da hier das Löschen eines Elementes dem Löschen einer Reihe bzw. Spalte entspricht.)

1.3.5 Transponieren eines Arrays

Die Ausrichtung eines Arrays kann mit dem MATLAB "Transponieren"-Operator geändert werden:

```
>> a = 0:3

a =  0  1  2  3
```

```
>> b = a'

b =  0
     1
     2
     3
```

1.3.6 Skaler-Array Mathematik

Addition, Subtraktion, Multiplikation und Division mit einem *Skalar* führt die Operation auf alle Elemente eines Arrays aus:

```
>> c = [1 2 3 4; 5 6 7 8; 9 10 11 12]

c = 1  2  3  4
    5  6  7  8
    9 10 11 12
```

`>> 2*c-1` multipliziert jedes Element in c mit zwei und subtrahiert von jedem Element des Ergebnisses eine eins.

```
ans =  1  3  5  7
       9 11 13 15
      17 19 21 23
```

1.3.7 Array-Array Mathematik

When zwei Arrays die gleichen Dimensionen haben, was bedeutet, dass sie die gleiche Anzahl von Reihen und Spalten aufweisen, dann wird die Addition, Subtraktion, Multiplikation und Division in MATLAB elementweise ausgeführt.

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

d = 1  2  3
    4  5  6

>> e = [2 2 2; 3 3 3]

e =  2  2  2
     3  3  3

>> f = d+e % Addiert d zu e on an elementweise
```

```
f = 3  4  5
    7  8  9

>> g = 2*d-e % Multipliziert d mit zwei und subtrahiert e von dem Ergebnis

g = 0  2  4
    5  7  9
```

Die elementweise Multiplikation und Division funktionieren ähnlich, jedoch unterscheidet sich die Notation:

```
>> h = d.*e

h =  2  4  6
    12 15 18
```

Die elementweise Multiplikation verwendet das punkt-mulitplikations Symbol .*, die elementweise Division entweder `./` oder `.\`

```
>> d./e

ans = 0.500  1.000  1.500
      1.333  1.666  2.000

>> e.\d

ans = 0.500  1.000  1.500
      1.333  1.666  2.000
```

In beiden Fällen werden die Element des Arrays vor dem Schrägstrich durch die Elemente des Arrays hinter dem Schrägstrich geteilt. Um die Matrizenmultiplikation zu berechnen, darf nur das Sternchen * verwendet werden. Zum Beispiel:

```
>> C = A * B
```

Hierfür muss die *Anzahl der Spalten* von `A` gleich der *Anzahl der Reihen* von `B` sein.

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

A =  1  2  3
     4  5  6

>> B = [1 2; 3 4; 5 6]

B =  1  2
     3  4
     5  6

>> C = A * B

C =  22  28
     49  64
```

Übungen

Variablen erzeugen

1.1) Definieren Sie eine skalare Variable `a` und weisen Sie der Variablen den Wert $(-\frac{1}{2} + 0.2\,\mathrm{j})$ zu.

1.2) Definieren Sie einen Spaltenvektor `b` mit 5 Elementen, die linear von 80 bis 60 absteigen.

1.3) Welche Ergebnisse liefern die Befehle: `length(b)` und `size(a)` für die Definitionen in 1.1 und 1.2?

1.4) Definieren Sie den Zeilenvektor `c` mit

$$\mathbf{c} = \begin{pmatrix} 1 & -1 & 1 \end{pmatrix} .$$

1.5) Ergänzen Sie zu dem Vektor `c` eine zweite Zeile, so dass die Matrix `C`

$$\mathbf{C} = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

entsteht.

1.6) Fügen Sie der Matrix $\mathbf{C}$ weitere Spalten hinzu, so dass sich die Matrix `D`

$$\mathbf{D} = \begin{pmatrix} 1 & -1 & 1 & 2 & -2 \\ 0 & 1 & 1 & 2 & 2 \end{pmatrix}$$

ergibt.

Matrixoperationen

Gegeben sind die Matrix $\mathbf{A} = \begin{pmatrix} 3 & 2 \\ 1 & 1 \\ 2 & 3 \end{pmatrix}$ und der Spaltenvektor $\mathbf{b} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$.

Erzeugen Sie die Variablen `A` und `b` in MATLAB. Bestimmen und überprüfen Sie die Ergebnisse folgender MATLAB-Befehle:

2.1) `A*b'`

2.2) `A*b`

2.3) `(A*b)'`

2.4) `b'*A'`

2.5) `ones(3,2).*A`

2.6) `A*zeros(2,5)`

Chapter 2

Grundlagen des Plottens mit Hilfe von Matlab

2.1 Zwei-Dimensionale Grafiken

Grafiken (in 2-D) werden mit dem `plot` -Befehl erstellt. In seiner einfachsten Form verwendet er ein einzelnes Vektorargument **y**, wie in `plot(y)`. In diesem Fall werden die Elemente von **y** über ihre Indizes aufgetragen. Zum Beispiel plottet

```
plot(rand(1, 20))
```

20 Zufallszahlen im ganzzahligen Intervall von 1 bis 20. Aufeinanderfolgende Punkte werden hierbei miteinander verbunden.

Gerade Linien werden geplottet, indem man die x- und y-Koordinate der Endpunkte in zwei Vektoren angibt. Um zum Beispiel eine Linie zwischen zwei Punkten mit den kartesischen Koordinaten (0, 1) und (4, 3) zu plotten, verwendet man den Befehl:

```
plot([0 4], [1 3])
```

Übung 1

- Plotten Sie Linien, welche die folgenden Punkte miteinander verbinden: (0, 0), (1, 1), (2, 0) , (1,-1) und (0, 0).

Wenn **x** und **y** Vektoren mit der selben Länge sind, öffnet der Befehl `plot(x,y)` ein Grafikfenster und plottet die Elemente von **y** über die Elemente von **x**. Zum Beispiel,

```
x = -4 : 0.01 : 4 ;
y = sin(x) ;
plot(x, y)
```

Der Vektor **x** ist Teil eines Definitionsbereiches mit der Schrittweise 0.01 und **y** ist ein Vektor, der die Werte eines Sinus an den Knoten dieses Bereiches besitzt (Zur Erinnerung: `sin(x)` operiert eintragsweise). Beim Plotten einer Kurve verbindet die Plot-Routine alle aufeinanderfolgenden Punkte im gegebenen Bereich mit Linienelementen. Aus diesem Grund sollte die Schrittweite ausreichend klein gewählt werden, damit die Kurve möglichst glatt erscheint.

Übung 2

- Plotten Sie $y = \exp(-x^2)$ gegen x für x im Bereich von -10 bis 10.
  Wählen Sie eine geeignete Schrittweite.
- Plotten Sie $y = \sin(2\pi f t)$ gegen t für t im Bereich von -2 bis 2 und $f = 2$.
  Wählen Sie wieder eine geeignete Schrittweite.

2.1.1 Mehrere Plots auf der selben Achse

Der einfachste Weg ist es "hold" zu verwenden, um den aktuellen Plot auf den Achsen zu behalten. Alle weiteren Plots werden den Achsen hinzugefügt bis "hold" aufgehoben wird. Entweder mit "hold off" oder mit "hold", welches den "hold" Status wechselt.
Der zweite Weg ist der Verwendung des plot-Befehles mit mehreren Argumenten. Zum Beispiel plottet

```
plot(x1, y1, x2, y2, x3, y3, ... )
```

den Vektor `y1` gegen den Vektor `x1`, den Vektor `y2` gegen den Vektor `x2`, etc. in einer gemeinsamen Darstellung. Der Vorteil dieser Methode liegt darin, dass die Vektorpaare unterschiedliche Längen haben können. MATLAB wählt automatisch eine andere Farbe für jedes Vektorpaar. Zum Beispiel:

```
x = -2*pi : 0.01 : 2*pi  ;
plot(x,cos(x),x,sin(x))
%Equivalent to this command is:
plot(x,cos(x))
hold on
plot(x,sin(x))
hold off
```

Übung 3

- Plotten Sie $x^2, x^3, x^4, \exp(-x^2)$ gemeinsam in einer Abbildung. Wählen Sie eine geeignete Schrittweite.
  Fügen Sie eine Legende hinzu, indem Sie den MATLAB-Befehl
  `legend('x^2','x^3','x^4','exp(-x^2)')` verwenden.

2.1.2 Linienstile, Markierungen und Farbe

Linienstil, Markierung und Farbe können für einen Plot mit Hilfe eines Textargumentes im plot-Befehl gewählt werden. Zum Beispiel verbindet

```
plot(x, y, '--')
```

die geplotteten Punkte mit gestrichelten Linien, während

```
plot(x, y, 'o')
```

Kreise auf die Datenpunkte zeichnet, ohne sie mit Linien zu verbinden. Man kann auch alle drei Eigenschaften definieren. Zum Beispiel plottet

```
plot(x,sin(x), x, cos(x), '--mo')
```

sin(x) mit dem Standardstil und cos(x) mit Kreisen, die mit Bindestrichen in Magenta verbunden sind. Die verfügbaren Farben sind mit den Symbolen c, m, y, k, r, g, b und w gekennzeichnet. Über das Kommando `help plot` kann die Hilfefunktion zum Plot mit einer Auflistung der standardmäßigen unterstützten Farben und Linienstilen aufgerufen werden.

2.1.3 Plotten von sich schnell ändernden mathematischen Funktionen

In allen bisherigen grafischen Beispielen, wurde die x-Koordinate der geplotteten Punkte gleichmäßig erhöht, z.B. $x = 0 : 0.01 : 4$. Ändert sich die geplottete Funktion in einigen Bereichen rapide, kann das uneffizient sein und sogar einen irreführenden Graphen ergeben. Zum Beispiel:

```
x = 0.01:.001:.1;
plot(x, sin(1./x))
```

MATLAB hat eine Funktion `fplot`, welche einen eleganteren Ansatz verwendet. Während die oberen Methode $\sin(1/x)$ mit gleichbleibenden Abständen auswertet, wertet `fplot` die Funktion in den Regionen häufiger aus, wo sie sich stärker ändert. Die Funktion wird wie folgt verwendet:

```
fplot(@(x) sin(1./x),[0.01 0.1]) % @(x)... defines an Anonymous Function
```

Übung 4

- Zeichnen Sie die Funktion $y = \frac{1}{x}$ im Bereich $x \in [-1, 1]$ unter Verwendung von `fplot`.

2.2 Drei-Dimensionale Graphen

MATLAB's primäre Befehle zu Erstellung von drei-dimensionalen Graphen von numerisch definierten Funktionen sind `plot3, mesh, surf,` und `surfl`.

2.2.1 Kurvenplots

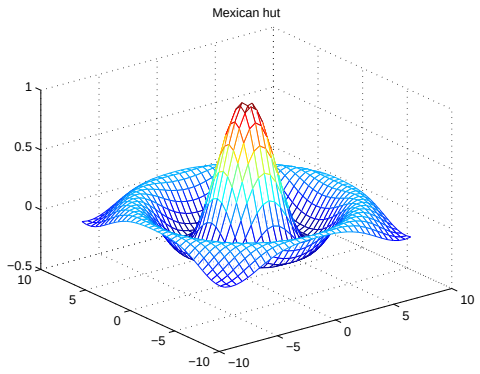
Analog zu den Plots in zwei Dimensionen, erstellt der Befehl `plot3` Kurven im drei-dimensionalen Raum. Sind x,y und z drei Vektoren der selben Länge, dann erstellt der Befehl `plot3(x,y,z)` einen perspektivischen Plot von der stückweise linearen Kurve durch Durchlaufen der Koordinatenpunkte aus den Vektoren x,y und z. Diese Vektoren sind normalerweise parametrisch definiert. Zum Beispiel:

```
t = 0.01 : 0.01 : 20*pi ;
x = cos(t) ;
y = sin(t) ;
z = t.^3 ;
plot3(x, y, z) %produces a helix that is compressed near the x-y plane
```

2.2.2 Maschen- und Oberflächenplots

Der `mesh`-Befehl stellt drei-dimensionale Gitternetzflächen auf. Ähnlich dazu erzeugt der `surf`-Befehl drei-dimensionale Oberflächenplots. Die folgende MATLAB M-Datei erstellt unter Verwendung des `mesh`-Befehles einen mexikanischen Hut.

```
% Mexican hat as an example for 3D plot.
[x y] = meshgrid (-8: 0.5 : 8) ;
r = sqrt(x.^2 + y.^2) ;
z = sin(r)./r ;
mesh(x,y,z)
title('Mexican hat')
```



Das folgende Skript zeigt, wie man die Funktion

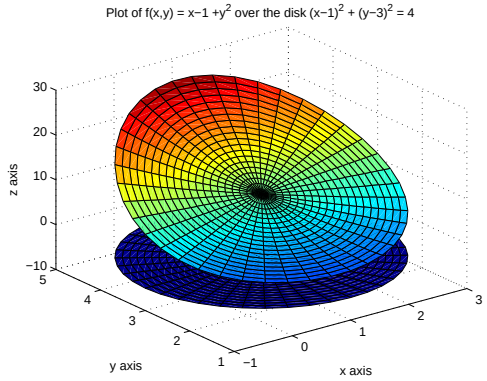
$$f(x, y) = x - 1 + y^2$$

über die Scheibe

$$(x - 1)^2 + (y - 3)^2 = 4$$

aufträgt.

```
% Example for 3D Plot from the book 'A {\sc Matlab} comapanion for a multivariable calcu
% by J. Cooper, University of Maryland
% First make a meshgrid in r, theta coordinates

r = linspace(0,2,21);
theta = linspace(0, 2*pi, 41);
[R,TH] = meshgrid(r,theta)
% Now convert into a curvilinear meshgrid in x,y
% coordinates
X = 1 + R.*cos(TH);
Y = 3 + R.*sin(TH);
Z = X-1 + Y.^2;
surf(X,Y,Z)
hold on
% add the plane z = -5 with the curvilinear
% meshgrid
surf(X,Y,-5+0*Z)
hold off
```



Übung 5

- Plotten Sie eine gauß'sche Wahrscheinlichkeitsdichtefunktion $f_{x,y}(x,y)$ mit dem Mittelwert null und der Varianz eins im Bereich $x \in [-5,5], y \in [-5,5]$:

$$f_{x,y}(x,y) = \frac{1}{2\pi} \mathrm{e}^{-\frac{1}{2}(x^2+y^2)}$$

  Verwenden Sie zum Plotten des Graphen die Befehle `mesh`, `surf` und `contour`.
  Wählen Sie geeignete Schrittweiten.

Übung 6

- Plotten Sie die Funktion

$$g(r,\theta) = (r/2)\sin(\theta) + r^3 \cos(3\theta)$$

  über den im Ursprung zentrierten Einheitskreis.
  Wählen Sie geeignete Schrittweiten.

2.2.3 Mehrere Plots in einer Darstellung: subplot

Mit der `subplot`-Funktion kann man sich mehrere Plots in dem selben Fenster anzeigen lassen. Der Ausdruck `subplot(m,n,p)` unterteilt das Anzeigefenster in $m \times n$ große Achsenabschnitte und wählt den p-ten Abschnitt für den aktuellen Plot (Reihenweise durchnummeriert, beginnend von links oben).

Übung 7

- Plotten Sie in einem Bild die folgenden Potenzen in von x in vier Subplots im Bereich $x \in [-1,1]$:
  erste Zeile, erste Spalte: $y(x) = x$, erste Zeile, zweite Spalte: $y(x) = x^2$,
  zweite Zeile, erste Spalte: $y(x) = x^3$, zweite Zeile, zweite Spalte: $y(x) = x^4$.
  Geben Sie den Achsen eine sinnvolle Beschriftung.
  Wählen Sie eine geeignete Schrittweite.

Chapter 3

Funktionen

Dieses Kapitel untersucht wichtige Charakteristiken von Funktionen unter Verwendung von MATLAB.

3.1 Definitionsbereich, Wertebereich und Symmetrie

Betrachten Sie die Funktionen

$$f(x) = \frac{1}{\sqrt{1 - \frac{x^2}{4}}}$$
$$g(x) = \frac{1}{\sqrt{\left|1 - \frac{x^2}{4}\right|}}.$$

Übung 3.1

- Zeichnen Sie die Funktionen $f(x)$ und $g(x)$ für $-5 \leq x \leq 5$ mit eindeutigen Farben, Markierungen und Beschriftung in dieselbe Abbildung ein. Bestimmen Sie den Definitionsbereich, den Wertebereich und die Symmetriepunkte der Funktionen $f(x)$ und $g(x)$, so dass sie reellwertig sind.
- Beschriften Sie die x- und y-Achse mit den MATLAB-Befehlen

```
xlabel('x \rightarrow') ;
ylabel('f(x) , g(x) \rightarrow')
```

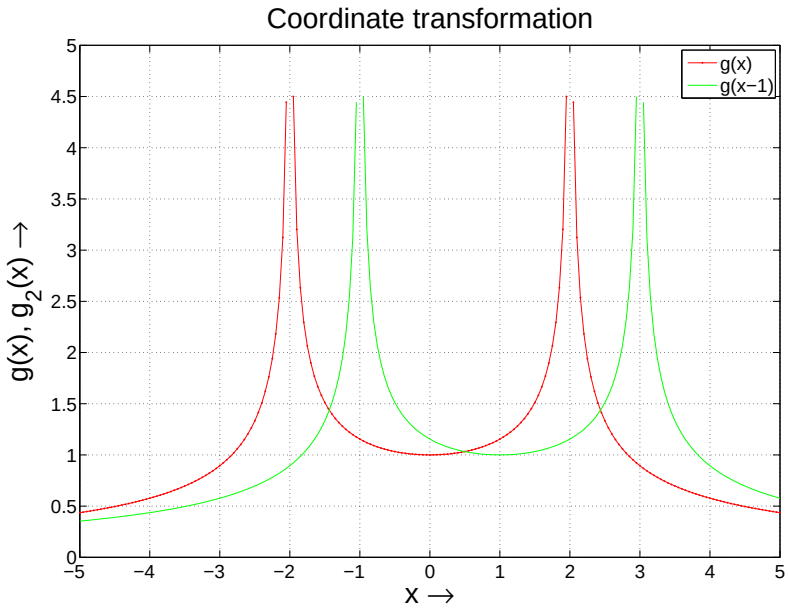
3.2 Koordinatentransformation

Betrachten Sie die Funktion $g(x)$ im Intervall $-5 \leq x \leq 5$:

```
clc; clear;close all
x = -5 : .05 : 5 ;
g_x = 1 ./ sqrt( abs(1 - 0.25*x.^2 ) ) ;
plot( x , g_x , '-r.' )
hold on;
```

Eine einfache Koordinatentransformation erhält man durch die Verschiebung von $g(x)$ zu $g_2(x) = g(x - x_0)$. Für $x_0 = 1$:

```
x_0  = 1;
g2_x = 1./sqrt(abs(1-0.25*(x-x_0).^2)) ;
plot(x,g2_x,'-g')
hold on;
```

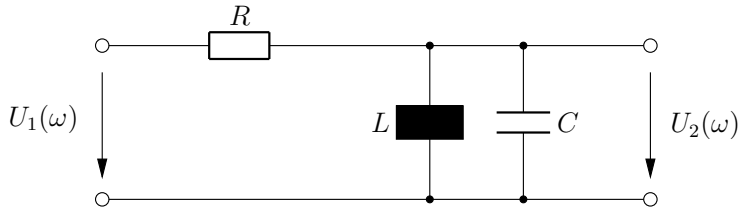


Übung 3.2

- Stellen Sie auf derselben Abbildung die Koordinatentransformation $g_3(x) = g(ax)$ für $a = 2$ dar (Skalierung der x-Achse).
- Stellen Sie auf derselben Abbildung die Koordinatentransformation $g_4(x) = g(ax - x_0)$ für $a = 2$ und $x_0 = 1$ dar.
- Erstellen Sie eine neue Abbildung mit der nicht-linearen Transformation $g_5(x) = g(x^2)$.
- Fügen Sie zu allen Abbildungen eine Legende hinzu.

3.3 Normierung von physikalischen Größen

Physikalische Größen werden normiert, um die Berechnung zu vereinfachen und um die Verwendung von physikalischen Einheiten von verschiedenen Variablen zu vernachlässigen. Betrachten Sie die Übertragungsfunktion des folgenden RLC Filters:



$$H(\omega) = \frac{U_2(\omega)}{U_1(\omega)} = \frac{\mathrm{j}\omega L/R}{1 - \omega^2 LC + \mathrm{j}\omega L/R}.$$

Das Betragsquadrat des RLC Filters, unter der Verwendung der technischen Frequenz f, ist gegeben durch:

$$|H(2\pi f)|^2 = \frac{(2\pi f L/R)^2}{\left[1 - (2\pi f)^2 LC\right]^2 + [2\pi f L/R]^2}.$$

Ausgedrückt:

f in kHz,
L in µH,
C in µF,
R in Ω

Ergibt:

$$|H(2\pi f)|^2 = \frac{\left(2\pi \frac{f}{\mathrm{kHz}} \frac{L/\mu\mathrm{H}}{R/\Omega} \cdot 10^{-3}\right)^2}{\left(1 - \left(2\pi \frac{f}{\mathrm{kHz}}\right)^2 \frac{L}{\mu\mathrm{H}} \frac{C}{\mu\mathrm{F}} \cdot 10^{-6}\right)^2 + \left(2\pi \frac{f}{\mathrm{kHz}} \frac{L/\mu\mathrm{H}}{R/\Omega} \cdot 10^{-3}\right)^2}.$$

Übung 3.3a

- Stellen Sie $|H(2\pi f)|^2$ im Bereich $0 \leq f \leq 20$ kHz für $L = 100$ µH, $C = 10$ µF und $R = 10\ \Omega$ mit den entsprechenden Beschriftungen der x- und y-Achse dar.

3.4 Lokale Minima und Maxima

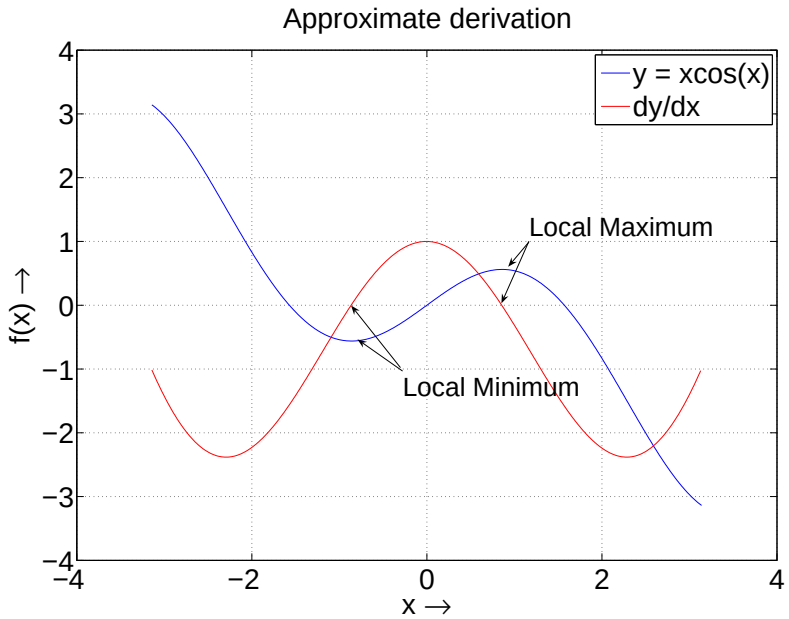
Für eine Funktion $y = f(x)$ werden die lokalen Minima und Maxima durch Lösen der Gleichung

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} = 0$$

bestimmt.

Eine graphische Näherung der Ableitung kann in MATLAB durch den `diff`-Befehl berechnet werden. Für einen Vektor $\mathbf{y}$ berechnet `diff(y)` die Differenz der benachbarten Elemente von $\mathbf{y}$. Als Beispiel wollen wir graphisch die lokalen Minima und Maxima der Funtkion $y(x) = x\cos(x)$ für $-\pi \leq x \leq \pi$ bestimmen:

```
clc; clear; close all
Delta = 0.01;
x = -pi : Delta : pi;
y = x .* cos( x );
plot(x,y);
grid on; hold on;
plot(x(1:end-1),diff(y)./diff(x),'r')
xlabel('x \rightarrow')
ylabel('f(x) \rightarrow')
title('Approximate derivation')
legend('y = xcos(x)', 'dy/dx')
```



Beachten Sie, dass der Befehl `diff(y)` die Dimension des Vektors $\mathbf{y}$ um ein Element reduziert. Aus diesem Grund wurde die Abbildung unter Verwendung des Befehls `x(1:end-1)` erstellt, welcher den Vektor $\mathbf{x}$ um ein Element reduziert.

Übung 3.3b

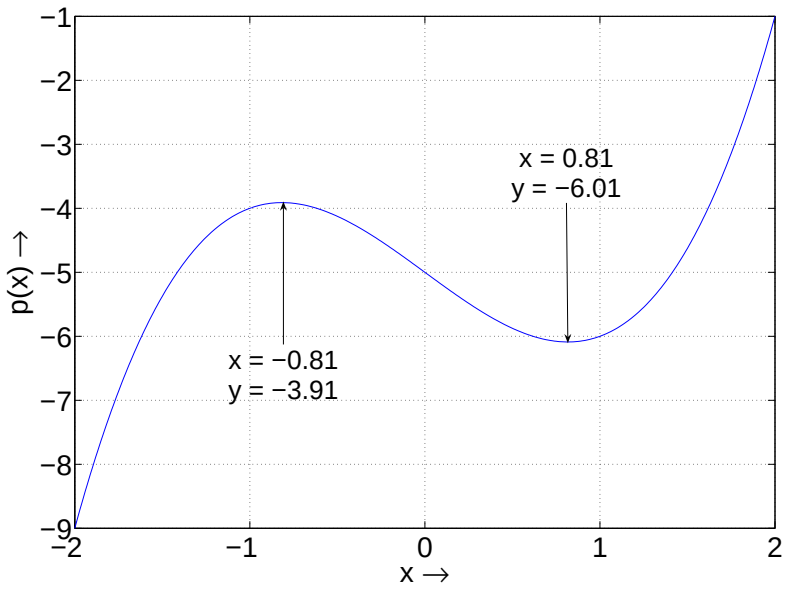
- Stellen Sie in der Abbildung aus Übung 3.3a zusätzlich die Ableitung von $|H(2\pi f)|^2$ für $0 \leq f \leq 20$ kHz, $L = 100$ µH, $C = 10$ µF und $R = 10\ \Omega$ mit den entsprechenden Beschriftungen der x- und y-Achse sowie entsprechender Legende in einem Diagramm dar.

Die Orte der lokalen Minima und Maxima können analytisch gelöst werden, falls die Funktion in Form eines Polynoms ausgedrückt werden kann. Die Vorgehensweise wird im Folgenden beschrieben.

Als Erstes wird in MATLAB ein Polynom durch einen Reihenvektor definiert, welcher die Koeffizienten des Polynoms enthält. Zum Beispiel wird das Polynom
$p(x) = x^3 - 2x - 5$ durch

```
p = [1 0 -2 -5];
```

dargestellt.



Die Ableitung $q(x) = \dfrac{\mathrm{d}p(x)}{\mathrm{d}x}$ wird mit dem MATLAB-Befehl `polyder` berechnet:

```
q = polyder(p);
```

Die Berechnung der Nullstellen von $q(x)$ liefert die Orte der lokalen Minima und Maxima. Hierfür wird der Befehl `roots` verwendet:

```
r = roots(q);
```

Als letztes werden die Werte der lokalen Minima und Maxima von $p(x)$ durch den Befehl `polyval` bestimmt:

```
polyval(p,r);
```

Komplexere Funktionen können oft durch das Verhältnix von zwei Polynomen $H(x) = \dfrac{B(x)}{A(x)}$ (wie zum Beispiel im Fall der Übertragungsfunktion der meisten linearen zeitinvarianten Systeme) dargestellt werden. Die Ableitung von $H(x)$ ist das Ergebnis des Verhältnisses von $Q(x)$ und $D(x)$:

$$\frac{\mathrm{d}H(x)}{\mathrm{d}x} = \frac{Q(x)}{D(x)}.$$

Die Polynome $Q(x)$ und $D(x)$ werden mit dem Befehl `polyder` wie folgt berechnet:

```
[Q,D] = polyder(B,A);
```

Zum Schluss können die Positionen der lokalen Minima und Maxima durch die Berechnung der Nullstellen von $Q(x)$ mit dem Befehl `roots` berechnet werden (siehe oben).

Übung 3.4

- Bestimmen Sie die lokalen Minima und Maxima der Funktion

$$|H(2\pi f)|^2 = \frac{\left(2\pi f \dfrac{L}{R}\right)^2}{\left(1 - (2\pi f)^2 LC\right)^2 + \left(2\pi f \dfrac{L}{R}\right)^2}.$$

  für $L = 100$ µH, $C = 10$ µF und $R = 10\ \Omega$. Vergleichen Sie das Ergebnis mit der Resonanzfrequenz $f_0 = \dfrac{1}{2\pi\sqrt{LC}}$.

3.5 Pol- und Nullstellen

Die Pol- und Nullstellen einer Funktion $H(x)$ erhält man durch die Berechnung der Nullstellen des Zählers und Nenners von $H(x)$, beziehungsweise:

$$H(x) = k\frac{(x - x_{\mathrm{N},1})\,(x - x_{\mathrm{N},2})\cdots}{(x - x_{\mathrm{P},1})\,(x - x_{\mathrm{P},2})\cdots}$$

Übung 3.5

- Zeichnen Sie das Pol- und Nullstellendiagramm der Systemfunktion des RLC Filters:

$$H_{\mathrm{L}}(p) = \frac{p \cdot \dfrac{L}{R}}{p^2 \cdot LC + p \cdot \dfrac{L}{R} + 1}$$

  für $L = 100$ µH, $C = 10$ µF und $R = 10\ \Omega$.

Chapter 4

Kurvenanpassung, Interpolation und Approximation von Funktionen mit Polynomen

Dieses Kapitel behandelt grundlegenden MATLAB Werkzeuge für die Kurvenanpassung, Interpolation und Approximation von Funktionen durch Polynome.

4.1 Polynomische Kurvenanpassung

Der MATLAB-Befehl `polyfit` findet die Koeffizienten eines Polynoms, welches mit minimalem quadratischen Fehler zu der dazugehörigen Datenreihe passt:

```
p = polyfit(x, y, n)
```

findet das best passendste Polynom `n`-ten Grades, welches sich den Datenpunkten **x** und **y** annähert.
x und **y** sind Vektoren, die die `x` und `y` Daten beinhalten, denen sich angenähert werden soll and `n` ist der Grad des Polynoms, welches ausgegeben wird. Betrachten Sie zum Beispiel die folgenden **x**-**y** Testdaten:

```
x = [1 2 3 4 5];
y = [5.5 43.1 128 290.7 498.4];
% A third degree polynomial that approximately fits the data is:
p = polyfit(x,y,3)
p =
    -0.1917 31.5821 -60.3262 35.3400
```

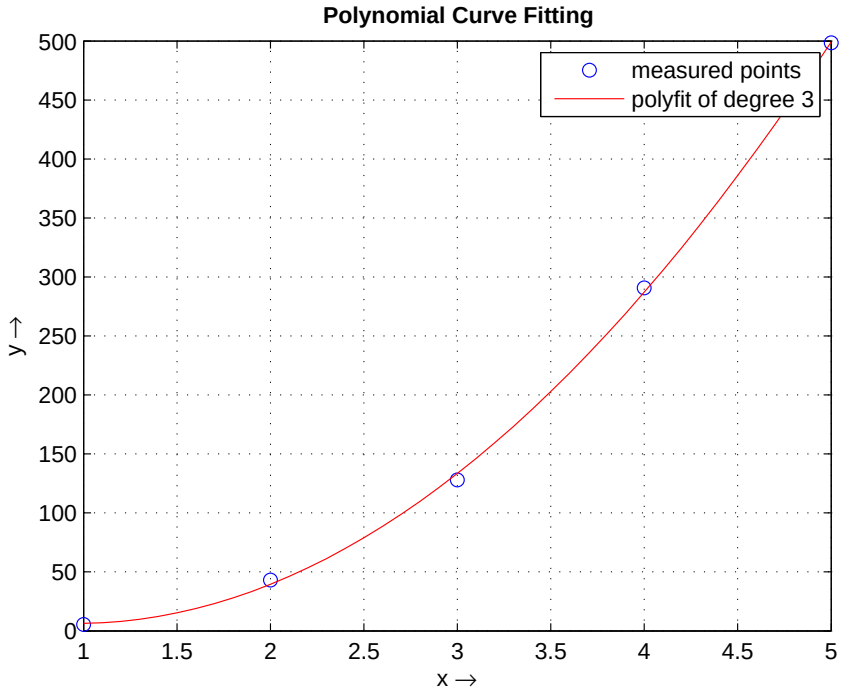
Lassen Sie uns nun zum Vergleich die Werte des mit `polyfit` geschätzten Polynoms über einen feineren Bereich berechnen und die geschätzten und echten Datenwerte in ein Diagramm plotten:

```
x2 = 1:.1:5;
y2 = polyval(p,x2);
plot(x,y,'o',x2,y2)
```

```
grid on
legend('measured points','polyfit of degree 3 ')
title('\bf{Polynomial Curve Fitting}')
```

Beachte: Der Befehl `polyval(p,x2)` berechnet das Polynom **p** für jedes Element des Vektors **x2**. Weitere Informationen finden Sie in der MATLAB-Hilfe.

Die folgende Abbildung zeigt die gemessenen Daten und die angenäherte Kurve.



Übung 4.1

Betrachten Sie die folgenden Testdaten:

```
x = [0 1 2 3 4 5]
y = [0 0.632 0.865 0.95 0.982 0.993]
```

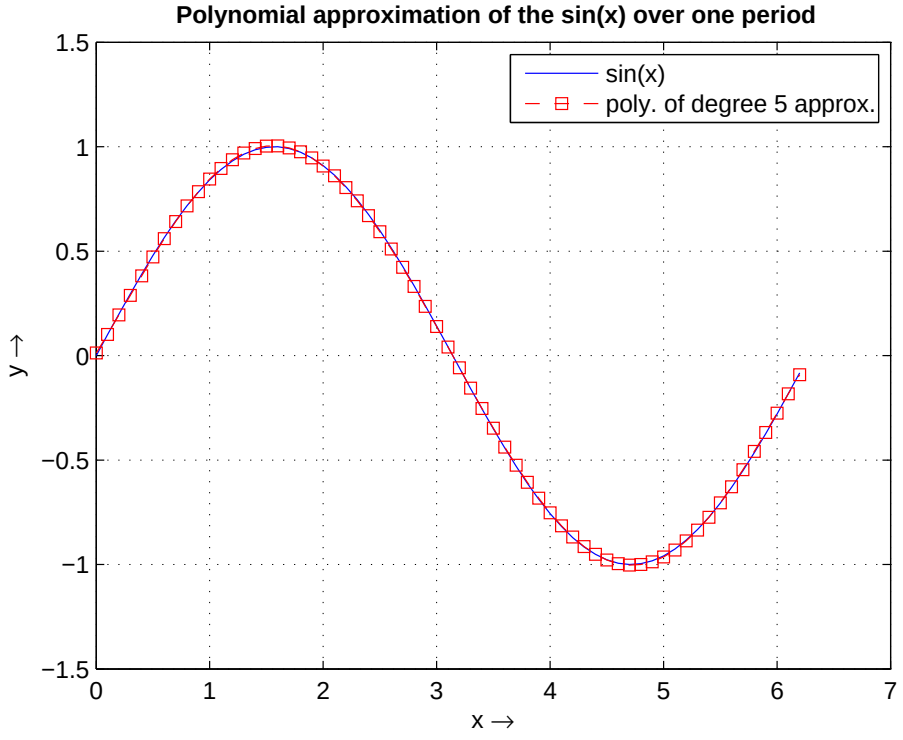
- Finden Sie ein Polynom fünften Grades, welches die Daten annähert. Berechnen Sie die Werte des geschätzten Polynoms im Bereich $0 \leq x \leq 5$ mit einer hochauflösenden Abtastperiode $\Delta x = 10^{-3}$) und plotten Sie zum Vergleich die geschätzten und realen Werte in ein Diagramm.

4.2 Polynominterpolation

Polynome sind, mit Hilfe der Taylorreihenentwicklung oder eines low-degree best-fit Polynoms durch Verwendung der `polyfit`-Funktion, leicht zu berechnende Näherungen von komplexeren Funktionen.

Beispiel:
Lassen Sie uns die Funktion $y = \sin(x)$ zwischen 0 und 2π durch ein Polynom fünften Grades annähern und plotten beide zum Vergleich in die selbe Abbildung.

```
% Example: function approximation using polynomial
% the given function is y = sin(x)
% the objective is to approximate the function between 0 and 2*pi using a
% polynomial of degree 5.

close all
x  = 0:0.1:2*pi;
y  = sin(x);
plot(x,y)
xlabel('x \rightarrow');
ylabel('y \rightarrow');
hold on
p  = polyfit(x,y,5);
xx = 0: 0.1: 2*pi;
yy = polyval(p,xx);
plot(xx,yy,'-sr')
legend('sin(x)', 'poly. of degree 5 approx.');
```



Übung 4.2

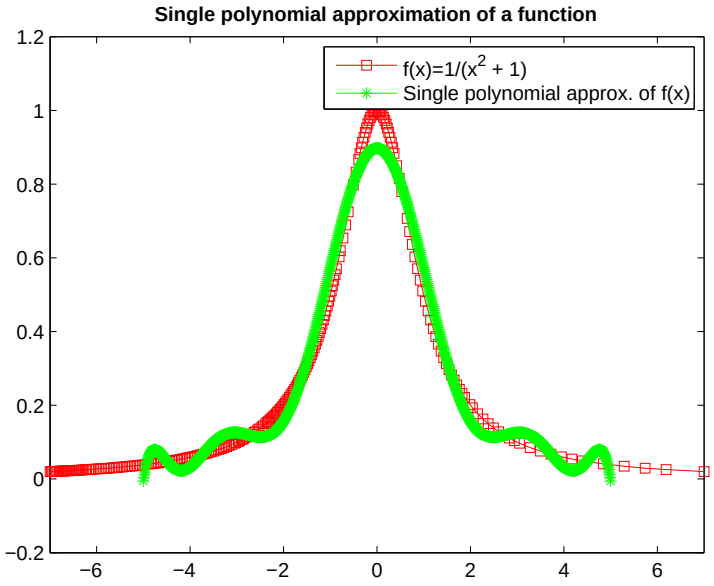
a) Tasten Sie die Funktion $y = \sin(x)$ im Bereich $0 \leq x < 2\pi$ mit der Abtastperiode $\Delta x_1 = 0.5$ ab. Bestimmen Sie das Polynom fünften Grades, das diese Abtastwerte mit dem geringsten mittleren quadratischen Fehler annähert.
Plotten Sie das annähernde Polynom und die Funktion $y = \sin(x)$ im Bereich $-2\pi \leq x < 4\pi$ für eine eine hochauflösende Abtastperiode $\Delta x_1 = 0.001$.

b) Führen Sie die obigen Schritte für eine Annäherung der Funktion $y = \sin(x)$ durch ein Polynom siebten Grades durch.

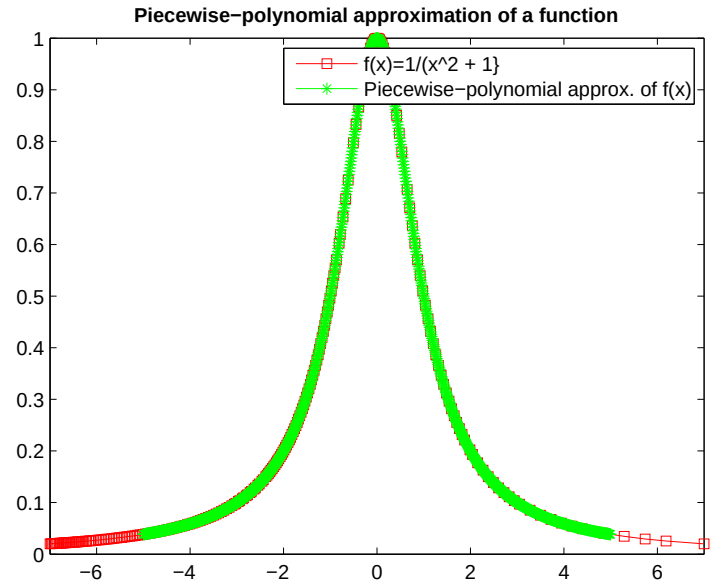
Eine schrittweise Polynominterpolation durch z.B. kubische *Splines* ist typischerweise genauer als eine Annäherung durch ein einzelnes hochgradiges Polynom. Das folgende Beispiel zeigt den Unterschied zwischen beiden.

```
% comparison of piecewise polynomial versus single high-degree polynomial
% interpolations
%
% first in Fig. 1 single polynomial approximation of higher order
close all
n = 10;
x = -5:0.1:5;
y = 1./(x.^2 + 1);
p = polyfit(x,y,n);
figure(1)
fplot('1/(x^2 + 1)', [-7 7], 'rs-');
hold on
xx = -5:0.01:5;
plot(xx,polyval(p,xx),'g*-')
legend('f(x)=1/(x^2 + 1)', 'Single polynomial approx. of f(x)');
title('\bf{Single polynomial approximation of a function }');

% now the piecewise polynomial interpolation using spline or spline + ppval
figure(2)
yy = spline(x,y,xx) ;
fplot('1/(x^2 + 1)', [-7 7], 'rs-')
hold on
plot(xx,yy,'g*-')
legend('f(x)=1/(x^2 + 1)', 'Piecewise polynomial approx. of f(x)')
title('\bf{Piecewise polynomial approximation of a function }')
```

Das Ergebnis des ersten Plots mit der Approximation durch ein einzelnes Polynom ist:



Anhand dieser Abbildung ist zu erkennen, dass solange n erhöht wird, sich der Fehler im Zentrum verbessert,zum Rand hin jedoch verschlechtert.
Die zweite Abbildung, welche unten dargestellt ist, mit der stückweisen Polynominterpolation zeigt ein besseres Ergebnis.



Übung 4.3

Betrachtet wird die Funktion $y = \sin^3(x)$ im Bereich $-2\pi \leq x < 2\pi$.

a) Tasten Sie die Funktion in diesem Bereich mit der Abtastperiode $\Delta x_1 = 0.5$ ab. Bestimmen Sie das Polynom siebten Grades, das diese Abtastwerte mit dem geringsten mittleren quadratischen Fehler annähert. Plotten Sie die originale Funktion und das angenähernde Polynom in dieselbe Abbildung für eine eine hochauflösende Abtastperiode $\Delta x_1 = 0.001$.

b) Interpolieren Sie die obigen Abtastwerte der Funktion $y = \sin^3(x)$ stückweise mittels kubischer Splines für eine hochauflösende Abtastperiode $\Delta x_1 = 0.001$ und plotten Sie die originale Funktion und das Resultat der Interpolation in dieselbe Abbildung.

4.3 Taylorreihenentwicklung

Die Taylerreihe für eine analytische Funktion $f(x)$ um den Entwicklungspunkt $x_0 = a$ ist gegeben durch:

$$f(x) = \sum_{n=0}^{\infty} (x-a)^n \cdot \frac{f^{(n)}(a)}{n!} \tag{4.1}$$

Die MATLAB Syntax und die Beschreibung der Taylorreihenentwicklung ist über die MATLAB Hilfe unter dem Punkt Symbolic Math Toolbox verfügbar. Die Hauptbefehle sind:

`taylor(f)` ist die Taylorpolynom Approximation fünfter Ordnung von $f(x)$ bei $x_0 = 0$.

`taylor(f,x,'Order',N,'ExpansionPoint',x_0)` liefert eine Taylorreihenentwicklung von $f(x)$ durch ein Polynom $N-1$-ter Ordnung um den Entwicklungspunkt x_0.

Bemerkung: Die unabhängige Variable `x` in der oberen Beschreibung ist symbolisch und muss daher zuerst als `syms x` deklariert werden. Mit der symbolische Variable `x` wird die symbolische Funktion `f` definiert, z.B. `f=sin(x)/x` Der Befehl `Res_symb = taylor(f)` liefert ebenfalls eine symbolische Funktion `Res_symb` zurück. Eine Konversion in eine numberisch auswertbare Funktion `Res_num` kann über den folgenden Befehl realisiert werden: `Res_num = {\sc Matlab}Function(Res_symb)`.

Beispiel
Die Taylorreihenentwicklung von $f(x) = \sin(x)$ bei $x_0 = 0$ mit der Ordnung 5 ist:

```
syms x
f = sin(x)
f_taylor = taylor(f)
```

Die MATLAB-Ausgabe für diesen Befehl ist:

```
f_taylor =

x^5/120 - x^3/6 + x
```

Die Taylorreihenentwicklung von $f(x) = \sin(x)$ bei $x_0 = 0$ mit der Ordnung 7 ist:

```
syms x
f = sin(x)
f_tr_7 = taylor(f,'Order',8)
```

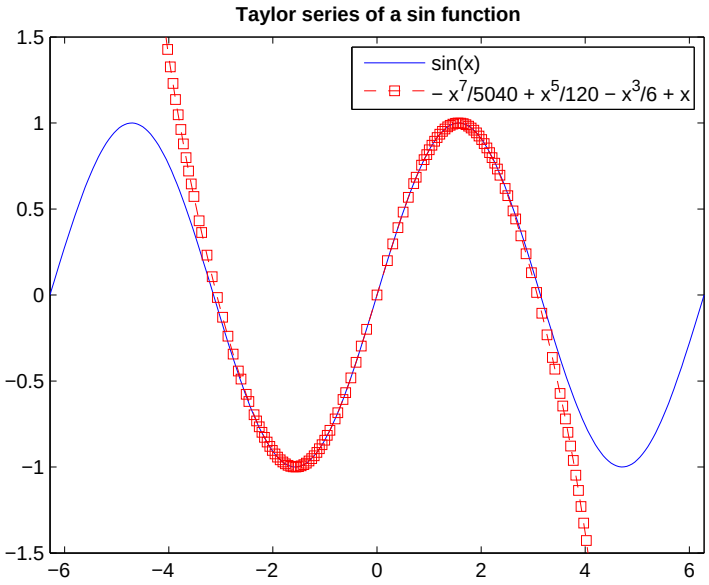
Die MATLAB-Ausgabe für diesen Befehl ist:

```
f_tr_7 =

- x^7/5040 + x^5/120 - x^3/6 + x
```

Nun plotten wir zum Vergleich die Funktion $\sin(x)$ und die oben bestimmten Taylorreihen.

```
x = linspace(-2*pi,2*pi,300);
f_taylor_7 = {\sc Matlab}Function(f_tr_7);
plot(x,sin(x),'b-')
hold on
plot(x,f_taylor_7(x),'rs--')
grid on
axis([-2*pi 2*pi -1.5 1.5])
```

Beachten Sie anhand der unteren Abbildung, wie gut die durch die Taylorreihenentwicklung bestimmten Polynome die Funktion für kleine Werte von x $(-3 \leq x \leq 3)$ annähern.



Übung 4.4

Bestimmen Sie die Taylorreihenentwicklung der folgenden Funktionen und plotten Sie die Funktion und die entsprechende Taylorreihe in die gleiche Abbildung.

a) $f_1(x) = \exp(x)$,
Entwicklungspunkt $x_0 = 0$, Ordnung $N = 5$, Wertebereich $-4 \leq x < 4$

b) $f_2(x) = \ln(x)$
Entwicklungspunkt $x_0 = 1$, Ordnung $N = 9$, Wertebereich $10^{-5} \leq x < 3$

Taylorreihenentwicklung mit den MATLAB-Befehl `taylortool`

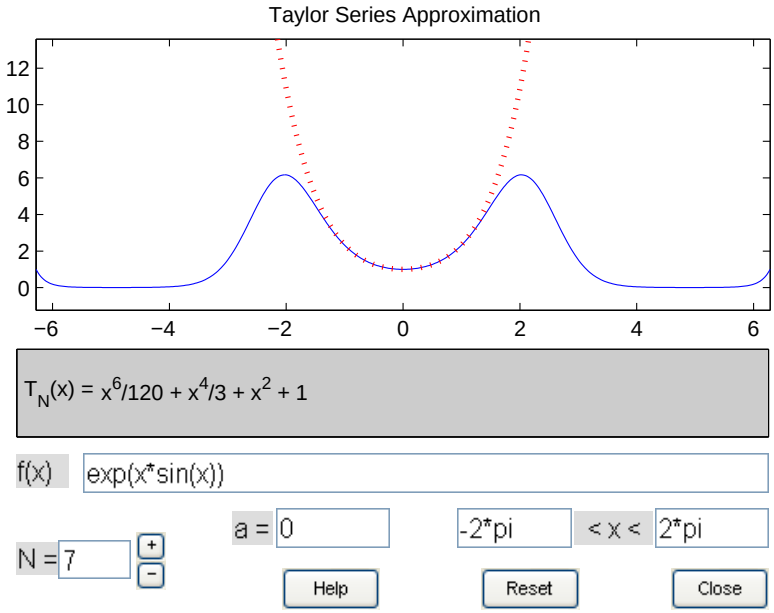
Syntax
`aylor`ool
`taylortool('f')`

Beschreibung

`taylortool` initialisiert eine Benutzeroberfläche welche eine Funktion gegen die N-te Partialsumme ihrer Taylorreihe um den Entwicklungspunkt $x = a$. Die Standardfunktion, Werte von N, Entwicklungspunkt und Berechnungsintervall von `taylortool` sind $f = x \cdot \cos(x)$, $N = 7$, $a = 0$, and $[-2\pi, 2\pi]$.

`taylortool('f')` initialisiert die GUI für den gegebenen Ausdruck f.

Beispiel
`taylortool('exp(x*sin(x))')`, Dieser Befehl öffnent die unten gezeigte GUI:



Übung 4.5

Verwenden Sie den Befehl `taylortool` und bestimmen Sie die Taylorreihenentwicklung bei für die folgenden Funktionen:

a) $f_3(x) = \tan(x)$
Entwicklungspunkt $x_0 = a = \pi$, Ordnung $N = 6$, Wertebereich $0 < x < 2\pi$

b) $f_4(x) = \dfrac{\sin(x)}{x}$
Entwicklungspunkt $x_0 = a = 0$, Ordnung $N = 10$, Wertebereich $-3\pi < x < 3\pi$

Chapter 5

Vektoren und Matrizen

Dieses Kapitel behandelt die grundlegenden Operationen auf Vektoren und Matrizen in der MATLAB-Umgebung.

5.1 Erstellung von Matrizen

Die MATLAB-Umgebung verwendet die Bezeichnung *Matrix*, um darauf hinzuweisen, dass eine Variable komplexe oder reelle Zahlen in einen zweidimensionalen Raster aufweist. Ein Array ist in der Regel ein Vektor, eine Matrix oder mehr-dimensionales Raster von Zahlen. Alle Arrays in MATLAB sind rechteckig, was bedeutet, dass die Teilvektoren in jeder Dimension die gleiche Länge aufweisen.

MATLAB hat eine Reihe von Funktion, um verschiedene Arten von Matrizen zu erstellen. Die unten gezeigten Funktion erstellen Matrizen, die am häufigsten benutzt werden.

Funktion	**Beschreibung**
`ones`	Erstellt eine Matrix oder Array gefüllt mit Einsen.
`zeros`	Erstellt eine Matrix oder Array gefüllt mit Nullen.
`eye`	Erstellt eine Matrix mit Einsen auf der Diagonalen und Nullen sonst.
`diag`	Erstellt die Diagnonalmatrix von einem Vektor.
`magic`	Erstellt eine Quadratmatrix mit Reihen, Spalten und Diagonalen, welche aufaddiert die selbe Zahl ergeben.
`rand`	Erstellt eine Matrix oder Array gefüllt mit gleichverteilten Zufallszahlen.
`randn`	Erstellt eine Matrix oder Array gefüllt mit normalverteilten Zufallszahlen und Arrays.

Für weitere Information lesen Sie `help matfun`.

Beispiel
Hier sind einige Beispiele, wie man die Funktionen benutzen kann.
Erstellen einer Zufallsmatrix. Die Funktion `rand` erstellt eine Matrix oder Array mit Elementen, die zwischen Null und Eins gleichverteilt sind. Dieses Beispiel multipliziert jedes Element mit 20:

```
A = rand(5)*20

A =

   16.2945    1.9508    3.1523    2.8377   13.1148
   18.1158    5.5700   19.4119    8.4352    0.7142
    2.5397   10.9376   19.1433   18.3147   16.9826
   18.2675   19.1501    9.7075   15.8441   18.6799
   12.6472   19.2978   16.0056   19.1898   13.5747
```

Erstellen einer Diagonalmatrix. Verwenden Sie `diag`, um von einem Vektor eine Diagonalmatrix zu erstellen. Man kann den Vektor entlang der Hauptdiagonalen der Matrix platzieren oder wie hier gezeigt, auf einer darüber oder darunterliegenden Diagonalen. Die Eingabe einer -1 platziert den Vektor eine Reihe unter der Hauptdiagonalen:

```
>> A = [1 2 3 4 1]

A =

     1     2     3     4     1

>> diag(A)

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     1

>> diag(A,-1)

ans =

     0     0     0     0     0     0
     1     0     0     0     0     0
     0     2     0     0     0     0
     0     0     3     0     0     0
     0     0     0     4     0     0
     0     0     0     0     1     0
```

Erstellen einer "Magic Square" Matrix. Ein "Magic Square" ist eine Matrix in

welcher die Summe der Elemente jeder Reihe, Spalte oder Hauptdiagonalen die Gleiche ist. Um eine 5×5 "Magic Square" Matrix zu erstellen, verwendet man die `magic` Funktion wie folgt.

```
>> A = magic(5)

A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Beachten Sie, dass die Aufaddierung der Elemente jeder Reihe, Spalte und Hauptdiagonalen den Wert 65 liefert.

Verknüpfung von Matrizen
Das Verknüpfen von Matrizen ist ein Prozess, welches eine oder mehrere Matrizen zu einer neuen Matrix zusammenfügt. Der häufig verwendete Klammer-Operator `[]`, dient nicht nur zur Erstellung von Matrizen, sondern auch als Verknüpfungsoperator in MATLAB. Der Ausdruck `C = [A B]` verknüpft die Matrizen `A` und `B` horizontal. der Ausdruck `C = [A;B]` verknüpft sie vertikal.

```
>> A = ones(2,5) * 6; % 2-by-5 matrix of 6's
>> B = rand(3,5);
>> C = [A; B]         % vertically concatenate A and B

C =

    6.0000    6.0000    6.0000    6.0000    6.0000
    6.0000    6.0000    6.0000    6.0000    6.0000
    0.7577    0.6555    0.0318    0.0971    0.3171
    0.7431    0.1712    0.2769    0.8235    0.9502
    0.3922    0.7060    0.0462    0.6948    0.0344
```

Zugriff auf einzelne Elemente
In den vorherigen Kapiteln haben wir schon behandelt, wie man auf einzelne Elemente einer Matrix zugreift. Die Syntax lautet: `A(row,column)`, wobei `A` eine variable Matrix ist.

Lineare Indizierung
Man kann sich auf die Elemente einer Matrix einfach mit dem Index `A(k)` beziehen. MATLAB speichert Matrizen und Arrays nicht in der Art und Weise, wie sie in dem

Befehlsfenster von MATLAB angezeigt werden, sondern als eine einzige Spalte von Elementen. Diese einzelne Spalte setzt sich, eine nach der anderen, aus allen anderen Spalten der Matrix zusammen.
Matrix `A`

```
>> A = [2 6 9; 4 2 8; 3 5 1]

A =

     2     6     9
     4     2     8
     3     5     1
```

ist also im Speicher als Sequenz

```
2, 4, 3, 6, 2, 5, 9, 8, 1
```

gespeichert. Das Element in Reihe 3, Spalte 2 von Matrix `A` (Wert=5) kann auch als sechstes Element in der aktuellen Speichersequenz wiedergefunden werden. Um auf dieses Element zuzugreifen, hat man die Wahl zwischen der Standardsyntax `A(3,2)` oder `A(6)`, die sogenannte *Lineare Indizierung*.

Zugriff auf mehrere Elemente
Für die unten dargestellte 4×4 Matrix **A** ist es möglich die Summe der Elemente in der vierten Spalte der Matrix **A** durch Eingabe von

```
>> A = magic(4);
>> A(1,4) + A(2,4) + A(3,4) + A(4,4)

ans =

    34
```

zu berechnen. Die Größe des Ausdruckes kann durch die Verwendung des Doppelpunkt Operators reduziert werden. Indizierte Ausdrücke, welche Doppelpunkte beinhalten beziehen sich auf Teile einer Matrix. Der Ausdruck

`A(1:m,n)` bezieht sich auf die Elemente in den Reihen 1 bis `m` der Spalte `n` der Matrix **A**. Mit Hilfe dieser Notation kann man die Summe der vierten Spalte in einer kürzeren Darstellung berechnen.

```
sum(A(1:4, 4))
```

Nicht aufeinanderfolgende Elemente
Um sich auf nicht aufeinanderfolgende Elemente in einer Matrix zu beziehen, verwendet

man den Doppelpunkt Operator mit einer Schrittweite. Der Ausdruck `m:3:n` bedeutet, dass eine Zuweisung zu jedem dritten Element der Matrix getätigt wird. Beachten Sie, dass dieses Beispiel für die lineare Indizierung gilt:

```
>> B = A;
>> B(1:3:16) = -10

B =

   -10     2     3   -10
     5    11   -10     8
     9   -10     6    12
   -10    14    15   -10
```

MATLAB unterstützt eine Art der Array Indizierung, bei der ein Array als Index eines anderen Arrays dient. Hierbei kann man sich entweder auf die Werte selbst oder die Position der Elemente in dem indizierende Array beziehen.
Hier ist ein Beispiel für die wertebasierende Indizierung, wobei das Array **B** die Werte 1, 3, 6, 7 und 10 eines Arrays **A** indiziert. In diesem Fall werden die *numerischen Werte* von Array **B** den Elementen von **A** zugeordnet:

```
>> A = 5:5:50

A =

     5    10    15    20    25    30    35    40    45    50

>> B = [1 3 6 7 10];
>> A(B)

ans =

     5    15    30    35    50
```

Die end Anweisung
MATLAB bietet die Anweisung "end" an, um das letzte Element der jeweiligen Dimension eines Arrays zu bestimmen. Dieser Ausdruck ist hilfreich, wenn man die genaue Anzahl von Reihen und Spalten einer Matrix nicht kennt. Man kann den Ausdruck in dem vorherigen Beispiel mit

```
>> B(1:3:end) = -10
```

ersetzen.

Angabe aller Elemente einer Zeile oder Spalte
Der Doppelpunkt selbst bezieht sich auf alle Elemente in einer Reihe oder Spalte einer Matrix. Durch Verwendung der folgenden Syntax kann man die Summe aller Elemente in der zweiten Spalte der 4×4 Magic-Square-Matrix **A** berechnen:

```
>> A = magic(4);
>> sum(A(:,2))

ans =

    34
```

Durch Verwendung des Doppelpunktes in Verbindung mit linearer Indizierung kann man sich auf alle Elemente der gesamten Matrix beziehen. Der Befehl `A(:)` zeigt alle Elemente der Matrix **A** angeordnet in einer Spalte an.

Verwendung von Logiken bei der Array Indizierung
Eine logische Arrayindizierung bestimmt die Elemente eines Arrays **A** basierend auf ihrer Position im indizierenden Array **B**, nicht dessen Werte. Bei dieser Art der Maskierungsoperation wird jedes wahre Element des indizierenden Arrays als Positionsindex für das Array, auf welches zugegriffen wird, behandelt.
In dem folgenden Beispiel ist **B** eine Matrix mit logischen Einsen und Nullen. Die Positionen dieser Elementen in **B** bestimmen welche Elemente von **A** durch den Ausdruck `A(B)` zugeordnet werden:

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

A =

     1     2     3
     4     5     6
     7     8     9

>> B = logical([0 1 0; 1 0 1; 0 0 1]);
>> A(B)

ans =

     4
     2
     6
     9
```

Die `find`-Funktion kann hilfreich in Verbindung mit logischen Arrays sein, weil sie die lineraren Indizes der Elemente in **B** liefert, die nicht Null sind. Es hilft des Ausdruck `A(B)` zu interpretieren:

```
>> find(B)

ans =

     2
     4
     8
     9
```

5.2 Inverse und Determinanten

Wenn **A** und **B** quadratische Matrizen der Ordnung $n \times n$ sind, welche die Gleichung

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$$

erfüllen, dann wird **B** die Inverse von **A** genannt. Wir schreiben $\mathbf{B} = \mathbf{A}^{-1}$. Da die Definition symmetrisch ist, ist **A** die Inverse von **B**, also $\mathbf{A} = \mathbf{B}^{-1}$.
Untersuchen wir nun einen allgemeine 2×2 Matrix, bestimmen die inverse Matrix und betrachten die Bedingungen für die Existenz einer inversen Matrix.

$$\mathbf{Ax} = \mathbf{d} \ ,$$

wobei

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

deshalb

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= d_1 \\ a_{21}x_1 + a_{22}x_2 &= d_2 \ . \end{aligned}$$

Dies sind lineare Gleichung mit den Unbekannten x_1 und x_2.
x_1 und x_2 sind, vorausgesetzt $a_{11}a_{22} - a_{21}a_{12} \neq 0$, gegeben durch

$$x_1 = \frac{a_{22}d_1 - a_{12}d_2}{a_{11}a_{22} - a_{21}a_{12}}, \qquad x_2 = \frac{-a_{21}d_1 + a_{11}d_2}{a_{11}a_{22} - a_{21}a_{12}}$$

oder

$$\mathbf{x} = \frac{1}{a_{11}a_{22} - a_{21}a_{12}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \ .$$

Die Zahl $a_{11}a_{22} - a_{21}a_{12}$ ist bekannt als die Determinante der Matrix $\mathbf{A}$, $\det(\mathbf{A})$.

$$\mathbf{Ax} = \mathbf{d} \Leftrightarrow \mathbf{x} = \mathbf{A}^{-1}\mathbf{d} \qquad \Rightarrow \qquad \mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$$

Der MATLAB-Befehl zur Berechnung der Determinaten einer quadritschen Matrix $\mathbf{A}$ ist `det(A)`. Die Inverse von $\mathbf{A}$ kann, wenn sie existiert, durch den Befehl `inv(A)` bestimmt werden, d.h. wenn $\mathbf{A}$ nicht singulär ist.

Wenn $\mathbf{A}$ quadratisch und nicht singulär ist, dann ist ohne Rundungsfehler `X = inv(A)*d` theoretisch dasselbe wie `X = A\d` und `Y = d*inv(A)` ist theoretisch dasselbe wie `Y = d/A`. Allerdings sind für die Berechnung die backslash und slash Operatoren vorzuziehen, da sie weniger Rechenzeit und weniger Speicher benötigen und besserer Fehlererkennungs-Eigenschaften aufweisen.

Übung 5.1

a) Prüfen Sie *händisch* ob $\mathbf{A}$ für

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ -1 & 4 \end{bmatrix},$$

singulär ist oder nicht. Bestimmen Sie, wenn möglich, *händisch* die Inverse.

Die folgenden Punkte sind mit Hilfe von MATLAB zu lösen.

b) Gegeben sind die Matrizen:

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ -1 & 4 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 1 & -1 \end{bmatrix}.$$

Bestimmen Sie $\mathbf{A}^{-1}$, $\mathbf{B}^{-1}$, $(\mathbf{AB})^{-1}$, und $\mathbf{B}^{-1}\mathbf{A}^{-1}$.

c) Bestimmen Sie $\mathbf{A}^{-1}$, für

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 0 \\ 2 & -1 & 5 \\ -1 & -1 & 2 \end{bmatrix}.$$

Bestimmen Sie die Determinaten von $\mathbf{A}^{\mathrm{T}}$, $\mathbf{A}$ und $\mathbf{A}^{-1}$.

d) Gegeben sind

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 1 & 1 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ -1 & -1 \end{bmatrix}, \qquad \mathbf{C} = \begin{bmatrix} 2 & 1 \\ -1 & 1 \\ 0 & 1 \end{bmatrix},$$

verifizieren Sie das Distributivgesetz $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$ für die drei Matrizen.

e) Gegeben sind die Matrizen:

$$\mathbf{A} = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} -2 & -1 \\ 4 & 2 \end{bmatrix} .$$

Bestimmen Sie die Inversen von $(\mathbf{BA})$ und $(\mathbf{AB})$ wenn sie existieren.

f) Gegeben ist die Matrix:

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

Bestimmen Sie eine Matrix $\mathbf{C}$ so, dass $\mathbf{A}+\mathbf{C}$ die Einheitsmatrix $\mathbf{I}_3$ ergibt, d.h. 3×3 Einheitsmatrix.
Bestimmen Sie $\mathbf{AC}$, $\mathbf{CA}$ und $\mathbf{A}^2 + \mathbf{C}^2$.
Überprüfen Sie ob $\mathbf{AC} = \mathbf{CA}$ ist.

g) Für eine allgemeine $n \times n$ Matrix kann gezeigt werden, dass $\mathbf{A} + \mathbf{A}^{\mathrm{T}}$ eine symmetrische Matrix und $\mathbf{A} - \mathbf{A}^{\mathrm{T}}$ eine schiefsymmetrische Matrix ergibt. Drücken Sie die Matrix

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

als eine Summe von einer symmetrischen und einer schiefsymmetrischen Matrix aus.

Pseudoinverse
Rechteckige Matrizen sind Matrizen mit n Zeilen und $k \neq n$ Spalten. Sie besitzen keine Inverse oder Determinante. Mindestens eine der Gleichungen $\mathbf{A} \cdot \mathbf{X} = \mathbf{I}_n$ und $\mathbf{X} \cdot \mathbf{A} = \mathbf{I}_k$ weist keine Lösung auf. Einen teilweisen Ersatz für die Inverse ist durch die *Moore-Penrose Pseudoinverse* gegeben, welche durch die `pinv` Funktion berechnet wird.

Die MATLAB Syntax ist:

```
B = pinv(A)
```

Die Moore-Penrose Pseudoinverse ist eine Matrix $\mathbf{B}$ mit denselben Dimensionen wie $\mathbf{A}^{\mathrm{T}}$, welche die folgenden vier Bedingungen erfüllt:

$$\begin{gathered} \mathbf{A} \cdot \mathbf{B} \cdot \mathbf{A} = \mathbf{A} \\ \mathbf{B} \cdot \mathbf{A} \cdot \mathbf{B} = \mathbf{B} \\ \mathbf{A} \cdot \mathbf{B} \text{ ist hermitesch} \\ \mathbf{B} \cdot \mathbf{A} \text{ ist hermitesch} \end{gathered}$$

Sofern $\left(\mathbf{A}^{\mathrm{H}} \cdot \mathbf{A}\right)^{-1}$ existiert wird die Moore-Penrose Pseudoinverse gebildet durch $\mathbf{B} = \mathbf{A}^{\dagger} = \left(\mathbf{A}^{\mathrm{H}} \cdot \mathbf{A}\right)^{-1} \cdot \mathbf{A}^{\mathrm{H}}$ und es gilt:

$$\mathbf{B} \cdot \mathbf{A} = \mathbf{A}^{\dagger} \cdot \mathbf{A} = \left(\mathbf{A}^{\mathrm{H}} \cdot \mathbf{A}\right)^{-1} \cdot \mathbf{A}^{\mathrm{H}} \cdot \mathbf{A} = \mathbf{I}_k$$

Beispiel

```
>> rand('state', 0);
>> A = rand(3,2)

A =

    0.9501    0.4860
    0.2311    0.8913
    0.6068    0.7621

>> B = pinv(A)

B =

    1.1459   -0.7411    0.1360
   -0.5022    1.0729    0.3776

>> A*B

ans =

    0.8447   -0.1827    0.3127
   -0.1827    0.7850    0.3680
    0.3127    0.3680    0.3703

>> (A*B)'

ans =

    0.8447   -0.1827    0.3127
   -0.1827    0.7850    0.3680
    0.3127    0.3680    0.3703

>> B*A

ans =

    1.0000   -0.0000
    0.0000    1.0000

>> (B*A)'

ans =

    1.0000    0.0000
   -0.0000    1.0000
```

$\mathbf{B} \cdot \mathbf{A}$ ist die 2×2 Einheitsmatrix, aber $\mathbf{A} \cdot \mathbf{B}$ ist nicht die 3×3 Einheitsmatrix. Jedoch fungiert $\mathbf{A} \cdot \mathbf{B}$ wie eine Einheitsmatrix auf einen Teil des Vektorraumes in dem Sinne, dass $\mathbf{A} \cdot \mathbf{B}$ symmetrisch ist, wobei $\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{A}$ gleich $\mathbf{A}$ und $\mathbf{B} \cdot \mathbf{A} \cdot \mathbf{B}$ gleich $\mathbf{B}$ ist.

Übung 5.2

Gegeben ist $\mathbf{A}$ mit `A = 10*rand(3,2)`.

- Bestimmen Sie die *Moore-Penrose Pseudoinverse* $\mathbf{B}$ und überprüfen Sie: $\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{A} = \mathbf{A}$, $\mathbf{B} \cdot \mathbf{A} \cdot \mathbf{B} = \mathbf{B}$, $\mathbf{A} \cdot \mathbf{B}$ ist hermitesch, $\mathbf{B} \cdot \mathbf{A}$ ist hermitesch.

5.3 Vektornormen und das Skalarprodukt in Matlab

Die p-Norm eines Vektors $\mathbf{x}$

$$\|\mathbf{x}\|_p = \left(\sum |x_i|^p \right)^{1/p}$$

wird durch `norm(x,p)` berechnet. Diese ist definiert für jeden Wert $p > 1$. Die üblichsten Werte für p sind $1, 2$ und ∞. Der Standardwert ist $p = 2$, welches der *Euklidischen Länge* entspricht:

Beispiel

```
>> v = [2 0 -1];
>> [norm(v,1) norm(v) norm(v,inf)]

ans =

    3.0000    2.2361    2.0000
```

Übung 5.3

- Berechnen Sie die 1., 2., 3., 10. und die unendliche Norm der folgenden beiden Vektoren: `x = [2 2 1]` und `y = [-1 1 -10]`.

Das Skalarprodukt von zwei Vektoren
Angenommen zwei Vektoren $\mathbf{a}$ und $\mathbf{b}$ sind in ihrer dreidimensionalen Komponentenform in Zeilenvektordarstellung gegeben durch:

$$\mathbf{a} = (a_1, a_2, a_3) \quad \text{and} \quad \mathbf{b} = (b_1, b_2, b_3)$$

Das Skalarprodukt ist definiert durch:

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a} \cdot \mathbf{b}^{\mathrm{T}} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

Hinweis: Sind die Vektoren als Spaltenvektoren gegeben, dann gilt: $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^{\mathrm{T}} \cdot \mathbf{b}$.
Das MATLAB Äquivalent hierzu ist die elementweise Multiplikation von zwei Vektoren der selben Größe und der Aufsummierung der Ergebnisse.

Übung 5.4

Gegeben ist ein Vektor in der $x - z$ Ebene $\mathbf{a} = (-1, 0, 1)$ und ein Vektor $\mathbf{b} = (2, 3, 2)$.

- Bestimmen Sie das Skalarprodukt $\langle \mathbf{a}, \mathbf{b} \rangle$.
- Prüfen Sie die Identität: $\langle (\mathbf{a} - \mathbf{b}), (\mathbf{a} + \mathbf{b}) \rangle \; = \; |\mathbf{a}|^2 - |\mathbf{b}|^2$.

5.4 Matlab Funktionen, die mit den Diagonalen einer Matrix arbeiten

Funktion	**Beschreibung**
`trace`	Berechnet die Summe der Elemente auf der Hauptdiagonalen.
`diag`	Gibt die Diagonalen einer Matrix aus.
`tril`	Gibt den unteren dreieckigen Teil einer Matrix aus.
`triu`	Gibt den oberen dreieckigen Teil einer Matrix aus.

Übung 5.5

Der MATLAB-Befehl `pascal(3)` erstellt eine *symmetrische* 3×3 Matrix.
Erstellen Sie eine symmetrische 4×4 Matrix mit der Funktion `pascal` und bestimmen Sie dann:

- den unteren dreieckigen Teil der Matrix.
- den oberen dreieckigen Teil der Matrix.
- einen Vektor mit den Elementen auf der Hauptdiagonalen der Matrix.
- die Summe der Elemente auf der Hauptdiagonalen.

Chapter 6

Lineare Gleichungssysteme

6.1 Lineare Systeme, Koeffizientenmatrix, erweiterte Matrix

Ein lineares Gleichungssystem mit n Unbekannten $x_1, x_2, \cdots, x_n$ ist eine Reihe von m Gleichungen der Form:

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + \ldots + a_{2n}x_n &= b_2 \\ \vdots \qquad & \quad \vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n &= b_m \ . \end{aligned} \tag{6.1}$$

Ein System mit zwei Gleichungen und drei Unbekannten ist somit:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \ . \end{aligned}$$

Die Zahlen a_{jk} sind bekannt und werden als die Koeffizienten eines Systems bezeichnet. Sind alle b_i gleich Null, dann wird (6.1) **homogenes System** genannt. Ist mindestens ein b_i ungleich Null, dann wird (6.1) **nicht-homogenes** System genannt.
Eine Lösung von (6.1) ist eine Reihe von Zahlen $x_1, \ldots, x_n$, welche alle m Gleichungen erfüllen. Ein Lösungsvektor von (6.1) ist ein Vektor $\mathbf{x}$, dessen Komponenten eine Lösung von (6.1) darstellen.

Matrixform eines linearen Systems (6.1). Die m Gleichungen von (6.1) können als eine einzelne Vektorgleichung geschrieben werden:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \ , \tag{6.2}$$

wobei die Koeffizientenmatrix $\mathbf{A} = [a_{jk}]$ die $m \times n$ Matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \text{ ist,} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \qquad \text{und} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_m \end{bmatrix}$$

sind Spaltenvektoren. Es wird angenommen, dass $\mathbf{A}$ keine Nullmatrix ist, was bedeutet, dass nicht alle Koeffizienten a_{jk} Nullen sind. Die erweiterte Matrix $\tilde{\mathbf{A}}$ von (6.1) ist

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_n \end{bmatrix}$$

Das Lösen eines solchen Systems durch Determinanten (Cramersche Regel) ist nicht zweckmäßig. Das Gaußsche Eliminationsverfahren wird hingegen oft benutzt und im Folgenden kurz behandelt.

6.2 Lösungen von linearen Systemem: Existenz und Eindeutigkeit

Die notwendigen und hinreichenden Bedingungen für die Existenz von Lösungen und für die Eindeutigkeit, sind gegeben durch den Rang der Koeffizientenmatrix $\mathbf{A}$ und $\tilde{\mathbf{A}}$.

(a) Existenz
Ein lineares System mit m Gleichungen und n Unbekannten (6.1) hat nur dann Lösungen, wenn die Koeffizientenmatrix $\mathbf{A}$ und die erweiterte Matrix $\tilde{\mathbf{A}}$ den selben Rang haben.
(b) Eindeutigkeit
Das System (6.1) hat nur genau eine Lösung, wenn der gemeinsame Rang r von $\mathbf{A}$ und $\tilde{\mathbf{A}}$ gleich n ist.
(c) Unendlich viele Lösungen
Wenn der Rang r kleiner ist als n, dann hat das System unendlich viele Lösungen. All diese erhält man durch Bestimmung von r geeigneten Unbekannten im Sinne der restlichen $n - r$ Unbekannten, denen beliebige Werte zugeordnet werden können.
(d) Gauss Elimination
Wenn Lösungen existieren können diese durch das Gaußsche Eliminationsverfahren bestimmt werden.

Ein homogenes lineares System hat immer **die triviale Lösung** $x_1 = 0, \ldots, x_n = 0$. Nichttriviale Lösungen (unendlich viele) existieren nur dann, wenn gilt: $\text{rang}(\mathbf{A}) < n$.

Übung 6.1

Bestimmen Sie die Existenz von Lösungen für die folgenden Systeme von linearen Gleichungen und bestimmen Sie gegebenenfalls auch die Anzahl von Lösungen.

a)

$$\begin{aligned} 2x_1+\ 2x_2+\ 4x_3 &= -2 \\ 4x_1+\ 5x_2+13x_3 &= -7 \\ 10x_1+14x_2+43x_3 &= -25 \end{aligned}$$

b)

$$\begin{aligned} 3x_1-4x_2 &= -2 \\ -x_1+5x_2 &= \ \ 4 \\ x_1+6x_2 &= \ \ 3 \end{aligned}$$

c)

$$\begin{aligned} 2x_1+\ x_2 &= -1 \\ -x_1+\ x_2 &= \ \ 2 \\ 3x_1+3x_2 &= \ \ 0 \end{aligned}$$

d)

$$\begin{aligned} x_1+\ x_2+x_3 &= 1 \\ -x_1-2x_2+x_3 &= 2 \\ x_1-\ x_2+5x_3 &= 0 \end{aligned}$$

e)

$$\begin{aligned} x_1+\ x_2+\ x_3+3x_4 &= 0 \\ 2x_2+\qquad 2x_4 &= 5 \\ -x_1-\ x_2-2x_3-2x_4 &= 4 \\ 2x_1+4x_2+2x_3+8x_4 &= 5 \end{aligned}$$

6.3 Nummerische Methoden in der linearen Algebra

Gaußsche Eliminationsverfahren

Diese Standardmethode ist ein systematischer Eliminierungsprozess, um lineare Gleichungssysteme zu lösen, welcher (6.1) zu einer "*Dreiecksform*" reduziert, wodurch das System einfach durch "*Rückwärts-Substitution*" gelöst werden kann. Ein dreieckiges System ist zum Beispiel

$$\begin{aligned} 3x_1 + 5x_2 + 2x_3 &= 8 \\ 8x_2 + 2x_3 &= -7 \\ 6x_3 &= 3 \end{aligned} .$$

LU-Zerlegung

Nun wird davon ausgegangen, dass $\mathbf{A}$ eine $n \times n$ Quadratmatrix ist. Im Folgenden werden kurz die Methoden, welche Modifikationen des Gaußschen Eliminationsverfahrens sind, behandelt. Sie sind benannt nach Doolittle, Crout und Cholesky und verwenden die Idee der LU-Zerlegung, welche als Erstes erläutert wird.
Eine *LU-Zerlegung* einer gegebenen Quadratmatrix $\mathbf{A}$ ist von der Form

$$\mathbf{A} = \mathbf{LU} , \tag{6.3}$$

wobei $\mathbf{L}$ die untere und $\mathbf{U}$ die obere Dreiecksmatrix ist.
Zum Beispiel,

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 8 & 5 \end{bmatrix} = \mathbf{LU} = \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & -7 \end{bmatrix} .$$

Es kann bewiesen werden, das für jede nicht-singuläre Matrix (interessierte Leser können sich auf Sec.6 von advanced engineering mathematics von Kreyszig,E beziehen) die Reihen so neu angeordnet werden können, dass die resultierende Matrix $\mathbf{A}$ eine LU-Zerlegung besitzt, in welcher $\mathbf{U}$ wie sich gezeigt hat, ein Dreieckssystem nach Beendigung der Gaußschen Elimination und $\mathbf{L}$ die Matrix der Multiplikatoren m_{jk} der Gaußschen Elimination mit der Diagonalen $(1, \ldots, 1)$ ist.
Die Grundidee ist hier, dass $\mathbf{L}$ und $\mathbf{U}$ in (6.3) direkt ausgerechnet werden können ohne das Gleichungssystem zu lösen. Folglich ohne die Verwendung des Gaußschen Eliminationsverfahren.

Sobald wir (6.3) haben, kann $\mathbf{Ax} = \mathbf{b}$ in zwei Schritten gelöst werden. Einfach durch die Festlegung das $\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$ als

$$\mathbf{Ly} = \mathbf{b} , \tag{6.4a}$$

geschrieben werden kann,wobei

$$\mathbf{Ux} = \mathbf{y} \tag{6.4b}$$

ist und dem lösen von (6.4a) für $\mathbf{y}$ und anschließend von (6.4b) für $\mathbf{x}$. Dieses wird als *Doolittle's Methode* bezeichnet. Beide Systeme (6.4a) und (6.4b) sind Dreieckig, so dass

ihre Lösung die selbe ist wie bei der Rückwärts-Substitution der Gaußschen Elimination. Eine ähnliche Methode, die *Crout's Methode*, erhält man von (6.3), wenn $\mathbf{U}$ (anstelle von $\mathbf{L}$) benötigt wird, um die Hauptdiagonale $(1, \ldots, 1)$ zu erhalten. In jedem Fall ist die Zerlegung (6.3) die selbe.

Beispiel 6.2.1
Ein System ist durch die Matrix $\mathbf{A}$ und den Vektor $\mathbf{b}$ wie folgt gegeben:

$$\mathbf{A} = \begin{bmatrix} 3 & 5 & 2 \\ 0 & 8 & 2 \\ 6 & 2 & 8 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -7 \\ -4 \\ 2 \end{bmatrix} .$$

Das System soll mit Hilfe der *Doolittle Methode* (LU-Zerlegung mit Einsen auf der Hauptdiagonalen von $\mathbf{L}$) gelöst werden.

Lösung:

$$\mathbf{A} = \begin{bmatrix} 3 & 5 & 2 \\ 0 & 8 & 2 \\ 6 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$\Rightarrow u_{11} = 3, \qquad l_{21}u_{11} = 0 \Leftrightarrow l_{21} = 0, \qquad \ldots$$

$$\mathbf{A} = \begin{bmatrix} 3 & 5 & 2 \\ 0 & 8 & 2 \\ 6 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} 3 & 5 & 2 \\ 0 & 8 & 2 \\ 0 & 0 & 6 \end{bmatrix} .$$

Löse zuerst für $\mathbf{Ly} = \mathbf{b}$:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -7 \\ -4 \\ 2 \end{bmatrix} \qquad \Rightarrow \mathbf{y} = \begin{bmatrix} -7 \\ -4 \\ 12 \end{bmatrix} .$$

Löse nun $\mathbf{Ux} = \mathbf{y}$:

$$\begin{bmatrix} 3 & 5 & 2 \\ 0 & 8 & 2 \\ 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -7 \\ -4 \\ 12 \end{bmatrix} \qquad \Rightarrow \mathbf{x} = \begin{bmatrix} -2 \\ -1 \\ 2 \end{bmatrix} .$$

Cholesky's Methode

Für eine *symmetrische, positiv definite* Matrix $\mathbf{A}$ (wobei $\mathbf{A} = \mathbf{A}^{\mathrm{T}}, \mathbf{x}^{\mathrm{T}}\mathbf{Ax} > \mathbf{0}$ für alle $x \neq 0$) kann für $\mathbf{U}$ (manchmal auch $\mathbf{R}$ genannt) in (6.3) sogar $\mathbf{R} = \mathbf{L}^{\mathrm{T}}$ gewählt werden, wobei $u_{jk} = l_{kj}$ allerdings keine Bedingungen an die Einträge der Hauptdiagonalen stellt. Die geläufigste Methode um $\mathbf{Ax} = \mathbf{b}$, basierend auf der Zerlegung $\mathbf{A} = \mathbf{LL}^{\mathrm{T}}$ zu lösen, wird *Cholesky's Methode* genannt. Wenn $\mathbf{A}$ symmetrisch aber nicht positiv definit ist, kann diese Methode dennoch angewendet werden, führt allerdings dann zu einer komplexen Matrix $\mathbf{L}$ was unpraktikabel ist

Beispiel 6.2.2
Lösung mit *Cholesky's Methode*:

$$\begin{aligned} 4x_1 + 2x_2 + 14x_3 &= 14 \\ 2x_1 + 17x_2 - 5x_3 &= -101 \\ 14x_1 - 5x_2 + 83x_3 &= 155 \end{aligned}$$

Lösung:

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 14 \\ 2 & 17 & -5 \\ 14 & -5 & 83 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

$$\Rightarrow l_{11} = \sqrt{a_{11}} = 2 \qquad l_{21} = \frac{a_{21}}{l_{11}} = \frac{2}{2} = 1 \qquad l_{31} = \frac{a_{31}}{l_{11}} = \frac{14}{2} = 7 \qquad \ldots$$

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 14 \\ 2 & 17 & -5 \\ 14 & -5 & 83 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 4 & 0 \\ 7 & -3 & 5 \end{bmatrix} \begin{bmatrix} 2 & 1 & 7 \\ 0 & 4 & -3 \\ 0 & 0 & 5 \end{bmatrix} .$$

Zuerst wird $\mathbf{Ly} = \mathbf{b}$ gelöst:

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 4 & 0 \\ 7 & -3 & 5 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 14 \\ -101 \\ 155 \end{bmatrix} \qquad \Rightarrow \mathbf{y} = \begin{bmatrix} 7 \\ -27 \\ 5 \end{bmatrix} .$$

Nun löse $\mathbf{L}^{\mathrm{T}}\mathbf{x} = \mathbf{y}$:

$$\begin{bmatrix} 2 & 1 & 7 \\ 0 & 4 & -3 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ -27 \\ 5 \end{bmatrix} \qquad \Rightarrow \mathbf{x} = \begin{bmatrix} 3 \\ -6 \\ 1 \end{bmatrix} .$$

6.4 Matlab-Syntax und Beschreibungen für die LU- und Cholesky Zerlegung

```
LU matrix factorization

Syntax

[L,U] = lu(A)
[L,U,P] = lu(A)

Description

The lu function expresses a matrix A as the product of two essentially
triangular matrices, one of them a permutation of a lower triangular
matrix and the other an upper triangular matrix. The factorization is
often called the LU, or sometimes the LR, factorization. A can be
rectangular. For a full matrix A, lu uses the Linear Algebra Package
(LAPACK) routines.

[L,U] = lu(A) returns an upper triangular matrix in U and a permuted
lower triangular matrix in L such that A = L*U. Return value L is a
product of lower triangular and permutation matrices.

[L,U,P] = lu(A) returns an upper triangular matrix in U, a lower
triangular matrix L with a unit diagonal, and a permutation matrix P,
such that L*U = P*A. The statement lu(A,'matrix') returns identical
output values.
```

Übung 6.2

a) Lösen mit der *Doolittle Methode* händisch ohne die Verwendung von MATLAB das folgende Gleichungssystem:

$$\begin{aligned} 4x_1 + 5x_2 &= 7 \\ 12x_1 + 14x_2 &= 18 \end{aligned}$$

b) Finden Sie mit MATLAB die Matrizen $\tilde{\mathbf{L}}$ und $\tilde{\mathbf{U}}$ der (mit der Permutationsmatrix $\mathbf{P}$ permutierten) LU-Zerlegung der Systemmatrix $\tilde{\mathbf{A}} = \mathbf{P}\tilde{\mathbf{L}}\tilde{\mathbf{U}}$ des folgenden Gleichungssystems.

$$\begin{aligned} 2x_1 + 2x_2 + 4x_3 &= -2 \\ 4x_1 + 5x_2 + 13x_3 &= -7 \\ 10x_1 + 14x_2 + 43x_3 &= -25 \end{aligned}$$

Cholesky factorization

Syntax

```
R = chol(A)
L = chol(A,'lower')
[R,p] = chol(A)
[L,p] = chol(A,'lower')
```

Description

R = chol(A) produces an upper triangular matrix R from the diagonal and upper triangle of matrix A,satisfying the equation R'*R=A. The lower triangle is assumed to be the (complex conjugate) transpose of the upper triangle. Matrix A must be positive definite; otherwise, {\sc Matlab} software displays an error message.

L = chol(A,'lower') produces a lower triangular matrix L from the diagonal and lower triangle of matrix A, satisfying the equation L*L'=A. When A is sparse, this syntax of chol is typically faster. Matrix A must be positive definite; otherwise {\sc Matlab} displays an error message.

[R,p] = chol(A) for positive definite A, produces an upper triangular matrix R from the diagonal and upper triangle of matrix A, satisfying the equation R'*R=A and p is zero. If A is not positive definite, then p is a positive integer and {\sc Matlab} does not generate an error. When A is full, R is an upper triangular matrix of order q=p-1 such that R'*R=A(1:q,1:q). When A is sparse, R is an upper triangular matrix of size q-by-n so that the L-shaped region of the first q rows and first q columns of R'*R agree with those of A.

[L,p] = chol(A,'lower') for positive definite A, produces a lower triangular matrix L from the diagonal and lower triangle of matrix A, satisfying the equation L*L'=A and p is zero. If A is not positive definite, then p is a positive integer and {\sc Matlab} does not generate an error. When A is full, L is a lower triangular matrix of order q=p-1 such that L*L'=A(1:q,1:q). When A is sparse, L is a lower triangular matrix of size q-by-n so that the L-shaped region of the first q rows and first q columns of L*L' agree with those of A.

Übung 6.3

- Bestimmen Sie die Matrizen $\mathbf{L}^{\mathrm{T}}$ und $\mathbf{L}$ der Cholesky-Zerlegung der Systemmatrix $\mathbf{A}$, welche zur Lösung des folgenden linearen Systems mit der Cholesky Methode benötigt werden.
  Überprüfen Sie zuerst die Bedingungen $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$ (symmetrisch) und $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0$ (positiv definit).

$$
\begin{aligned}
4x_1 + 6x_2 + 8x_3 &= 0 \\
6x_1 + 34x_2 + 52x_3 &= -160 \\
8x_1 + 52x_2 + 129x_3 &= -452
\end{aligned}
$$

6.5 Matlab-Lösungen für lineare Gleichungssysteme

Computerbetrachtungen
Eines der bedeutendsten Problemstellungen der Datenverarbeitung ist die Lösung von linearen Gleichungssystemen.

Es ist lehrreich ein 1-mal-1 Beispiel zu betrachten. Besitzt beispielsweise die Gleichung

$$7x = 21$$

eine eindeutige Lösung? Die Antwort ist natürlich Ja. Die Gleichung hat die eindeutige Lösung x = 3. Die Lösung erhält man leicht durch Division:

$$x = 21/7 = 3 \; .$$

Die Lösung erhält man üblicherweise nicht durch Berechnung der Inversen von 7 welches $7^{-1} = 0.142857...$ ergibt und anschließender Multiplikation von 7^{-1} mit 21. Das würde mehr Arbeit bedeuten und ist, wenn 7^{-1} durch eine endliche Anzahl von Nachkommastellen repräsentiert wird, auch weniger genau. Ähnliche Überlegungen gelten auch für lineare Gleichungssysteme mit mehr als einer Unbekannten. Die Software MATLAB löst solche Gleichungen ohne die Inverse einer Matrix zu berechnen.

Obwohl es nicht die mathematische Standardschreibweise ist, verwendet MATLAB ähnlich wie im skalaren Fall, die Terminologie der Division, um die Lösung eines üblichen linearen Gleichungssystems zu beschreiben. Die beiden Divisionssymbole Slash '/' und Backslash '\' entsprechen der beiden MATLAB Funktionen `mldivide` und `mrdivide`. `mldivide` und `mrdivide` werden für die beiden Situationen verwendet, wo die unbekannte Matrix auf der linken oder rechten Seite der Koeffizientenmatrix erscheint:

`x = A\b` liefert das Ergebnis für die Matrixgleichung $\mathbf{Ax} = \mathbf{b}$

`x = A/b` liefert das Ergebnis für die Matrixgleichung $\mathbf{xA} = \mathbf{b}$

Die Koeffizientenmatrix $\mathbf{A}$ muss nicht quadratisch sein. Wenn $\mathbf{A}$ eine $m \times n$ Matrix ist, ergeben sich drei Fälle:

$m = n$ Quadratisches System: Suche eine exakte Lösung.
$m > n$ Überdeterminiertes System: Finde eine angenäherte Lösung.
$m < n$ Unterdeterminiertes System: Finde eine Basislösung mit höchstens m Komponenten ungleich Null.

Der mldivide Algorithmus

```
Description

A\B is the matrix division of A into B, which is roughly the
    same as INV(A)*B , except it is computed in a different way.
    If A is an N-by-N matrix and B is a column vector with N
    components, or a matrix with several such columns, then
    X = A\B is the solution to the equation A*X = B. A warning
    message is printed if A is badly scaled or nearly singular.
    A\EYE(SIZE(A)) produces the inverse of A.

    If A is an M-by-N matrix with M < or > N and B is a column
    vector with M components, or a matrix with several such columns,
    then X = A\B is the solution in the least squares sense to the
    under- or overdetermined system of equations A*X = B. The
    effective rank, K, of A is determined from the QR decomposition
    with pivoting. A solution X is computed which has at most K
    nonzero components per column. If K < N this will usually not
    be the same solution as PINV(A)*B.  A\EYE(SIZE(A)) produces a
    generalized inverse of A.
```

Beispiel 6.5.1

```
>> A = pascal(3)  % symmetrische 3 mal 3 Matrix

A =

     1     1     1
     1     2     3
     1     3     6

>> b = [3 1 4]' %  Spaltenvektor der dim. 3

b =

     3
     1
     4

>> r1 = rank(A)  % A ist 3 mal 3, d.h. n = 3

r1 =
```

```
     3

>> r2 = rank([A b]) % r2 ist der Rang der erweiterten Matrix

r2 =

     3

>> % rank(A)=rank([A b])= n, daher erwarten wir eine eindeutige Lösung
>> x = A\b % ist die Lösung des Gleichungssystems: Ax = b

x =

    10
   -12
     5

>> % Beweis das Ax tatsächlich gleich b
>> Ax = A*x

Ax =

     3
     1
     4

>> % A*x ist tatsächlich gleich b und x ist deshalb die eindeutige
Lösung für Ax = b.
```

Übung 6.4

Bestimmen Sie für die Gleichungssysteme aus Übung 6.1 (a-c) eine (gegebenenfalls genäherte) Lösung.

a)

$$\begin{aligned} 2x_1 + 2x_2 + 4x_3 &= -2 \\ 4x_1 + 5x_2 + 13x_3 &= -7 \\ 10x_1 + 14x_2 + 43x_3 &= -25 \end{aligned}$$

b)

$$\begin{aligned} 3x_1 - 4x_2 &= -2 \\ -x_1 + 5x_2 &= 4 \\ x_1 + 6x_2 &= 3 \end{aligned}$$

c)

$$\begin{aligned} 2x_1 + x_2 &= -1 \\ -x_1 + x_2 &= 2 \\ 3x_1 + 3x_2 &= 0 \end{aligned}$$

Chapter 7

Eigenwerte und Eigenvektoren

7.1 Eigenwerte, Eigenvektoren

Sei $\mathbf{A}$ eine $n \times n$-Matrix und betrachten wir die Vektorgleichung

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \,, \tag{7.1}$$

in welcher $\mathbf{x}$ ein unbekannter Vektor und λ ein unbekanntes Skalar ist, welche wir beide bestimmen wollen. Selbstverständlich ist $\mathbf{x} = \mathbf{0}$ eine Lösung, die jedoch von keinem praktischen Interesse ist. Ein Wert λ, für den (7.1) eine Lösung $\mathbf{x} \neq \mathbf{0}$ hat, wird *Eigenwert* oder charakteristischer Wert der Matrix $\mathbf{A}$ genannt. Die entsprechenden Lösungen $\mathbf{x} \neq \mathbf{0}$ von (7.1) zu einem Eigenwert λ werden *Eigenvektoren* oder charakteristische Vektoren von $\mathbf{A}$ genannt.
Die Menge von simultanen Gleichungen $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ können in folgender Form geschrieben werden:

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0} \,, \tag{7.2}$$

mit der $n \times n$ Einheitsmatrix $\mathbf{I}$. Dieses ist äquivalent zu:

$$(\lambda\mathbf{I} - \mathbf{A})\mathbf{x} = \mathbf{0}. \tag{7.3}$$

Die Matrixgleichung (7.3) repräsentiert einfach eine Reihe von homogenen Gleichungen und wie wir aus dem letzten Kapitel wissen existiert eine nicht-triviale Lösung wenn

$$p(\lambda) = \det(\lambda\mathbf{I} - \mathbf{A}) = 0 \,, \tag{7.4}$$

wobei $p(\lambda)$ eine Erweiterung der Determinante und ein Polynom n-ten Grades von λ ist. Es ist bekannt als das *charakteristische Polynom* von $\mathbf{A}$. Die Gleichung $p(\lambda) = 0$ wird *charakteristische Gleichung* von $\mathbf{A}$ genannt.

Beispiel 7.1.1
Bestimmen Sie die charakteristische Gleichung und die Eigenwerte von

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

Lösung: Die charakteristische Gleichung ist gegeben durch

$$p(\lambda) = \begin{vmatrix} \lambda - 1 & -3 \\ -2 & \lambda - 2 \end{vmatrix} = 0$$

$\Rightarrow (\lambda - 1)(\lambda - 2) - 6 = 0$
$\Leftrightarrow \lambda^2 - 3\lambda - 4 = 0$
$\Rightarrow \lambda_1 = -1, \quad \lambda_2 = 4$ sind die Eigenwerte der Matrix $\mathbf{A}$.
Aus der linearen Algebra ist bekannt das $\lambda_1 + \lambda_2 = \text{trace}(\mathbf{A})$, hier $-1 + 4 = 1 + 2$, und $\lambda_1 \cdot \lambda_2 = \det(\mathbf{A})$.

Zu jedem Eigenwert λ existiert ein Eigenvektor $\mathbf{x}$, welcher die Gleichung (7.1) erfüllt.

Beispiel 7.1.2
Bestimmen Sie die Eigenvektoren der Matrix $\mathbf{A}$ aus Beispiel 7.1.1.
Sei $\mathbf{x}_1 = \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix}$ der Eigenvektor für $\lambda_1 = -1$:

$$\mathbf{A}\mathbf{x}_1 = \lambda_1 \mathbf{x}_1 \, .$$

Lösung:

$$\begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} - \lambda_1 \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} = \mathbf{0}$$

$$\Leftrightarrow \begin{bmatrix} 1 - \lambda_1 & 3 \\ 2 & 2 - \lambda_1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} = \mathbf{0}$$

$$\Leftrightarrow \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} = \mathbf{0}$$

$$\Rightarrow 2x_{11} + 3x_{21} = 0$$

$$\Rightarrow x_1 1 = -\frac{3}{2} x_{21}$$

$$\Rightarrow \mathbf{x}_1 = \begin{bmatrix} -\frac{3}{2} x_{21} \\ x_{21} \end{bmatrix}$$

Jeder Wert x_{21}, der ungleich null ist, erzeugt einen gültigen Eigenvektor. Wir wählen einen geeigneten Wert für diesen Parameter, um eine Lösung zu erhalten. Die Anderen sind Vielfache davon. Wählle $x_{21} = 2$ und $x_{11} = -3$:

$$\mathbf{x}_1 = \begin{bmatrix} -3 \\ 2 \end{bmatrix} .$$

Es ist manchmal vorteilhaft die Eigenvektoren entsprechend einer bestimmten Vorgabe zu skalieren. Häufig **normiert** man die Eigenvektoren so, dass sie die einheitliche Länge ± 1 besitzen.
Sei $\mathbf{x}_{1\mathrm{n}}$ der normierte Eigenvektor von $\mathbf{x}_1$:

$$\mathbf{x}_{1\mathrm{n}} = \frac{1}{\sqrt{(-3)^2 + 2^2}}\mathbf{x}_1 = \frac{1}{\sqrt{13}} \begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

In der selben Art und Weise können wir den zweiten Eigenvektor $\mathbf{x}_2$ für $\lambda_2 = 4$ berechnen:

$$\mathbf{A}\mathbf{x}_2 = \lambda_1 \mathbf{x}_2 \quad \text{with} \quad \mathbf{x}_2 = \begin{bmatrix} x_{12} \\ x_{22} \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 1-\lambda_2 & 3 \\ 2 & 2-\lambda_2 \end{bmatrix} \begin{bmatrix} x_{12} \\ x_{22} \end{bmatrix} = \mathbf{0}$$

$$\Leftrightarrow \begin{bmatrix} 1-4 & 3 \\ 2 & 2-4 \end{bmatrix} \begin{bmatrix} x_{12} \\ x_{22} \end{bmatrix} = \mathbf{0}$$

$$\Leftrightarrow \begin{bmatrix} -3 & 3 \\ 2 & -2 \end{bmatrix} \begin{bmatrix} x_{12} \\ x_{22} \end{bmatrix} = \mathbf{0}$$

$$\Rightarrow \qquad -3x_{12} + 3x_{22} = 0 \qquad \text{and} \qquad 2x_{12} - 2x_{22} = 0$$

$$\Rightarrow \qquad x_{22} = x_{12}$$

Wähle $x_{12} = 1 \ \Rightarrow x_{22} = 1$:

$$\mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\mathbf{x}_{2\mathrm{n}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} .$$

7.2 Matlab Methoden zur Bestimmung von Eigenwerten und Eigenvektoren

Für eine $n \times n$ Matrix $\mathbf{A}$ generiert der MATLAB-Befehl `p = poly(A)` einen Reihenvektor mit $n+1$ Elementen, wobei die Elemente in absteigender Reihenfolge der Potenzen angeordnet, die Koeffizienten des charakteristischen Polynoms $p(\lambda)$ von $\mathbf{A}$ sind. Die Eigenwerte von $\mathbf{A}$ werden durch die Quadratwurzel des Polynoms mit Hilfe des Befehls `roots(p)` berechnet.

Der Befehl

```
[V, D] = eig(A)
```

generiert die normierten Eigenvektoren von $\mathbf{A}$ als Spalten der Matrix $\mathbf{V}$ und die dazugehörigen Eigenwerte als Diagonalelemente der Diagonalmatrix $\mathbf{D}$. Sind die Argumente auf der linken Seite nicht vorhanden, generiert der Befehl `eig(A)` einfach nur die Eigenwerte von $\mathbf{A}$.

Beispiel 7.2.1
Betrachte die Matrix aus Beipiel 7.1.1:

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} .$$

```
>> A = [1 3; 2 2]

A =

     1     3
     2     2

>> [V, D] = eig(A)

V =

   -0.8321   -0.7071
    0.5547   -0.7071


D =

    -1     0
     0     4
```

D ist die Diagonalmatrix, dessen Diagonalelemente die Eigenwerte der gegebenen Quadratmatrix **A** sind. **V** ist die Quadratmatrix, dessen Spaltenvektoren die normierten Eigenvektoren $\mathbf{x}_{1n}$ und $\mathbf{x}_{2n}$ sind.
Beachte, dass diese Lösungen mit den Ergebnissen aus Beispiel 7.1.1 und 7.1.2 übereinstimmen.

7.3 Lineare Unabhängigkeit von Eigenvektoren

Seien $\lambda_1, \lambda_2, \cdots, \lambda_n$ *eindeutige* Eigenwerte einer $n \times n$ Matrix, dann sind die dazugehörigen Eigenvektoren $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ linear unabhängig von einander.
Unterscheiden sich die Eigenwerte jedoch nicht, so sind eventuell einige Eigenvektoren nicht linear unabhängig von einander.

Beispiel 7.3.1
a) Gegeben ist die Matrix $\mathbf{A} = \begin{bmatrix} 1 & 1 & -2 \\ -1 & 2 & 1 \\ 0 & 1 & -1 \end{bmatrix}$.

Die Eigenwerte von **A** sind:

```
>> A = [1  1  -2; -1  2  1 ; 0  1  -1]

A =
     1     1    -2
    -1     2     1
     0     1    -1

>> d = eig(A)

d =
    2.0000
    1.0000
   -1.0000
```

Drei eindeutige Eigenwerte, was bedeutet, das wir drei linear unabhängige Eigenvektoren erwarten.

```
>> [V, D] = eig(A)


V =
    0.3015   -0.8018    0.7071
    0.9045   -0.5345    0.0000
    0.3015   -0.2673    0.7071
```

```
D =
    2.0000         0         0
         0    1.0000         0
         0         0   -1.0000
```

Die drei Spaltenvektoren der Matrix $\mathbf{V}$ sind die drei linear unabhängigen Eigenvektoren. Da diese Eigenvektoren linear unabhängig sind hat die Matrix $\mathbf{V}$ eine Inverse.

b) Gegeben ist die Matrix $\mathbf{A} = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 2 & 1 \\ -1 & 2 & 2 \end{bmatrix}$.

```
>> A = [1  2  2; 0  2  1 ; -1  2  2]

A =
     1     2     2
     0     2     1
    -1     2     2

>> [V, D] = eig(A)

V =
   0.8944                  0.8944                  0.5774
   0.4472 + 0.0000i        0.4472 - 0.0000i        0.5774
   0.0000 + 0.0000i        0.0000 - 0.0000i       -0.5774


D =
   2.0000 + 0.0000i             0                        0
        0                  2.0000 - 0.0000i             0
        0                       0                   1.0000
```

$\lambda_1 = \lambda_2 = 2$ und die dazugehörigen Eigenvektoren, welche die ersten und zweiten Spaltenvektoren der Matrix $\mathbf{V}$ sind, sind genau gleich. Aus diesem Grund besitzt die Matrix $\mathbf{A}$ keine vollständige Anzahl an linear unabhängigen Eigenvektoren.

c) Gegeben ist die Matrix $\mathbf{A} = \begin{bmatrix} 3 & -3 & 2 \\ -1 & 5 & -2 \\ -1 & 3 & 0 \end{bmatrix}$.

```
>> A = [3  -3  2; -1  5  -2 ; -1  3  0]
```

```
A =

     3    -3     2
    -1     5    -2
    -1     3     0

>> [V, D] = eig(A)

V =

    0.5774   -0.5774   -0.7518
   -0.5774   -0.5774    0.1735
   -0.5774   -0.5774    0.6361


D =

    4.0000         0         0
         0    2.0000         0
         0         0    2.0000
```

$\lambda_2 = \lambda_3 = 2$, aber die Eigenvektoren, die Spaltenvektoren der von $\mathbf{V}$, sind linear unabhängig.

Übung 7.1

Bestimmen Sie die Eigenwerte und Eigenvektoren der folgenden Matrizen und überprüfen Sie auch, ob die Eigenvektoren linear unabhängig voneinander sind.

a) $\mathbf{A} = \begin{bmatrix} 2 & 2 & 1 \\ 1 & 3 & 1 \\ 1 & 2 & 2 \end{bmatrix}$

b) $\mathbf{A} = \begin{bmatrix} -3 & -7 & -5 \\ 2 & 4 & 3 \\ 1 & 2 & 2 \end{bmatrix}$

c) $\mathbf{A} = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 0 & 1 \\ -1 & -1 & 2 \end{bmatrix}$

d) $\mathbf{A} = \begin{bmatrix} 0 & -2 & -2 \\ -1 & 1 & 2 \\ -1 & -1 & 2 \end{bmatrix}$

e) $\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

f) $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 3 & 5 \\ 0 & 0 & 7 \end{bmatrix}$

7.4 Orthonormalität, symmetrische Matrizen

Eine quadratische Matrix $\mathbf{A}$ wird symmetrisch genannt, wenn $\mathbf{A}^{\mathrm{T}} = \mathbf{A}$.

- Die Eigenwerte einer reell symmetrischen Matrix sind reell.
- Für eine reell symmetrische $n \times n$ Matrix ist es immer möglich n linear unabhängige Eigenvektoren $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ zu finden, welche orthogonal zueinander, $\mathbf{x}_i^{\mathrm{T}} \cdot \mathbf{x}_j = 0$ für $i \neq j$, sind.

Eine Matrix wird schief-symmetrisch oder anti-symmetrisch genannt, wenn $\mathbf{A}^{\mathrm{T}} = -\mathbf{A}$.

Eine Matrix wird orthogonal genannt, wenn $\mathbf{A}^{\mathrm{T}} = \mathbf{A}^{-1} \Leftrightarrow \mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{I}_n$, wobei $\mathbf{I}_n$ die $n \times n$ Einheitsmatrix ist.

Beispiel 7.4.1

Sei $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$ eine 2×2 Matrix. Es gilt:

$$\mathbf{A}^{\mathrm{T}} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} = \mathbf{A}$$

$\Rightarrow \mathbf{A}$ ist eine reell symmetrische Matrix.

```
A =
     1     1
     1     2

>> [V, D] = eig(A)

V =
   -0.8507    0.5257
    0.5257    0.8507

D =
    0.3820         0
         0    2.6180

>> norm(V(:,1))

ans =
    1.0000

>> norm(V(:,2))

ans =
    1.0000
```

Wenn die Matrix $\mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \cdots \ \mathbf{v}_n]$ n orthonormale Eigenvektoren hat, dann ist $\mathbf{V}^{\mathrm{H}}\mathbf{V} = \mathbf{I}_n$, wobei $\mathbf{I}_n$ die $n \times n$ Einheitsmatrix ist. Mit anderen Worten ist $\mathbf{v}_i^{\mathrm{H}} \cdot \mathbf{v}_j = \delta_{ij}$, wobei δ_{ij} das Kronecker Delta darstellt, welches gleich 1 für $i = j$ und ansonsten 0 ist. Hier gilt $i \in \{1, 2\}, j \in \{1, 2\}$ und im Allgemeinen gilt für eine reell symmetrische $n \times n$ Matrix: $i \in \{1, \ldots, n\}, j \in \{1, \ldots, n\}$.

```
>> VHV = V'*V

VHV =

    1.0000         0
         0    1.0000
```

In der Tat sind die Eigenvektoren, welche die Spalten der Matrix $\mathbf{V}$ sind, orthonormal.

Übung 7.2

Berechnen Sie die Eigenwerte und die entsprechenden orthogonalen Eigenvektoren der symmetrischen Matrizen.

a) $\mathbf{A} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$

b) $\mathbf{A} = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 5 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

c)
```
>> A = pascal(3)

A =

     1     1     1
     1     2     3
     1     3     6
```

7.5 Diagonalisierung einer Matrix, kanonische Form

Wenn eine $n \times n$ Matrix $\mathbf{A}$ eine vollständige Anzahl von n linear unabhängigen Eigenvektoren $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ besitzt, dann ist:

$$\mathbf{V}^{-1}\mathbf{A}\mathbf{V} = \mathbf{\Lambda}\,, \tag{7.5}$$

wobei die Matrix $\mathbf{V}$ aus den n, als Spalten $\mathbf{V} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_n \end{bmatrix}$ angeordneten Eigenvektoren besteht und $\mathbf{\Lambda} = \mathbf{D}$ die Diagonalmatrix, mit den als Diagonalelementen angeordneten Eigenwerte $\lambda_1, \lambda_2, \cdots, \lambda_n$ von $\mathbf{A}$ ist.

Um (7.5) zu beweisen beginnen wir von (7.1).

$$\mathbf{A}\mathbf{x}_1 = \lambda_1\mathbf{x}_1, \quad \mathbf{A}\mathbf{x}_2 = \lambda_2\mathbf{x}_2, \qquad \cdots, \quad \mathbf{A}\mathbf{x}_n = \lambda_n\mathbf{x}_n$$

$$\Rightarrow \qquad \mathbf{A}\begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \end{bmatrix}\mathbf{\Lambda}$$

$$\Rightarrow \qquad \mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{\Lambda}$$

$$\Rightarrow \qquad \mathbf{V}^{-1}\mathbf{A}\mathbf{V} = \mathbf{\Lambda}$$

$$\Rightarrow \qquad \mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$$

Deshalb ist (7.5) immer gültig, wenn $\mathbf{V}$ als Spalten angeordnete linear unabhängige Eigenvektoren besitzt, was bedeutet, dass die Inverse von $\mathbf{V}$ existiert.

Potenzen von quadratischen Matrizen

Für Matrizen die wie (7.5) geschrieben werden können, lassen sich die Potenzen und die Eigenwerte und Eigenvektoren dieser Potenzen leicht wie folgt berechnen:

Sei $\mathbf{A}$ eine 3×3 Matrix mit drei eindeutigen Eigenwerten $\lambda_1, \lambda_2, \lambda_3$.

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

$$\mathbf{\Lambda}^2 = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

$$\mathbf{\Lambda}^2 = \begin{bmatrix} \lambda_1^2 & 0 & 0 \\ 0 & \lambda_2^2 & 0 \\ 0 & 0 & \lambda_3^2 \end{bmatrix}$$

Nach (7.5) ist jedoch:

$$\mathbf{\Lambda}^2 = \mathbf{\Lambda} \cdot \mathbf{\Lambda} = (\mathbf{V}^{-1}\mathbf{A}\mathbf{V})(\mathbf{V}^{-1}\mathbf{A}\mathbf{V}) = \mathbf{V}^{-1}\mathbf{A}\mathbf{I}_3\mathbf{A}\mathbf{V} = \mathbf{V}^{-1}\mathbf{A}\mathbf{A}\mathbf{V} \ ,$$

wobei $\mathbf{I}_3$ die 3×3 Einheitsmatrix ist:

$$\Rightarrow \mathbf{A}^2 = \mathbf{V}\mathbf{\Lambda}^2\mathbf{V}^{-1}$$

Daraus folgt, dass die Eigenwerte von $\mathbf{A}^2$ gleich λ_1^2, λ_2^2 und λ_3^2 und die Eigenvektoren von $\mathbf{A}^2$ die selben wie die Eigenvektoren von $\mathbf{A}$ sind.

Deshalb kann $\mathbf{A}^6 = \mathbf{A} \cdot \mathbf{A} \cdot \mathbf{A} \cdot \mathbf{A} \cdot \mathbf{A} \cdot \mathbf{A}$ berechnet werden als:

$$\mathbf{A}^6 = \mathbf{V}\mathbf{\Lambda}^6\mathbf{V}^{-1} \ .$$

Übung 7.3

Gegeben sind die Matrizen $\mathbf{A}$ und $\mathbf{B}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 3 & 2 & 1 & 0 \\ 2 & 3 & 1 & 0 \\ 1 & 1 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

a) Prüfen Sie, ob beide Matrizen reell symmetrisch sind.

b) Prüfen Sie,ob die Eigenwerte von $\mathbf{A}^5$ gleich λ_i^5 für $i \in \{1, \ldots, 4\}$ sind, wobei λ_i die Eigenwerte von $\mathbf{A}$ sind.

c) Prüfen Sie ob die Eigenvektoren von $\mathbf{B}^5$ identisch mit den Eigenvektoren von $\mathbf{B}$ sind.

d) Prüfen Sie ob $\mathbf{A}^3 = \mathbf{V}\mathbf{\Lambda}^3\mathbf{V}^{-1}$, wobei $\mathbf{\Lambda}$ die Diagonalmatrix mit den Eigenwerten von $\mathbf{A}$ ist und $\mathbf{V}$ die Eigenvektormatrix von $\mathbf{A}$ ist.

e) Prüfen Sie ob $\mathbf{B}^5 = \mathbf{V}\mathbf{\Lambda}^5\mathbf{V}^{-1}$, wobei $\mathbf{\Lambda}$ die Diagonalmatrix mit den Eigenwerten von $\mathbf{B}$ ist und $\mathbf{V}$ die Eigenvektormatrix von $\mathbf{B}$ ist.

7.6 Cayley-Hamilton Theorem

Eine quadratische Matrix $\mathbf{A}$ erfüllt ihre eigene charakteristische Gleichung. Wenn also

$$\lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_1\lambda + c_0 = 0$$

die charakteristische Gleichung einer $n \times n$ Matrix $\mathbf{A}$ ist, dann ist

$$\mathbf{A}^n + c_{n-1}\mathbf{A}^{n-1} + \cdots + c_1\mathbf{A} + c_0\mathbf{I}_n = \mathbf{0}$$

wobei $\mathbf{I}_n$ die $n \times n$ Einheitsmatrix ist.

Der Beweis dieses Theorems ist nicht trivial und hier nicht enthalten. Wir werden das Theorem mit einer einfachen Aufgabe veranschaulichen.

Übung 7.4

Überprüfen Sie das Cayley-Hamilton-Theorem für die folgenden Matrizen:

a) $\mathbf{A} = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}$

b) $\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix}$

Chapter 8

Nichtlineare Gleichungen

8.1 Nullstellen von Polynomen

Polynome werden als Koeffizientenvektor dargestellt. $p(x) = x^3 - 2x - 5$ ist demnach gegeben durch:

```
p = [1 0 -2 -5]

p =

     1     0    -2    -5
```

Die Lösungen der Gleichung $p(x) = 0$ wird bestimmt durch:

```
r = roots(p)

r =

   2.0946
  -1.0473 + 1.1359i
  -1.0473 - 1.1359i
```

Ist ein Nullstellenvektor `r` gegeben, erzeugt `poly(r)` die Koeffizienten des Polynoms mit diesen Nullstellen. Mit geringen Rundungsfehlern sollte das ursprüngliche Polynom erstellt werden.

Die `poly`-Funktion berechnet auch das charakteristische Polynom einer Matrix. Die Nullstellen dieses Polynoms sind die Eigenwerte der Matrix.

Beispiel 8.1.1

```
A = magic(3)

A =

     8     1     6
     3     5     7
     4     9     2

P = poly(A)

P =

    1.0000  -15.0000  -24.0000  360.0000
```

$\Rightarrow$ $p(\lambda) = \lambda^3 - 15\lambda^2 - 24\lambda + 360$ ist das charakteristische Polynom der Matrix $\mathbf{A}$.

```
eigValues = roots(poly(A))

eigValues =

   15.0000
   -4.8990
    4.8990

% check using the command eig

eigValues2 = eig(A)

eigValues2 =

   15.0000
    4.8990
   -4.8990
```

8.2 Lösen von reelen Funktionen mit Hilfe von fzero

Die `fzero`-Funktion bestimmt eine numerische Lösung für $f(x) = 0$, wenn $f(x)$ eine reelle Funktion mit einem reellen Funktionsbereich ist. Entweder gibt man einen Startwert vor oder man gibt zwei Werte für x an, für die sich die Funktionswerte von $f(x)$ im Vorzeichen unterscheiden.

Beschreibung
`x = fzero(fun, x0)` versucht eine Nullstelle von `fun` nahe `x0` zu finden, wenn `x0` ein Skalar ist. `fun` ist ein 'function handle' einer M-Datei oder eine 'anonyme Funktion'. Der Wert x, welcher von `fzero` zurückgegeben wird, ist nahe eines Punktes an dem `fun` das Vorzeichen ändert oder `NaN`, wenn die Suche fehlschlägt. In diesem Fall wird die Suche abgebrochen wenn das Suchintervall ins Unendliche geht (`Inf, NaN`) oder ein komplexer Wert gefunden wird.

```
Examples (From {\sc Matlab} help )
      FUN can be specified using @:
         X = fzero(@sin,3)
      returns pi.

         X = fzero(@sin,3,optimset('Display','iter'))
      returns pi, uses the default tolerance and displays iteration information.

      FUN can also be an anonymous function:
         X = fzero(@(x) sin(3*x),2)
```

Beispiel 8.2.1

Gegeben ist die Funktion $f(x) = x^2 - 2$.
$f(x) = 0$ hat zwei Lösungen $x_1 = \sqrt{2}$ und $x_2 = -\sqrt{2}$.
Durch Verwendung von `fzero` können wir x_1 und x_2 berechnen:

```
x_1 = fzero(@(x) x^2 -2 , 1) % function handle and solution search near 1

x_1 =

    1.4142

x_2 = fzero(@(x) x^2 -2 , -1) % solution search near -1

x_2 =

    -1.4142
```

Wenn $f(x)$ ein Polynom ist, kann auch die Funktion `roots` verwendet werden.

```
>> f = [1 0 -2]

f =

     1     0    -2

>> x = roots(f) % this returns the solution of f(x) = 0

x =

    1.4142
   -1.4142
```

Die Funktion $f(x) = x^2 - 2$ kann als 'anonyme Funktion' geschrieben werden:

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

f =

    @(x)x.^2-2
```

Diese 'anonyme Funktion' nimmt als Übergabeparameter sowohl skalare als auch vektorielle Argumente entgegen:

```
>> f(-sqrt(2))

ans =

  4.4409e-016

>> f(sqrt(2))

ans =

  4.4409e-016

>> f([-sqrt(2) sqrt(2)])

ans =

  1.0e-015 *
```

```
    0.4441    0.4441

>> v = [-2 -1 0 1 2]

v =

    -2    -1     0     1     2

>> f(v)

ans =

     2    -1    -2    -1     2
```

Die `fzero`-Funktion kann nur ein x finden für das $f(x)$ die x-Achse kreuzt. Sollte sich das Vorzeichen von $f(x)$ auf beiden Seiten der Nullstelle nicht unterscheiden, kann die Nullstelle nicht gefunden werden.

Beispiel 8.2.2
Im Folgenden werden zwei 'anonyme Funktionen' erstellt (reguläre M-Files können auch verwendet werden):

```
>> fa = @(x) (x-2)^2

fa =

    @(x)(x-2)^2

>> fb = @(x) (x-2)^2 - 1e-12

fb =

    @(x)(x-2)^2-1e-12
```

Die Nullstellen von `fa` können selbst mit den Startwerten 1.999 oder 2.001 nicht gefunden werden. Auch die Nullstellen von `fb` können nicht gefunden werden, wenn der Startwert zu weit von der Lösung abweicht. Betrachten Sie die folgenden Ergebnisse von `fa`:

```
>> fzero(fa, 1.999)
Exiting fzero: aborting search for an interval containing a sign change
    because NaN or Inf function value encountered during search.
(Function value at -1.71534e+154 is Inf.)
Check function or try again with a different starting value.

ans = NaN

>> fzero(fb, 2.001)
Exiting fzero: aborting search for an interval containing a sign change
    because NaN or Inf function value encountered during search.
(Function value at -1.71706e+154 is Inf.)
Check function or try again with a different starting value.

ans =  NaN

>> fzero(fb, [2 3])

ans = 2.0000
```

Übung 8.1

a) Gegeben ist die Matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix} .$$

- Bestimmen Sie das charakteristische Polynom durch Verwendung das MATLAB-Befehls `poly`.
- Finden Sie die Nullstellen des charakteristischen Polynoms durch Verwendung des MATLAB-Befehls `roots`.
- Erstellen Sie eine Diagonalmatrix $\mathbf{\Lambda}$ mit den Nullstellen des charakterischen Polynoms. Der MATLAB-Befehl `diag` ergibt eine Diagonalmatrix, wenn er auf einen Vektor angewandt wird.
- Bestimmen Sie die Eigenwerte und Eigenvektoren der Matrix $\mathbf{A}$ durch Verwendung das Standard MATLAB-Befehls `[V, D] = eig(A)`.
- Vergleichen Sie die mit den zwei oberen Methoden definierten Matrizen $\mathbf{D}$ und $\mathbf{\Lambda}$.

b) Gegeben ist die Funktion:

$$f(x) = x^3 - 9x^2 + 9x - 1 .$$

- Finden Sie mindestens eine Lösung dieser Funktion durch Verwendung des MATLAB-Befehls `fzero`.
- Bestimmen Sie die Nullstellen des Polynoms durch Verwendung des MATLAB-Befehls `roots` und vergleichen Sie die Ergebnisse mit den Ergebnissen von `fzero`.

c) Die Nullstelle einer elementaren Funktion (eingebaute Funktion) lässt sich ebenfalls mit Hilfe des MATLAB-Befehls `fzero` bestimmen (z.B. Nullstelle der Sinusfunktion um 3: `fzero(@sin, 3)`).

- Bestimmen Sie die Nullstelle der Cosinus-Funktion zwischen 1 und 2. Beachten Sie, dass cos(1) und cos(2) sich im Vorzeichen unterscheiden.

8.3 Lösen von Gleichung mit Hilfe der 'Symbolic Toolbox'

Für einen symbolischen Ausdruck `s` versucht die Anweisung `solve(s)` Werte für die symbolische Variable zu finden, für welche der symbolische Ausdruck `solve(s)` null ist. Die `solve(s)`-Funktion kann nicht alle Gleichungen lösen. Sie funktioniert gut bei Polynomen, kann aber Schwierigkeiten bei trigonometrischen oder transzendenten Gleichungen haben. Wenn eine exakte symbolische Lösung gefunden wird, kann sie, falls gewünscht, in eine Fließkomma Lösung konvertiert werden.

Beispiel 8.3.1

Gegeben ist die Gleichung $f(t) = t^3 - 9 \cdot t^2 + 9 \cdot t - 1$.

Um $f(t) = 0$ mit dem Befehl `solve` zu lösen, definieren wir zuerst die Variable t als symbolische Variable.

```
>> syms t
>> solve(t^3 -9*t^2 + 9*t -1)

ans =

            1
 4 - 15^(1/2)
 15^(1/2) + 4

>> double(ans)

ans =

    1.0000
    0.1270
    7.8730

>>  h = @(t) t^3 -9*t^2 + 9*t -1  % This is function handle to proof say h(1) = 0

h =

    @(t)t^3-9*t^2+9*t-1

>> h(1)

ans = 0
```

Die Eingabewerte für `solve` müssen symbolische Ausdrücke sein. Zur Definition dieser Ausdrücke werden symbolische Variablen verwendet, die mit dem Schlüsselwort `syms` definiert wurden.

Beispiel 8.3.2

Gegeben ist die Gleichung $\ln(x) = x - 2$.
Die Lösung mit Hilfe von `solve` ist:

```
>> syms x
>> zeros = solve(log(x) == x - 2)
zeros =

 -wrightOmega(- 2 - pi*1i)
 -wrightOmega(- 2 + pi*1i)
>> double(zeros)

ans =

    3.1462
    0.1586

>> fx = @(x) log(x) - x + 2 % function handle

fx =

    @(x)log(x)-x+2

>> fx(double(zeros))

ans =

     0
     0
```

Übung 8.2

- Lösen Sie die Gleichung $\cos(x)+\sin(x) = 0$ durch Verwendung des MATLAB-Befehls `solve`.

8.4 System von nichtlinearen Gleichungen

Ein System von n nichtlinearen Gleichungen, die gleichzeitig von n unabhängige Variablen erfüllt werden sollen, wird definiert durch:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

Der MATLAB-Befehl `fsolve` bestimmt die Lösungen eines Systems nichtlinearer Gleichungen. Eine detailierte Beschreibung der MATLAB-Syntax ist mit `help fsolve` abrufbar.

`x = fsolve(fun,x0)` startet bei `x0` und versucht die in `fun` beschriebene Gleichung zu lösen.

`x = fsolve(fun,x0,options)` löst die Gleichung mit Optimierungsmöglichkeiten, die in der Options-Struktur festgelegt werden. Verwenden Sie `optimset`, um diese Optionen zu setzen.

`[x,fval] = fsolve(fun,x0)` gibt den Funktionwert der Funktion `fun` bei der Lösung `x` zurück.

`[x,fval,exitflag] = fsolve(...)` gibt den Wert "exitflag" zurück, welcher die Exit-Bedingung beschreibt.

`[x,fval,exitflag,output] = fsolve(...)` gibt eine Ausgabestruktur zurück, welche Informationen über die Optimierungen enthält.

Das Eingabeargument `fun` ist eine Handle auf eine Funktion, die als Eingangsgröße einen Vektor $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ benötigt und diesen Vektor in einen zugehörigen Funktionswertevektor $\mathbf{f}(x_1, \ldots, x_n) = \big(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)\big)^{\mathrm{T}}$ überführt und als Ausgangsgröße zurückgibt. Die Funktion `fun` kann als 'function handle' für eine M-File Funktion definiert werden:

```
x = fsolve(@myfun,x0)
```

wobei `myfun` eine MATLAB-Funktion ist, so wie

```
function F = myfun(x)
F = ...            % Compute function values at x
```

Wenn die benutzerdefinierten Werte für `x0` und `F` Matrizen sind, werden sie mit Hilfe der linearen Indizierung in Vektoren konvertiert.

`fun` kann auch ein Handle für eine anonyme Funktion sein, z.B. `fsolve(@(x)sin(x.*x),x0)`.

Das Ausgabeargument `exitflag` gleich 1 bedeutet, dass die Funktion zu einer Lösung `x`

konvergiert. Für die anderen Werte von `exitflag` verwenden sie die MATLAB-Hilfe.

Beispiel 8.4.1

Dieses Beispiel findet eine Nullstelle eines Systems mit zwei Gleichungen und zwei Unbekannten:

$$\begin{aligned} 2x_1 - \ x_2 &= \mathrm{e}^{-x_1} \\ -x_1 + 2x_2 &= \mathrm{e}^{-x_2} \end{aligned}$$

Zunächst sind die die beiden Gleichungen so zu schreiben, dass die Funktionen $f_1(x_1, x_2)$ und $f_2(x_1, x_2)$ direkt ablesbar sind

$$\begin{aligned} f_1(x_1, x_2) &= 2x_1 - x_2 \ - \mathrm{e}^{-x_1} = 0 \\ f_2(x_1, x_2) &= -x_1 + 2x_2 - \mathrm{e}^{-x_2} = 0 \end{aligned}$$

Als initialer Startwert kann z.B. $\mathbf{x}_0 = (-5, -5)^{\mathrm{T}}$ verwendet werden.

Der erste wichtige Schritt ist eine M-Datei zu schreiben, welches `F` zurückgibt.

```
function F = myfun3(x)
F = [2*x(1) - x(2) - exp(-x(1)) ;
 -x(1) + 2*x(2) - exp(-x(2))];
```

Damit MATLAB diese Funktion verwenden kann ist die M-Datei unter dem gewählten Namen `myfun3.m` abzuspeichern! Es ist zu beachten, dass der erste Eintrag der Eingangsvariable `x(1)` der Unbekannten x_1 und der zweite Eintrag der Eingangsvariable `x(2)` der Unbekannten x_2 entspricht. Analog entsprechen die Einträge der Ausgangsvariable `F` den folgenden Funktionswerten:

$$\begin{aligned} \texttt{f(1)} &\leftrightarrow f_1(x_1, x_2) \\ \texttt{f(2)} &\leftrightarrow f_2(x_1, x_2) \end{aligned}$$

Mit der Funktion wird die Lösungssroutine mit dem initialen Startwert $\mathbf{x}_0 = (-5, -5)^{\mathrm{T}}$ aufgerufen:

```
>> x0 = [-5; -5]; % initial guess to the solution
>> options = optimset('Display','off'); % option to turn off display
>> [x,fval, exitflag] = fsolve(@myfun3,x0,options)  % Call optimizer
```

Die Lösungssroutine liefert die folgenden Ergebnisse:

```
x =

    0.5671
    0.5671


fval =

  1.0e-006 *

   -0.4059
   -0.4059


exitflag =

     1
```

Um zu sehen wie viele Iterationen nötig sind, um die Lösung zu erhalten, sind die Optionen wie folgt zu setzen:

```
>> x0 = [-5; -5]; % initial guess to the solution
>> options = optimset('Display','iter'); % option to display the output
>> [x,fval, exitflag] = fsolve(@myfun3,x0,options)  % Call optimizer
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	Trust-region radius
0	3	47071.2		2.29e+004	1
1	6	12003.4	1	5.75e+003	1
2	9	3147.02	1	1.47e+003	1
3	12	854.452	1	388	1
4	15	239.527	1	107	1
5	18	67.0412	1	30.8	1
6	21	16.7042	1	9.05	1
7	24	2.42788	1	2.26	1
8	27	0.032658	0.759511	0.206	2.5
9	30	7.03149e-006	0.111927	0.00294	2.5
10	33	3.29525e-013	0.00169132	6.36e-007	2.5

```
x =

    0.5671
    0.5671


fval =

  1.0e-006 *

   -0.4059
   -0.4059


exitflag =

     1
```

Es werden 10 Iterationen benötigt, um sich der Lösung anzunähern. Wird hingegen ein Startwert `x0` nahe der Lösung gewählt, so reduziert sich die Anzahl der Iterationen zur Annäherung der Lösung:

```
>> x0 = [0 ; 0]; % initial guess to the solution
>> options = optimset('Display','iter'); % option to display the output
>> [x,fval, exitflag] = fsolve(@myfun3,x0,options)  % Call optimizer

                                        Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0          3               2                          2              1
     1          6       0.0226976       0.707107          0.171              1
     2          9    3.40349e-006      0.0937779        0.00204           1.77
     3         12    7.72066e-014     0.00117685       3.08e-007           1.77

x =

    0.5671
    0.5671


fval =

  1.0e-006 *

   -0.1965
```

```
   -0.1965


exitflag =

     1
```

Übung 8.3

a) Lösen Sie das folgende Gleichungssystem:

$$\begin{aligned} 2x_1 + \ x_2 &= \ln(x_1) \\ x_1 + 2x_2 &= \ln(x_2)\ . \end{aligned}$$

Setzen Sie `options = optimset('Display','iter')` und benutzen Sie folgende Startwerte:

- $\mathbf{x}_{0,\mathrm{a}} = (1,1)^{\mathrm{T}}$
- $\mathbf{x}_{0,\mathrm{b}} = (1,10)^{\mathrm{T}}$

b) Bestimmen Sie eine Matrix $\mathbf{X}$, welche die Gleichung $\mathbf{X} \cdot \mathbf{X} \cdot \mathbf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ erfüllt. Verwenden Sie die Startmatrix $\mathbf{X}_0 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.
Hinweis: Schreiben Sie zuerst eine M-Datei zur Darstellung der Funktionen welche wie folgt aussieht:

```
function F = myfun4(X)
F = X*X*X - [1, 2 ; 3, 4];
```

Zur Lösung des Gleichungssystems nutzen Sie den MATLAB-Befehl `fsolve`.

Chapter 9

Ableitung

9.1 Ableitung von Polynomen

In Matlab berechnet die Funktion `polyder` die Ableitung für jedes beliebige Polynom. Zum Beispiel,

$$f(x) = x^3 - 3x \; .$$

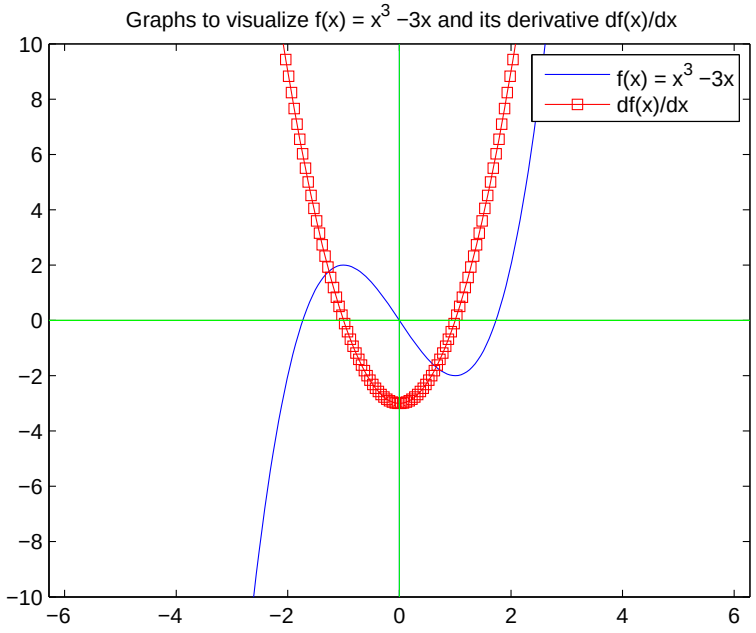
Von der Differentialrechnung wissen wir, dass die Ableitung von $f(x)$ bezüglich x leicht berechnet werden kann als:

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = 3x^2 - 3 \; .$$

In Matlab können wir zusätzlich Graphen erstellen, um die Funktion und ihre Ableitung an relevanten Punkten darzustellen. Zum Beispiel an lokalen Minima und Maxima.

```
>> f = @(x) x.^3 - 3*x  % function handle to the given function

f =

    @(x)x.^3-3*x

>> p = [1 0 -3 0] % polynomial representation of f(x)

p =
     1     0    -3     0

>> dfdx = polyder(p) % the derivative of p

dfdx =

     3     0    -3  % the coefficients of the polynomial of the derivative
                    % of f(x)

>> dfxdx = @(x) 3*x.^2 - 3 % function representation of the derivative function
```

```
dfxdx =
    @(x)3*x^2-3

>> x = linspace(-2*pi,2*pi,1e3);
>> plot(x,f(x))
>> hold on
>> plot(x,dfxdx(x), '-rs')
>> plot([0 0], [-10, 10],'-k', [-2*pi, 2*pi], [0, 0], '-g')
>> title('Graphs to visualize f(x) = x^3 -3x and its derivative df(x)/dx')
>> legend('f(x) = x^3 -3x', 'df(x)/dx')
>> axis([-2*pi 2*pi -10 10]) % setting axis limits
```



Übung 9.1

Aus Kapitel 4.3 wissen wir, dass das symbolische MATLABtool `taylor(f)` für eine gegebene symbolische Funktion $f(x)$ die Taylorreihenentwicklung um $x = 0$ zurückgibt.

Nun ist Folgendes gegeben:

```
>> syms x
>> p = taylor(sin(x)) % Taylor series of sin(x) at about x = 0

p =
    x^5/120 - x^3/6 + x
```

- Bestimmen Sie die Ableitung der Polynomentwicklung $p(x)$ und stellen Sie die Funktion $p(x)$ und ihre Ableitung für $x \in [-2\pi, 2\pi]$ dar.

`polyder` berechnet auch die Ableitung des Produktes oder des Quotienten von zwei Polynomen. Im Folgenden werden zwei Polynome `a` $\rightarrow a(x)$ und `b` $\rightarrow b(x)$ erstellt:

```
>> a = [1 3 5] ; % a(x) =  x^2 + 3x + 5
>> b = [2 4 6] ; % b(x) = 2x^2 + 4x + 6
```

Die Ableitung des Produktes `a*b` wird durch das Aufrufen von `polyder` mit einem einzigen Ausgabeargument ermittelt:

```
>> c = polyder(a,b)

c =

     8    30    56    38
```

Die Ableitung des Quotienten `a/b` wird das Aufrufen von `polyder` mit zwei Ausgabeargumenten ermittelt:

```
>> [q,d] = polyder(a,b)

q =

    -2    -8    -2


d =

     4    16    40    48    36
```

Mit `q` $\rightarrow q(x)$ und `d` $\rightarrow d(x)$ ist das Ergebnis der Ableitung $\frac{\mathrm{d}}{\mathrm{d}x}\frac{a(x)}{b(x)} = \frac{q(x)}{d(x)}$.

Übung 9.2

Gegeben ist $f(x) = \dfrac{1}{x^2+1}$.

- Bestimmen Sie die Ableitung von $f(x)$ durch Verwendung des Befehls `polyder` und stellen Sie die Funktion und ihre Ableitung in einer einzigen Abbildung dar.

9.2 Approximierte Ableitung

MATLAB bietet auch eine Funktion für eine genäherte Ableitung, basierend auf tabellarischen Daten einer Funktion, an. Diese Funktion, `diff` genannt, berechnet die Differenz zwischen den Elementen in einem Array. Die Ableitung ist definiert als

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Die Ableitung von $y = f(x)$ kann angenähert werden durch

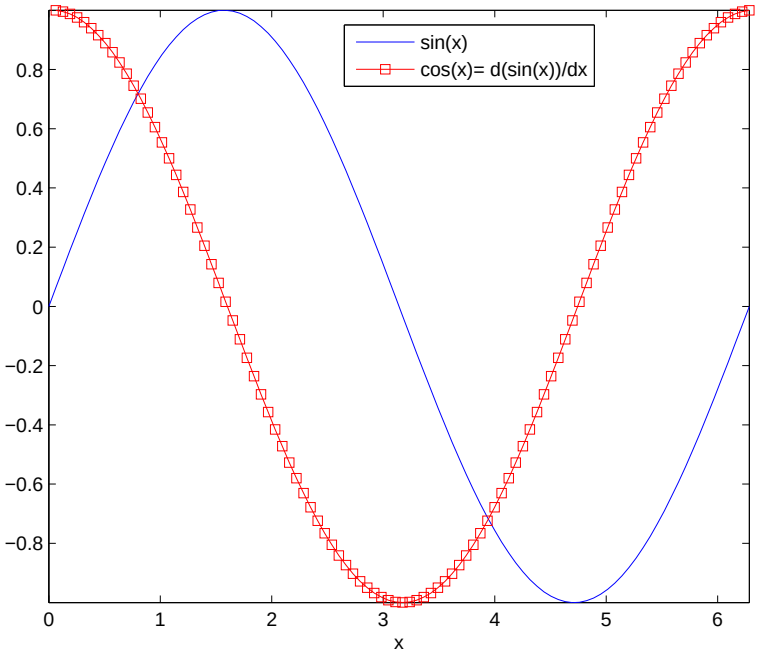
$$\frac{\mathrm{d}y}{\mathrm{d}x} \approx \frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

welches die vorwärtige Differenz von y geteilt durch die vorwärtige Differenz von x ist. Da `diff` die Differenz zwischen den Array-Elementen berechnet, kann die Ableitung in MATLAB approximiert werden. Wird das erste Element von x entfernt, dann liefert diese Prozedur eine rückwertige Differenzapproximation, welche die Informationen an den Stellen $x(n-1)$ und $x(n)$ verwendet, um sich der Ableitung an der Stelle $x(n)$ zu nähern. Auf der anderen Seite erhält man eine vorwärtige Differenzenapproximation, wenn man das letzte Element entfernt, welche $x(n+1)$ und $x(n)$ verwendet, um die Ergebnisse für $x(n)$ zu berechnen.

Beispiel 9.2.1

```
>> x = linspace(0,2*pi) ;
>> y = sin(x) ;
>> dydx = diff(y)./diff(x) ;
>> % x is linearly spaced, hence x(2)-x(1) can be used instead of diff(x)
>> xd = x(2: end) ;
>> plot(x,y,xd,dydx,'-rs') % x(1) removed, backward-difference derivation
>> axis tight
```

Figure 9.2: Backward Difference Derivative Approximation



Übung 9.3

Gegeben sind die folgenden Messergebnisse. Der k-te Element y_k des Ausgabevektors $\mathbf{y}$ wurde zu dem entsprechenden Element x_k des Eingabevektors $\mathbf{x}$ gemessen.

$\mathbf{x} = [1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10]$
$\mathbf{y} = [0.5000,\ 0.2000,\ 0.1000,\ 0.0588,\ 0.0385,\ 0.0270,\ 0.0200,\ 0.0154,\ 0.0122,\ 0.0099]$

a) Bestimmen Sie die approximierte Ableitung durch Verwendung des Befehls `dydx = diff(y)./diff(x)` und stellen Sie das Ergebnis mit `plot(xd,dydx)`, wobei `xd = x(1:end-1)` ist, dar.

b) Finden Sie eine Polynomapproximation 4. Ordnung für diese Daten durch Verwendung von `polyfit` (wie in Kapitel 4 beschrieben `p = polyfit(x,y,4)`). Werten Sie das Polynom hochauflösend im Bereich $[1, 10]$ aus und stellen Sie die gemessenen Punkte zusammen mit dem Polynom in einer einzigen Abbildung dar.

c) Bestimmen Sie die Ableitung des genäherten Polynoms mit Hilfe des Befehls `polyder`. Werten Sie das die Ableitung des Polynoms hochauflösend im Bereich [1,10] aus und stellen Sie die approximierte Ableitung zusammen mit der Ableitung des Polynoms in einer einzigen Abbildung dar.

d) Vergleichen Sie die beiden Ergebnisse für die Ableitung, indem Sie beide in die gleiche Darstellung plotten.

9.3 Symbolische Ableitung

Die Funktion `diff` berechnet auch die symbolische Ableitung einer Funktion, die durch einen symbolischen Ausdruck definiert wurde. Um einen symbolischen Ausdruck zu definieren, erstellt man zuerst eine symbolische Variable. Zum Beispiel,

```
>> syms x
>> f = sin(x)

f =

sin(x)

>> df = diff(f)

df =

cos(x)

>> g = tan(x)

g =

tan(x)

>> dg = diff(g)

dg =

tan(x)^2 + 1

>> diff(f*g)

ans =

cos(x)*tan(x) + sin(x)*(tan(x)^2 + 1)
```

Die Befehle `pretty`, `simple` oder `pretty(simple(...))` können verwendet werden, um lesbare und mathematisch vereinfachte Ausdrücke zu erhalten. Versuchen Sie es selber anhand der folgenden Beispiele.

```
>> pretty(f*g)

  sin(x) tan(x)

>> pretty(f/g)

  sin(x)
  ------
  tan(x)

>> pretty(simple(f/g))

  cos(x)

>> pretty(diff(f*g))

                                  2
  cos(x) tan(x) + sin(x) (tan(x)  + 1)

>> pretty(simple(diff(f*g)))

                                  2
  cos(x) tan(x) + sin(x) (tan(x)  + 1)
```

Chapter 10

Integration

10.1 Einführung in die numerische Integration

Numerische Integration bedeutet die numerische Auswertung von Integralen

$$J = \int_a^b f(x)\,\mathrm{d}x \tag{10.1}$$

wobei a und b gegeben sind und f eine Funktion ist, welche durch eine Formel analytisch oder durch eine Wertetabelle analytisch gegeben ist. Geometrisch gesehen, ist J die Fläche unter der Kurve f zwischen a und b.

Eine numerische Integrationsmethode kann angewendet werden, wenn einen analytische Auswertung kompliziert oder sogar unmöglich ist.

Recht-, Trapez- und Simpson Regel
Numerische Integrationsmethoden erhält man durch Approximation des Integranden f durch Funktion, die einfach integriert werden können. Die vorgestellten Integrationsmethoden basieren auf einer Abtastung der Funktion $f(x)$ mit einer konstanten Abtastperiode h. Bei einer Abtastung mit $N+1$ Abtastwerten ergibt sich die Abtastperiode zu $h = \frac{b-a}{N}$ und die Abtast(zeit)punkte x_i zu $x_0 = a,\ x_1 = a+h,\ \ldots,\ x_N = a + N \cdot h = b$.

Die einfachste Approximation, die *Rechteckregel*, erhält man, wenn der Verlauf der Funktion $f(x)$ zwischen den Abtast(zeit)punkten x_i und x_{i+1} , $i \in \{0, 1, \cdots, N\}$ durch ein Rechteck der Höhe $f(x_i)$ (Vorwärtsinterpolation) und der Breite h approximiert wird und die Fläche von $f(x)$ zwischen den Abtast(zeit)punkten x_i und x_{i+1} durch die Fläche des resultierenden Rechtecks $A_{\mathrm{R},i} = f(x_i) \cdot h$ approximiert wird. Dies entspricht der abschnittsweisen Interpolation der Funktion $f(x)$ durch ein Polynom nullter Ordnung.

Das Approximation des Integrationsergebnisses J über die Rechteckregel ergibt sich durch Aufsummierung der Flächen $A_{\mathrm{R},i}$:

$$\hat{J}_{\mathrm{R}} = \sum_{i=0}^{N-1} A_{\mathrm{R},i} = h[f(x_0) + f(x_2) + \cdots + f(x_{N-1})]. \tag{10.2}$$

Die *Trapezregel* ist üblicherweise genauer als die Rechteckregel. Die Fläche unter der Kurve $f(x)$ zwischen a und b wird hierbei approximiert durch N trapezförmige Flächen.

Dies entspricht einer abschnittsweisen Interpolation der Funktion $f(x)$ durch ein Polynom erster Ordnung (lineare Interpolation). Die Fläche von $f(x)$ zwischen den Abtast(zeit)punkten x_i und x_{i+1} wird hierbei durch die Fläche des resultierenden Trapezs $A_{\mathrm{T},i} = A_{\mathrm{R},i} + \frac{1}{2}h \cdot [f(x_i) - f(x_{i+1})1] = \frac{1}{2}h \cdot [f(x_i) + f(x_{i+1})]$ approximiert. Das Approximation des Integrationsergebnisses J über die Rechteckregel ergibt sich durch Aufsummierung der Flächen $A_{\mathrm{T},i}$:

$$\hat{J}_{\mathrm{T}} = \sum_{i=0}^{N-1} A_{\mathrm{T},i} = h \left[\frac{1}{2} f(a) + f(x_1) + f(x_2) + \cdots + f(x_{N-1}) + \frac{1}{2} f(b) \right]. \tag{10.3}$$

Stückweise konstante Interpolation von $f(x)$ führt zur Rechteckregel (10.2), stückweise lineare Interpolation zur Trapezregel (10.3) eine stückweise quadratische Interpolation ergibt die *Simpsonregel* und eine stückweise kubischeische Interpolation ergibt die *Simpsons 3/8-Regel*, welche von großer praktischer Bedeutung sind, da sie für die meisten Problemstellungen ausreichend genau sind.

10.2 Numerische Integration mit Hilfe von Matlab

Die Funktion `trapz` führt eine numerische Integration unter Verwendung der Trapez-Methode durch.
`Z = trapz(Y)` berechnet eine Approximation des Integrals von `Y` mittels der Trapez-Methode (mit einheitlichem Abstand, $h = 1$). Um das Integral mit anderen Abständen als $h = 1$ zu berechnen, multipliziert man `Z` mit einem Abstandwert. Die Eingabe `Y` kann komplex sein.
`Z = trapz(X,Y)` berechnet das Integral von `Y` bezüglich `X` mit Hilfe der trapezförmigen Integration. Eingaben `X` und `Y` können komplex sein.

Beispiel 10.2.1

Der exakte Wert von $\int_0^{\pi} \sin(x)\,\mathrm{d}x$ ist 2.
Eine numerische Näherung mit Hilfe der Trapez-Methode für ein einheitliches Raster lautet:

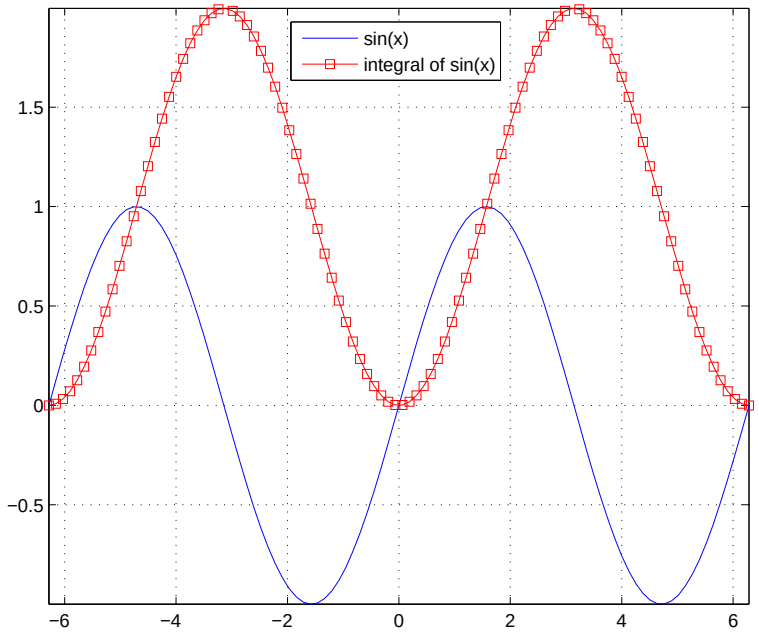
```
>> x = 0:pi/100:pi;
>> y = sin(x) ;
>> Area = trapz(x,y) % This integral is the area under the curve of sin(x)
 for x b/n 0 and pi

Area =

    1.9998
```

Die Funktion `cumtrapz` führt eine kumulative trapezförmige numerische Integration durch.

Beispiel 10.2.2
Gegeben ist eine Sinus-Funktion zwischen -2π und 2π und wir möchten die kumulative trapezförmige numerische Integration bestimmen und einen Plot erstellen, welcher das Ergebnis darstellt.

```
>> x = linspace(-2*pi, 2*pi,100) ;
>> y = sin(x) ;
>> z = cumtrapz(x,y) ;
>> figure(2)
>> plot(x,y,x,z,'-rs'), grid on
>> axis tight
```

Figure 10.2: Function cumtrapz performs cumulative trapezoidal numerical integration



Beispiel 10.2.3
Gegeben ist die Funktion:

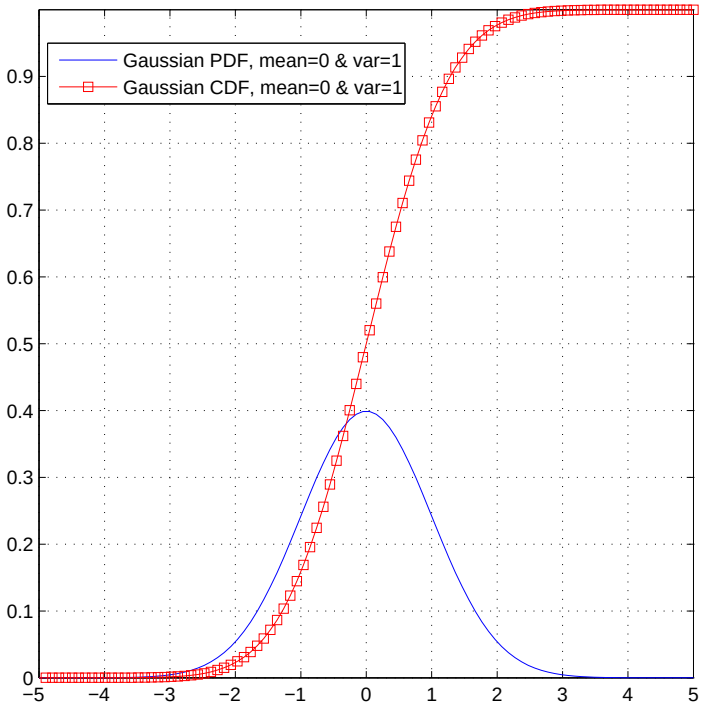
$$f_x(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

Diese Gleichung ist die Wahrscheinlichkeitsdichtefunktion (PDF) einer gaußverteilten Zufallsvariablen mit Mittelwert 0 und der Varianz 1.

Die kumulative Verteilungsfunktion (CDF), auch $F_x(x) = \int_{-\infty}^{x} f_x(u)\,\mathrm{d}u$ lässt sich z.B. mit Hilfe der kumulativen trapezförmigen numerischen Integrationsmethode berechnen.

```
>> fx = @(x)exp(-x.^2/2)/sqrt(2*pi);
>> x  = linspace(-5, 5, 100) ;
>> y  = fx(x) ;
>> z  = cumtrapz(x,y) ; % z = CDF and y = fx(x) is the PDF
>> figure(1), plot(x,y,x,z,'-rs'), grid on, axis tight
```

Fig. 10.1: Given the PDF of Normal distribtion, its CDF is calculated using cumtrapz



Übung 10.1

Gegeben ist:

$$y(x) = \sin^2(x)\,, \quad x \in [-2\pi\, 2\pi]$$

- Plotten Sie $\int_{-2\pi}^{x} y(u)\,\mathrm{d}u$ für $x \in [-2\pi\, 2\pi]$ und $y = \sin^2(x)$ in einer Abbildung. Verwenden Sie die kumulative trapezförmige numerische Integrationsmethode.

Für analytisch gegebene Funktionen ist die MATLAB-Funktion `quad` ist genauer als die `trapz`-Funktion. Während die `trapz` die trapezförmige Integrationsmethode mit fester, vom Nutzer vorgegebener Abtastperiode verwendet, verwendet `quad` die adaptive Simpson Quadratur-Methode verwendet (basierend auf der stückweisen quadratischen Interpolation, *Simpsonregel*). Hier wird adaptiv die Abtastperiode auf die gegebene Funktion angepasst. Hier ist die MATLAB-Syntax für `quad` aus einem Auszug der MATLAB-Hilfe:

```
quad
Numerically evaluate integral, adaptive Simpson quadrature

Syntax
q = quad(fun,a,b)

Description
Quadrature is a numerical method used to find the area under the graph
of a function, that is, to compute a definite integral.

q = quad(fun,a,b) tries to approximate the integral of function fun from a to b
to within an error of 1e-6 using recursive adaptive Simpson quadrature.
fun is a function handle.
```

Beispiel 10.2.4

Gegeben ist die Funktion

$$f_x(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right).$$

Im Folgenden wird das Integral $I = \int_0^{10} f_x(x)\,\mathrm{d}x$ bestimmt.

```
>> fx   = @(x)exp(-x.^2/2)/sqrt(2*pi)

>> I = quad(fx, 0, 10)

I =

    0.5000
```

Übung 10.2

Berechnen Sie die folgenden Integrale mit Hilfe von `quad`.

a)
$$\int_{-2\pi}^{2\pi} \sin^2(x)\,\mathrm{d}x$$

b)
$$\int_{-1}^{1} x\mathrm{e}^x\,\mathrm{d}x$$

c) Es gilt analytisch exakt:
$$\int_{-1}^{1} \frac{1}{1+x^2}\,\mathrm{d}x = \frac{\pi}{2}.$$

Bestimmen Sie den resultierenden Näherungsfehler bei der Verwendung von `quad`. Tasten Sie den Integrand im Bereich $x \in [-1, 1]$ mit 100 Abtastwerten ab und bestimmen Sie jeweils den resultierenden Näherungsfehler bei der Verwendung von der Rechteckregel und Trapezregel.

10.2.1 Zweidimensionale Integration analytischer Funktionen

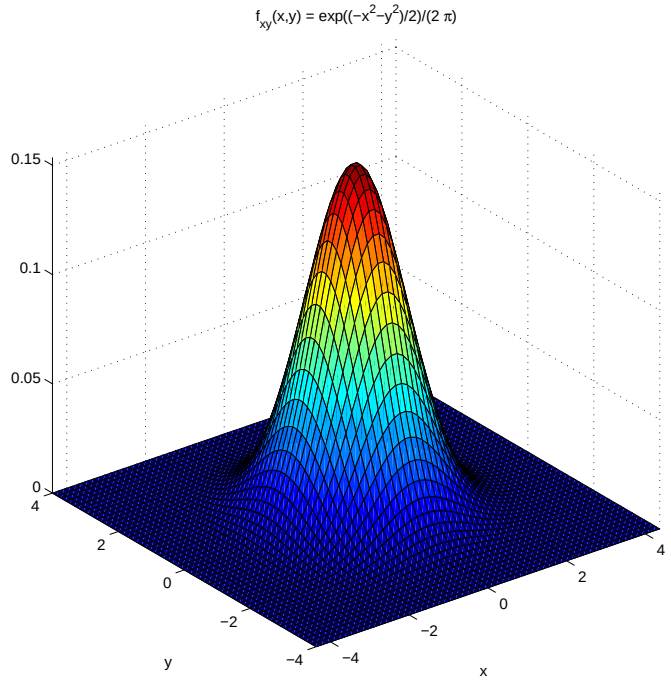
Für die Berechnung eines Doppelintegrals einer analytisch gegebenen Funktion kann man folgende Syntax verwenden:
`q = dblquad(fun,xmin,xmax,ymin,ymax)` ruft die `quad`-Funktion auf, um das Doppelintegral
$$I = \int_{y_{\min}}^{y_{\max}} \int_{x_{\min}}^{x_{\max}} f(x,y)\,\mathrm{d}x\,\mathrm{d}y$$
über der Rechteckregion $x_{\min} \le x \le x_{\max}$, $y_{\min} \le y \le y_{\max}$ auszuwerten.
`q = dblquad(fun,xmin,xmax,ymin,ymax,tol)` verwendet eine Toleranz `tol` anstatt dem Standardwert, welcher 10^{-6} ist.

Darüber hinaus gibt es noch für die numerische Berechnung Dreifachintegralen die Funktionen `triplquad`. Für detailliertere Syntax-Information wird auf die MATLAB-Hilfe verwiesen.

Übung 10.3

Betrachtet wird die folgende Funktion

$$f_{xy}(x,y)\frac{1}{2\pi}\exp\left(-\frac{x^2+y^2}{2}\right)$$



Gesucht ist das Volumen unter der Oberfläche $z = f_{xy}(x,y)$ oberhalb einer quadratischen Region R in der xy-Ebene. Die Region R ist gegeben durch

$$R : -2\pi \leq x \leq 2\pi \quad , -2\pi \leq y \leq 2\pi.$$

- Bestimmen Sie das Doppelintegral von $f_{xy}(x,y)$

$$\iint_R f_{xy}(x,y)\mathrm{d}x\mathrm{d}y$$

  für die gegebene Region R.

10.2.2 Zweidimensionale Integration auf Basis der Abtastwerte

Im Folgenden wird die gegebene Funktion $f(x, y)$ äquidistant mit $M+1$ Abtastwerten im Bereich $x_{\min} \leq x \leq x_{\max}$ und $N+1$ Abtastwerten im Bereich $y_{\min} \leq y \leq y_{\max}$ abgetastet. Es ergeben sich die Abtastperioden $\Delta x = \frac{x_{\max}-x_{\min}}{N}$ (in x-Richtung) und $\Delta y = \frac{y_{\max}-y_{\min}}{N}$ (in y-Richtung).
Damit ist der m-te Abtast(zeit-)punkt in x-Richtung ist $x_m = x_{\min} + m \cdot \Delta x$ mit $m \in \{0, 1, \cdots, M+1\}$ und der n-te Abtast(zeit-)punkt in y-Richtung ist $y_n = y_{\min} + n \cdot \Delta y$ mit $n \in \{0, 1, \cdots, N+1\}$. Die Überführung der Integration mit infinitesimal kleinen Abständen $\mathrm{d}x$ und $\mathrm{d}y$ durch eine Approximation gemäß der Rechteckregel durch Summation mit Δx und Δy ergibt ($\int \to \sum, \mathrm{d}x \to \Delta x, \mathrm{d}y \to \Delta y$):

$$I = \int_{y_{\min}}^{y_{\max}} \int_{x_{\min}}^{x_{\max}} f(x, y)\, \mathrm{d}x\, \mathrm{d}y \approx \sum_{n=0}^{N} \sum_{m=0}^{M} f(x_m, y_n) \cdot \Delta x \cdot \Delta y$$

Im Eindimensionalen ergibt sich durch diese Überlegungen die bekannte Rechteckregel mit $\Delta x = h$:

$$I = \int_{x_{\min}}^{x_{\max}} f(x)\, \mathrm{d}x \approx \sum_{m=0}^{M} f(x_m) \cdot \Delta x.$$

Für die in Übung 10.3 betrachte Funktion lässt sich die Integration entsprechend wie folgt berechnen:

```
fxy = @(x,y)1/(2*pi)*exp(-(x.^2+y.^2)/2)
vX = linspace(-2*pi,2*pi,15); % 15 Abtastwerte in x-Richtung
vY = linspace(-2*pi,2*pi,20); % 20 Abtastwerte in y-Richtung
[mX,mY] = meshgrid(vX,vY); % Überführung in ein 2D-Grid
mFxy = fxy(mX,mY); % Bestimmung der Abtastwerte (2D-Matrix wegen 2D-Grid)
dx = vX(2)-vX(1); % Abtastperiode in x-Richtung
dy = vY(2)-vY(1); % Abtastperiode in y-Richtung
I  = sum(sum(mFxy))*dx*dy % Integration durch Summation (Doppelsumme)
```

10.2.3 Zweidimensionale Integration in Polarkoordinaten

Es wird nun das Integral über eine Fläche $\tilde{R}$ in Polarkoordinaten bestimmt.
Die Überführung des Integrals einer Funktion $f(x, y)$ aus karthesischen Koordinaten in Polarkoordinaten ($x,\ y\ \rightarrow\ r,\ \theta$) wird wie folgt durchgeführt:

$$\iint\limits_{\tilde{R}} f(x,y)\ \mathrm{d}x\ \mathrm{d}y = \iint\limits_{\tilde{R}} f(r\cos(\theta), r\sin(\theta)) \cdot \left|\frac{\partial(x,y)}{\partial(r,\theta)}\right|\ \mathrm{d}r\ \mathrm{d}\theta$$

wobei $J = \left|\frac{\partial(x,y)}{\partial(r,\theta)}\right|$ die Determinante der Jacobimatrix ist (Funktionaldeterminante).
Dabei werden die karthesischen Koordinaten x und y in die Polarkoordinaten r und θ mit den Beziehungen

$$x = r \cdot \cos(\theta) \quad y = r \cdot \sin(\theta)$$

transformiert. Die Funktionaldeterminante lautet:

$$J = \left|\frac{\partial(x,y)}{\partial(r,\theta)}\right| = \begin{vmatrix} \cos(\theta) & -r\sin(\theta) \\ \sin(\theta) & r\cos(\theta) \end{vmatrix} = r$$

Übung 10.4

Betrachtet wird als Integrationsfläche eine Kreisfläche $\tilde{R} : 0 \leq r \leq 3, 0 \leq \theta \leq 2\pi$.
Die gegebene Funktion lautet wieder:

$$f_{xy}(x,y) = \frac{1}{2\pi}\exp\left(-\frac{x^2+y^2}{2}\right)\ .$$

- Überführen Sie die Funktion $f_{xy}(x, y)$ in Polarkoordinaten.
- Bestimmen Sie das Doppelintegral der Funktion über $\tilde{R}$.
- Tasten Sie die Funktion $f_{xy}(x, y)$ in Polarkoordinaten 20 Abtastwerten im Bereich $0 \leq r \leq 3$ und mit 50 Abtastwerten im Bereich $0 \leq \theta \leq 2\pi$ ab. Bestimmen Sie eine Approximation des Doppelintegrals über die Rechteckregel (Doppelsumme).

10.3 Symbolische Integration

Die 'Symbolic Math Toolbox' beeinhaltet eine Funktion namens `int` für die symbolische Integration.

Syntax und Beschreibung
`int(S,v)` liefert das unbestimmte Integral `S` bezüglich der symbolischen skalaran Variable `v`.
`int(S)` liefert das unbestimmte Integral `S` bezüglich seiner symbolischen Variablen im Sinne von `findsym`.
`int(S,v,a,b)` liefert das bestimmte Integral `S` bezüglich `v` von `a` bis `b`.
`int(S,a,b)` liefert das bestimmte Integral von `a` bis `b` von `S` bezüglich der symbolischen Standardvariablen. `a` und `b` sind symbolisch oder 'double'-Skalare.

Beispiel 10.3.1

```
>> syms x
>> int(-2*x/(1+x^2)^2)

ans =

1/(x^2 + 1)

>> int(-2*x/(1+x^2)^2, x)

ans =

1/(x^2 + 1)

>> syms z
>> int(x/(1+z^2),z)

ans =

x*atan(z)
```

Übung 10.5

Berechnen Sie die folgenden Integrale mit Hilfe des 'Symbolic Toolbox'-Befehls `int`.

a)
$$\int x \ln(1+x)\,\mathrm{d}x$$

b)
$$\int_0^1 x \ln(x)\,\mathrm{d}x$$

c)
$$\int_{\sin(t)}^1 2x\,\mathrm{d}x$$

Chapter 11

Gewöhnliche Differentialgleichungen

Dieses Kapitel beschreibt, wie man die in MATLAB integrierten numerischen Löser für gewöhnliche Differentialgleichungen (ODE 'ordinary differential equation') verwendet, um approximierte Lösungen für einzelne ODE oder Systeme von ODEs zu erhalten.

11.1 Einleitung

Eine gewöhnliche Differentialgleichung (ODE) ist eine Gleichung, die die Abhängigkeit der ersten Ableitung oder einer höheren Ableitung (n-te Ableitung) einer unbekannten Funktion, $y(t)$ oder $y(x)$ von dem Funktionswert $y(t)$ bzw. $y(x)$, dem Stimulus t bzw. x sowie im Allgemeinen von den Ableitungen der Ordnung $(n-1), (n-2), \ldots$ darstellt. Das Ziel ist, aus dieser Differentialgleichung die direkte Abhängigkeit des Funktionswert $y(t)$ bzw. $y(x)$ vom Stimulus t bzw. x zu bestimmen.

Zum Beispiel sind sind

$$y' = \cos(x),$$
$$y'' + 4y = 0,$$
$$x^2 y''' y' + 2\exp(x) y'' = (x^2+2)y^2$$

gewöhnliche Differentialgleichungen. Die erste ist erster Ordnung, die zweite zweiter Ordnung und die letzte ist eine ODE dritter Ordnung. Das Wort *gewöhnlich* unterscheidet sie von partiellen Differentialgleichungen, welche eine unbekannte Funktion mit zwei oder mehreren Variablen und ihrer partiellen Ableitung beinhaltet. In dieser Veranstaltung betragten wir ausschließlich ODEs.

Eine ODE mit Anfangsbedingungen wird *Anfangswertproblem* genannt. Eine ODE erster Ordnung wird linear genannt, wenn es als $y' + f(x)y = g(x)$ geschrieben kann. Beispiele für lineare ODEs erster Ordnung sind:

$y' - xy$	$= 0$	homogen
$xy' + 2y$	$= \exp(x)$	inhomogen
$y' + 4y$	$= 0$	mit konstanten Koeffizienten
$y' + 2y$	$= 2x^2 - 4$	inhomogen, mit konstanten Koeffizienten

Ein Beispiel für eine nicht-lineare ODE erster Ordnung ist:

$$y' = y^2$$

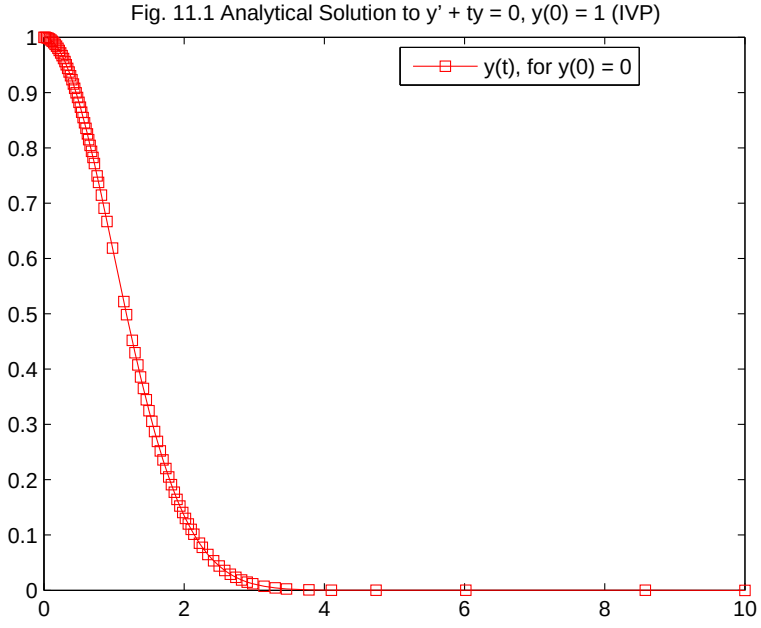
Die grundlegendste analytische Methode (Trennung der Variablen) kann mit Hilf des folgenden Anfangswertproblem erklärt werden:

$$y' + ty = 0; \qquad y(0) = 1 \; .$$

Lösung:

$$y' + ty = 0$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = -ty$$
$$\frac{\mathrm{d}y}{y} = -t \; \mathrm{d}t$$
$$\int \frac{\mathrm{d}y}{y} = -\int t \; \mathrm{d}t$$
$$\Rightarrow \ln|y| = -\frac{t^2}{2} + C_1$$
$$\Leftrightarrow |y| = \mathrm{e}^{-\frac{t^2}{2}+C_1}$$
$$\Leftrightarrow y = C\mathrm{e}^{-\frac{t^2}{2}}$$
$$\text{mit } C = \pm\mathrm{e}^{\mathrm{C}_1} \in \mathbb{R}$$
$$\text{Mit der Anfangswertbedingung}, y(0) = 1 \Rightarrow C = 1$$
$$y = \mathrm{e}^{-\frac{t^2}{2}}$$

Fig. 11.1 Analytical Solution to y' + ty = 0, y(0) = 1 (IVP)



Übung 11.1

Gesucht ist die allgemeine analytische Lösung des Anfangswertproblems:

$$y' + y = 2t,\ y(0) = 0$$

a) Bestimmen Sie zunächst die Lösung der homogenen Differentialgleichung $y' + y = 0$.

b) Bestimmen Sie darauf folgend die partikuläre Lösung der inhomogenen Differentialgleichung $y' + y = 2t$ sowie darauf basierend die Lösung des gegebenen Anfangswertproblems $y' + y = 2t,\ y(0) = 0$.

11.2 Numerische Methoden um Anfangswertprobleme (AWP) von ODEs zu lösen

Numerische Methoden sind von großer Bedeutung, da viele Praktische Probleme oft zu Differentialgleichungen führen, die möglicherweise nicht analytisch gelöst werden können. Ein Anfangswertproblem besteht aus einer Differentialgleichung und einer Bedingung, welche die Lösung erfüllen muss. Sie kann ausgedrückt werden als:

$$y' = f(x, y), \quad y(x_0) = y_0\ ,$$

vorausgesetzt, dass f so beschaffen ist, dass das Problem eine eindeutige Lösung auf einem Intervall besitzt, welches x_0 beinhaltet.

Euler Verfahren (Schritt für Schritt Verfahren)

Gegeben ist ein AWP:

$$y' = f(x, y), \quad y(x_0) = y_0, \quad a \leq x \leq b\ .$$

Der erste Schitt ist es das Intervall in n gleiche Schritte zu unterteilen:

$$h = \frac{b - a}{n}\ ,$$

h wird Schrittweite genannt. Euler's Schritt für Schritt Methode beginnt vom gegebenen $y_0 = y(x_0)$ und läuft wie folgt ab:

$x_1 = x_0 + h, x_2 = x_0 + 2h, x_3 = x_0 + 3h, \ldots$, mit der Schrittweite h, welche eine feste Zahl ist, zum Beispiel 0.2 oder 0.1.

Sei $P_0 = (x_0, y_0)$ ein Punkt auf der exakten Lösungskurve. Die Tangente bei P_0 approximiert die Lösung im Intervall $x_0 \leq x \leq x_1$. Wenn m_0 die Steigung der Tangenten im Punkt P_0 ist, was $y'(x_0)$ entspricht, dann lautet die Gleichung der Tangenten:

$$\frac{y - y_0}{x - x_0} = f(x_0, y_0) \Rightarrow \qquad y = y_0 + (x - x_0) \cdot f(x_0, y_0)\ .$$

Am Punkt $x_1 = x_0 + h$ lautet die Gleichung der Tangenten:

$$y_1 = y_0 + (x_1 - x_0) \cdot f(x_0, y_0) = y_0 + h \cdot f(x_0, y_0) \ .$$

y_1 ist der approximierte Wert der Lösung bei x_1.

$$\begin{aligned} y(x_1) &\approx y_1 = y_0 + h \cdot f(x_0, y_0) \\ y(x_2) &\approx y_2 = y_1 + h \cdot f(x_1, y_1) \\ &\vdots \\ y(x_k) &\approx y_k = y_{k-1} + h \cdot f(x_{k-1}, y_{k-1}) \end{aligned}$$

Runge-Kutta-Verfahren (4-ter Ordnung)
Dieses ist ein genaueres Verfahren und von großer praktischer Bedeutung. Einige der MATLAB ODE Löser verwenden diese Methode, um Anfangswertprobleme lösen. Dieses wird im nächsten Abschnitt behandelt.
Wie das Euler Verfahren, löst dieses Verfahren Anfangswertprobleme der Form:

$$y' = f(x, y), \quad y(x_0) = 0 \ .$$

Die Schritte, um $y(x_1) \approx y_1$ zu bestimmen sind:

$$\begin{aligned} y(x_1) \approx y_1 &= y_0 + \frac{h}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4) \\ k_2 &= f(x_0 + \frac{h}{2}, y_0 + \frac{h}{2} \cdot k_1) \\ k_3 &= f(x_0 + \frac{h}{2}, y_0 + \frac{h}{2} \cdot k_2) \\ k_4 &= f(x_0 + h, y_0 + h \cdot k_3) \end{aligned}$$

Die nächste Approximation $y(x_2) \approx y_2$ wird genauso bestimmt wie oben:

$$\begin{aligned} y(x_2) \approx y_2 &= y_1 + \frac{h}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= f(x_1, y_1) \\ k_2 &= f(x_1 + \frac{h}{2}, y_1 + \frac{h}{2} \cdot k_1) \\ k_3 &= f(x_1 + \frac{h}{2}, y_1 + \frac{h}{2} \cdot k_2) \\ k_4 &= f(x_1 + h, y_1 + h \cdot k_3) \end{aligned}$$

Diese Prozedur wird bis zum letzten Wert fortgesetzt.

$$y(x_n) \approx y_n = y_{n-1} + \frac{h}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4) \ .$$

Beide Verfahren, die hier behandelt werden, sind *Ein-Schritt*-Verfahren. Das bedeutet, dass jeder Schritt nur Werte vom vorangegangenen Schritt verwendet.
Im Gegensatz dazu verwenden Mehr-Schritt-Verfahren Werte von mehr als einem vorhergegangenem Schritt. Adams-Bashforth und Adams-Moulton sind solche Methoden.

11.3 ODEs lösen in Matlab

ODEs erster Ordnung

Eine gewöhnliche Differentialgleichung (ODE) beinhaltet eine oder mehrere Ableitungen einer abhängigen Variable y bezüglich einer einzelnen unabhängigen Variablen t, üblicherweise als Zeit bezeichnet. Die Ableitung von y bezüglich t wird als y' bezeichnet, die zweite Ableitung y'', usw.. Oft ist $y(t)$ ein Vektor mit den Elementen $y1, y2, \cdots, yn$. MATLABs ODE-Löser `ode45` ist die beste Funktion, welche man als ersten Versuch anwenden kann. Besonders für nicht-steife Probleme. Er verwendet die numerische Runge-Kutta Methode. Der `ode15s` wird für steife Problemstellungen verwendet. MATLAB biete die folgenden Funktionen zum Lösen von Anfangswertproblemen an:

`ode23`, `ode45`, `ode113` , `ode15s` , `ode23s` , `ode23t`, `ode23tb` and `ode15i`.

Detaillierte Informationen über jeden einzelnen Löser können in der MATLAB-Hilfe nachgelesen werden. MATLAB-Löser können die folgenden Typen von ODEs erster Ordnung lösen:

- Explizite ODEs der Form $y' = f(t, y)$
- Lineare implizite ODEs der Form $M(t, y)y' = f(t, y)$, wobei $M(t, y)$ eine Matrix ist
- Vollständig implizite ODEs der Form $f(t, y, y') = 0$ (nur `ode15i`)

Löser Syntax

Alle der ODE-Lösungsfunktionen, ausgenommen `ode15i`, teilen eine Syntax, was es einfach macht, die unterschiedlichen numerischen Verfahren auszuprobieren, wenn es nicht offensichtlich ist, welches das geeignetste Verfahren ist. Um eine andere Methode für die selbe Problemstellung anzuwenden, braucht man nur den Namen der ODE Lösungsfunktion zu ändern. Die einfachste Syntax, die für alle Lösungsfunktionen üblich ist, lautet:

`[T,Y] = solver(odefun,tspan,y0,options)`,

wobei `solver` eube der vorher aufgelisteten ODE-Lösungsfunktionen ist. Die grundsätzlichen Eingabeargumente sind:

- `odefun`: : ist ein 'handle' zu einer Funktion, welches das System von ODEs auswertet. Die Funktion hat die Form `dydt = odefun(t,y)`, wobei verb+t+ ein Skalar ist und `dydt` und `y` Spaltenvektoren sind.
- `tspan`: Vektor, welcher das Integrationsintervall bestimmt. Der Löser nimmt die Anfangswertbedingung bei `tspan(1)` an und integriert von `tspan(1)` bis `tspan(end)`.
- `y0`: Vektor der Anfangswertbedingung für dieses Problem.
- `options`: Struktur der optionalen Parameter, welche die standardmäßigen Integrationseigenschaften verändert.

Die Ausgabeargumente beinhalten die angenäherte Lösung an diskreten Punkten:

- `T`: Spaltenvektor der Zeitpunkte.
- `Y`: Lösungs-Array. Jede Reihe in `y` entspricht einer Lösung zu einer Zeit, welche die dazugehörige Reihe von `t` liefert.

Beispiel 11.2.1

Lösen Sie mit Hilfe des MATLAB ODE-Löser das folgende Anfangswertproblem:

$$y' + ty = 0; \qquad y(0) = 1 \ .$$

Diese Gleichung wurde im vorherigen Abschnitt analytisch gelöst (siehe Abbildung 11.1).

Hier wird `ode45` verwendet, welcher universell ist und für einen ersten Versuch der standardmäßige MATLAB ODE-Löser ist. Zuerst schreiben wir die gegebene Gleichung in die Standardform $y' = f(t, y)$:

$$y' = -ty \ .$$

Daher $f(t, y) = -ty$.

Schritt 1:
Schreibe eine Funktion für $f(t, y)$.

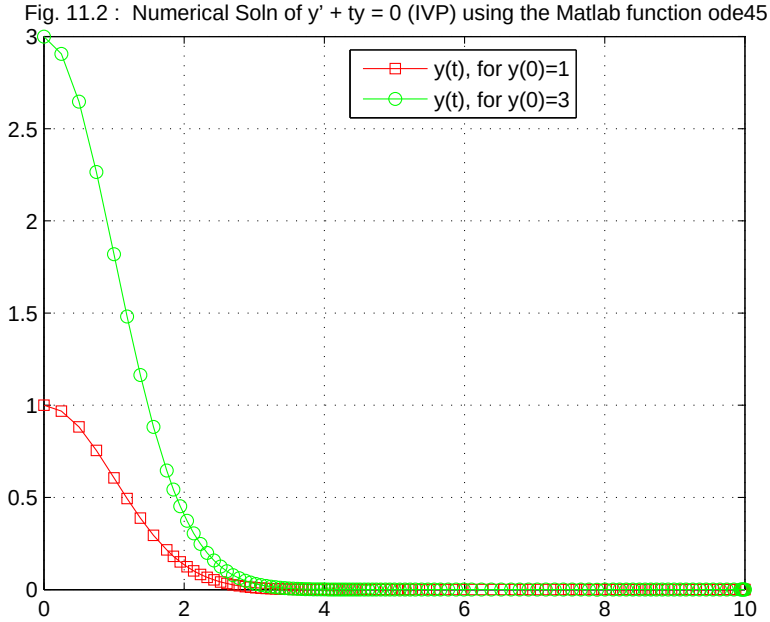
```
function dydt = odefun_1(t,y)

        dydt = -t*y;
```

Schritt 2
Aufruf des ODE-Lösers:

```
>> [T, Y] = ode45(@odefun_1, [0 10], 1);
>> plot(T,Y, '-rs'), grid on, axis tight
>> % for initial value of y(0) = 3, we will have another solution
>> [T, Y] = ode45(@odefun_1, [0 10], 3);
>> hold on; plot(T,Y, '-go'); grid on; axis tight
```

Der ODE-Löser plottet direkt bei Verwendung des Befehls `ode45(@odefun_1, [0 10], 1)` (ohne Ausgabeargumente) die Lösung des gegebenen Anfangswertproblems.
Wenn man hingegen die Resultate von verschiedenen Anfangswertbedingungen in einer bestimmten Farbe und Form haben möchte, muss man die Schritte aus Beispiel 11.2.1. verwenden.

Fig. 11.2 : Numerical Soln of y' + ty = 0 (IVP) using the Matlab function ode45



Übung 11.2

Lösen Sie die folgenden Anfangswertprobleme mit Hilfe des MATLAB ODE-Lösers `ode45` und stellen Sie die Lösung graphisch dar :

a) $y' + 5y = -26 \cdot \sin(t), \quad y(0) = 0,\ t \in [0, 20]$

b) $y' + y = \cos(t), \quad y(0) = 1,\ t \in [0, 10]$

c) $y' = y^2 \cdot \cos(t), \quad y(0) = 3,\ t \in [0, 3]$

d) $y' - y = \exp(t), \quad y(0) = 1,\ t \in [0, 10]$

Die analytische Lösung von $y' = y^2$ kann einfach durch die Methode "Trennung der Variablen" bestimmt werden.

e) Lösen Sie die folgende ODE analytisch:

$$y' = y^2, \quad y(0) = 1\ .$$

f) Lösen Sie die folgende ODE mit Hilfe des MATLAB ODE-Lösers `ode45` und stellen Sie die Lösung graphisch dar:

$$y' = y^2, \quad y(0) = 1,\ t \in \left[0, \frac{9}{10}\right]\ .$$

11.4 Systeme von ODEs erster Ordnung

Mit `ode45` oder anderen Lösern ist es ebenfalls möglich Gleichungssysteme von ODEs erster Ordnung zu lösen. Betrachten Sie das folgende System mit n Differentialgleichungen erster Ordnung.

$$\begin{aligned} y_1^{'} &= f_1(t, y_1, y_2, ..., y_n), \\ y_2^{'} &= f_2(t, y_1, y_2, ..., y_n), \\ &\vdots \\ y_n^{'} &= f_2(t, y_1, y_2, ..., y_n) \ . \end{aligned} \tag{11.1}$$

Das System (11.1) kann in Vektorform geschrieben werden:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}^{'} = \begin{bmatrix} y_1^{'} \\ y_2^{'} \\ \vdots \\ y_n^{'} \end{bmatrix} = \begin{bmatrix} f_1(t, [y_1, y_2, ..., y_n]^{\mathrm{T}} \\ f_2(t, [y_1, y_2, ..., y_n]^{\mathrm{T}} \\ \vdots \\ f_n(t, [y_1, y_2, ..., y_n]^{\mathrm{T}} \end{bmatrix} . \tag{11.2}$$

Definiert man die Vektoren $\mathbf{y} = [y_1, y_2, ..., y_n]^{\mathrm{T}}$ und $\mathbf{y}' = [\frac{\mathrm{d}y_1}{\mathrm{d}t}, \frac{\mathrm{d}y_2}{\mathrm{d}t}, \ldots, \frac{\mathrm{d}y_n}{\mathrm{d}t}]^{\mathrm{T}}$, dann wird das System (11.2) zu

$$\mathbf{y}' = \begin{bmatrix} f_1(t, \mathbf{y}^{\mathrm{T}}) \\ f_2(t, \mathbf{y}^{\mathrm{T}}) \\ \vdots \\ f_n(t, \mathbf{y}^{\mathrm{T}}) \end{bmatrix} . \tag{11.3}$$

Wenn wir schließlich $\mathbf{F}(t, \mathbf{y}^{\mathrm{T}}) = [f_1(t, \mathbf{y}^{\mathrm{T}}), f_2(t, \mathbf{y}^{\mathrm{T}}), ..., f_n(t, \mathbf{y}^{\mathrm{T}})]^{\mathrm{T}}$ definieren, kann das System (11.3) geschrieben werden als:

$$\mathbf{y}' = \mathbf{F}(t, \mathbf{y}^{\mathrm{T}}) \tag{11.4}$$

welches, was am wichtigsten ist, eine Form identisch zu einer einzelnen Differentialgleichung erster Ordnung aufweist,

$$y' = f(t, y)$$

die im vorherigen Abschnitt verwendet wurde.

Entsprechend lässt sich z.B. das Euler-Verfahren auf Systeme von ODEs erster Ordnung erweitern:

$$\mathbf{y}'(t_{k+1}) = \mathbf{y}'(t_k) + h \cdot \mathbf{F}(t_k, [\mathbf{y}(t_k)]^{\mathrm{T}}) \tag{11.5}$$

Daraus folgt: wenn die `dYdt = F(t,Y)` die erste Zeile einer MATLAB-Funktionsdatei zur Beschreibung eines Systems von n ODEs erster Ordnung ist (mit dem Namen `F.m`), dann bedeutet dies, dass `Y` ein Vektor mit den Einträgen $y_1, y_2, \ldots, y_n$ und `dYdt` ein Vektor mit den Einträgen $\frac{\mathrm{d}y_1}{\mathrm{d}t}, \frac{\mathrm{d}y_2}{\mathrm{d}t}, \ldots, \frac{\mathrm{d}y_n}{\mathrm{d}t}$ ist.

Beispiel 11.4.1

Verwenden Sie `ode45`, um das Anfangswertproblem

$$\begin{aligned} y_1' &= y_2 - y_1^2 \\ y_2' &= -y_1 - 2y_1y_2 \end{aligned} \tag{11.6}$$

im Intervall [0, 10] mit den Anfangswertbedingungen $y_1(0) = 0$ und $y_2(0) = 1$ zu lösen.

Lösung:
Wir können (11.5) als Vektorgleichung schreiben:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}' = \begin{bmatrix} y_2 - y_1^2 \\ -y_1 - 2y_1y_2 \end{bmatrix}. \tag{11.7}$$

Wenn wir $\mathbf{F}(t, [y_1,\, y_2]^{\mathrm{T}}) = [y_2 - y_1^2, -y_1 - 2y_1y_2]^{\mathrm{T}}$ und $\mathbf{y} = [y_1,\, y_2]^{\mathrm{T}}$ wählen, dann nimmt das System (11.6) die Form $\mathbf{y}' = \mathbf{F}(t, \mathbf{y})$ an. Der nächste Schritt ist es eine Funktion mit der folgenden ODE-Datei zu erstellen.

```
% Let odefun_7 be the name of the function.
function yprime = odefun_7(t,y)
yprime = zeros(2,1); % the output must be a column vector
yprime(1)= y(2) - y(1)^2 ;
yprime(2)= -y(1) - 2*y(1)*y(2) ;
```

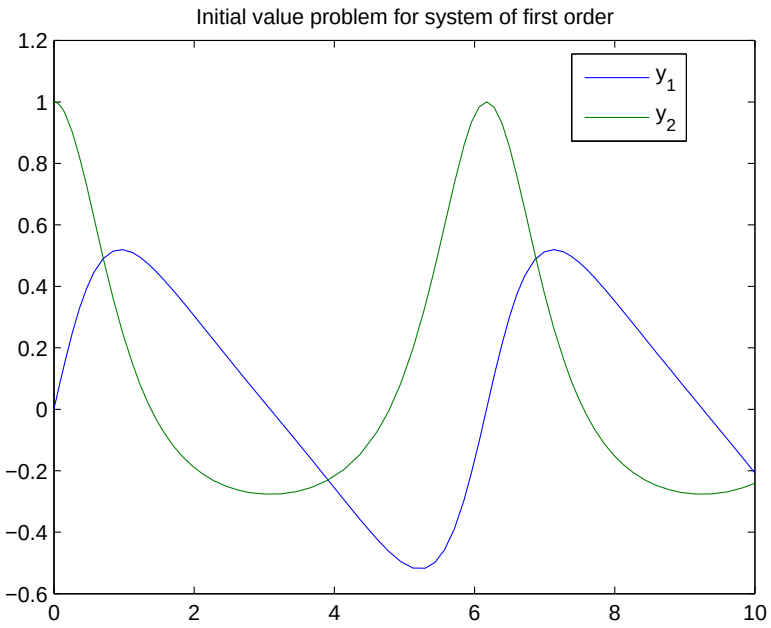
Der Vektor der Anfangswertbedingungen ist:

$$\mathbf{y}(0) = \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Nun rufen wir den ODE-Löser auf:

```
>> [t, y] = ode45(@odefun_7, [0,10], [0 ; 1]);
```

Der Befehl `plot(t,y)` liefert die folgenden Abbildung.



Übung 11.3

- Bestimmen Sie die Lösung des folgenden Anfangswertproblems mit Hilfe des MATLAB ODE-Lösers `ode45` und stellen Sie die Lösung graphisch dar:

$$y_1' = 9y_1 + 4y_2$$
$$y_2' = -2y_1 + 2y_2$$

$y_1(0) = 1,\ \ y_2(0) = -1$ über das Intervall $[0,\ 10]$.

Differentialgleichungen zweiter Ordnung

MATLAB ODE-Löser akzeptieren nur Differentialgleichungen erster Ordnung. Um die Löser mit ODEs höherer Ordnung zu verwenden, muss man jede Gleichung in ein äquivalentes System von Differentialgleichungen erster Ordnung der Form

$$y' = f(t, y)$$

umschreiben. Man kann jede ODE

$$y^n = f(t, y, y', y'', \ldots, y^{(n-1)})$$

unter der Verwendung der Substitution

$$y_1 = y,\ y_2 = y',\ \ldots, y_n = y^{(n-1)}$$

als ein Gleichungssystem erster Ordnung schreiben. Das Ergebnis ist ein äquivalentes

System von n ODEs erster Ordnung.

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ &\vdots \\ y_n' &= f(t, y_1, y_2, ..., y_n) \ . \end{aligned}$$

Beispiel 11.4.2

Gegeben: ODE zweiter Ordnung

$$y'' = f(t, y, y')$$

mit den gegebenen Anfangswerten $y(0)$ und $y'(0)$.
Lösung:
Substitution:

$$y_1 = y, \ \ y_2 = y'$$

Gleichungssystem:

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= f(t, y_1, y_2) \end{aligned}$$

Die nächsten Schritte sind wie in Beispiel 11.2.1.

Übung 11.4

- Bestimmen Sie die Lösung des folgenden Anfangswertproblems mit Hilfe des MATLAB ODE-Lösers `ode45` und stellen Sie die Lösung graphisch dar:

  $$y'' + yy' + y = 0$$

  $y(0) = 0, \ y'(0) = 1$ über das Intervall $[0, 10]$.