1 Einführung

MATLAB ist eine Simulationssprache, mit der sich vielfältige Probleme aus Naturwissenschaft und Technik lösen und veranschaulichen lassen. Der Name MATLAB steht für **Mat**rix **Lab**oratory. MATLAB wurde Ende der 1970er Jahre für die Lösung linearer Gleichungssystemen entwickelt und setzte dafür Matrizen als wichtigste Datentypen ein. Auch wenn sich MATLAB seitdem deutlich weiterentwickelt hat, sind Matrizen mit reellen oder komplexen Elementen noch immer die Grundelemente für alle Operationen.

In Naturwissenschaftr und Technik wird MATLAB häufig zur Entwicklung und Verifikation neuer Algorithmen eingesetzt und hat sich in diesem Bereich als eines der wichtigsten Werkzeuge in der Forschung etabliert. In der Industrie wird MATLAB häufig in der Vorentwicklung, also vor der eigentlichen Implementierung eines Produkts, eingesetzt, um Probleme zu analysieren und Lösungen zu entwickeln. Häufig werden Lösungen, die mit MATLAB entwickelt wurden, in C umgesetzt. Zunehmend wird MATLAB-Code mit Hilfe geeigneter Compiler auch direkt auf ein Zielsystem portiert. Dadurch lassen sich Entwicklungszeiten reduzieren.

MATLAB-Lösungen können im Wesentlichen auf zwei Arten erstellt werden: die Programmierung über Texteingabe sowie die grafische Programmierung mittels Blockdiagrammen (SIMULINK). Die Übungen zu Machine Learning konzentrieren sich auf die textuelle Eingabe von Befehlen. In diese Übung wollen wir uns mit MATLAB vertraut machen. Teile dieses Versuchs sind an [1] angelehnt.

2 MATLAB-Benutzeroberfläche

In **MS Windows** wird MATLAB üblicherweise über das Startmenü oder durch Klicken eines Icons gestartet. Dann öffnet sich die in Abbildung 1 dargestellte Benutzeroberfläche. Standardmäßig besteht sie aus den folgenden 3 Fenstern:

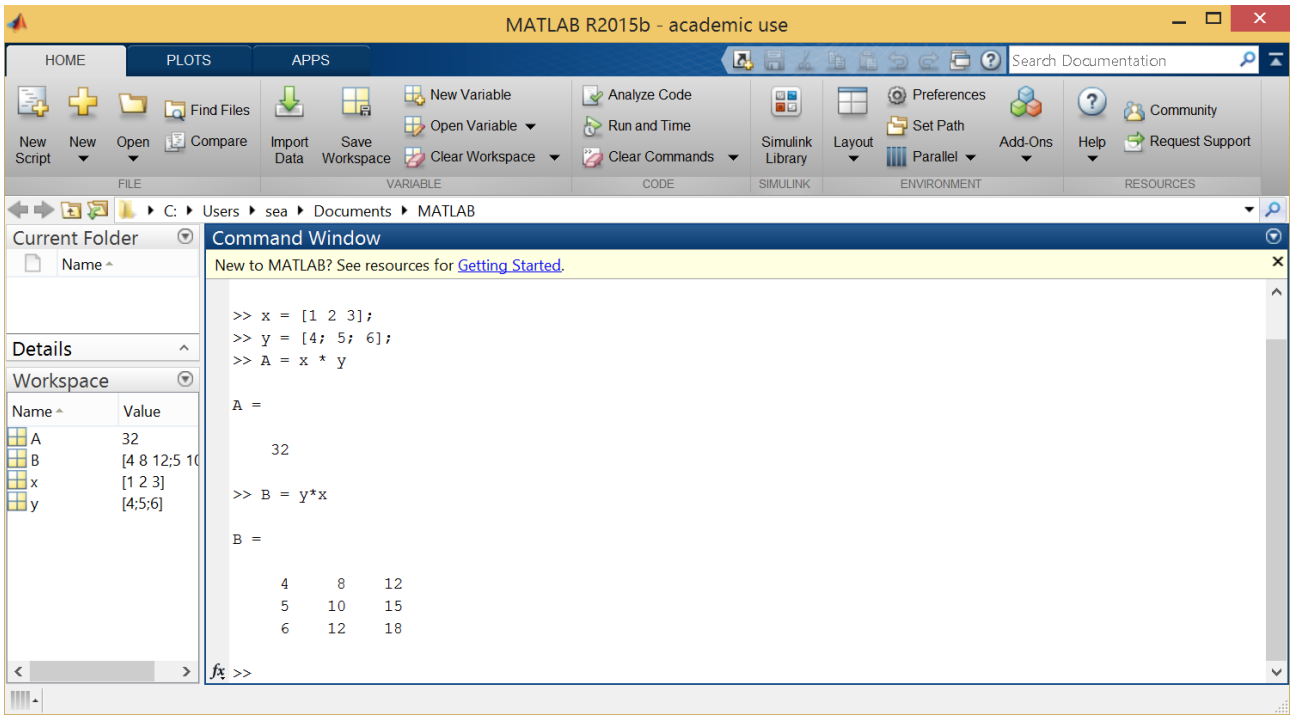

Abbildung 1: MATLAB-Benutzeroberfläche.

- **Command Window**

  Hier kann man direkt MATLAB-Befehle eingeben. Die Ergebnisse sowie etwaige Fehlermeldungen werden auch in diesem Fenster angezeigt.

Hinweis: Für ein effizientes Arbeiten mit MATLAB ist es jedoch häufig besser, die Befehle nicht direkt auf der Kommandozeile des Command Windows einzugeben, sondern ein sogenannte MATLAB-Skript anzulegen, in welches die Befehle nacheinander eingegeben werden. Dazu später mehr.

- **Current Folder**

  Zeigt das aktuelle Verzeichnis im Dateisystem an, in dem sich MATLAB befindet.

- **Workspace**

  Dieses Fenster zeigt alle Variablen an, die im MATLAB-Workspace gespeichert sind.

3 Elementare Befehle

Jetzt geben wir erste Befehle im Command Window ein. Wir können MATLAB wie einen leistungsfähigen Taschenrechner benutzen, der auch mit komplexen Zahlen rechnen kann. Wenn wir im Command Window nach dem prompt >> einfach `3+5` eingeben und die Eingabetaste drücken, erhalten wir als Antwort **`ans`**=`8`. Im Command Window steht dann:

```
>> 3+5
ans =
     8
```

Weitere Beispiele für Berechnungen auf der Kommandozeile:

```
>> 2*10
ans =
    20

>> 4/5
ans =
    0.8000

>> 2^4
ans =
    16

>> log10(100)
ans =
     2

>> sin(pi/4)
ans =
    0.7071

>> exp(j*pi/2)
ans =
   0.0000 + 1.0000i
```

Mit der Cursor-Taste ↑ können vorher eingegebene Befehle wieder auf die Kommandozeile geladen und ausgeführt werden. Mit Hilfe des "Layout"-Buttons im MATLAB "Toolstrip" (oben in der Benutzeroberfläche) können Sie sich zusätzlich zu den 3 oben besprochenen Fenstern noch das `Command History` Fenster anzeigen lassen, mit dessen Hilfe sie sehr komfortabel vorherige Eingaben auf die Kommandozeile zurückholen können.

Wenn wir eine Eingabe auf der Kommandozeile mit einem Semikolon ";" abschließen, wird die Anzeige des Ergebnisses unterdrückt. Das ist vor allem interessant, wenn wir Zwischenergebnisse nicht ausgeben wollen, z.B.:

```
>> x=5;
>> y=4;
>> z=x+5
z =
    10
```

Das obenstehende Beispiel zeigt auch, wie man in MATLAB Variablen anlegen kann. Nach Ausführung des Beispiels existieren im Workspace die Variablen `x,y,z`, wie man im Fenster Workspace erkennen kann. Wir können mit dem Befehl **`whos`** auch den Inhalt des Workspace im Command Window anzeigen lassen:

```
>> whos
  Name        Size              Bytes  Class     Attributes
  x           1x1                   8  double
  y           1x1                   8  double
  z           1x1                   8  double
```

Mit dem Befehl **`clear`** können einzelne Variablen oder auch der gesamte Workspace gelöscht werden: **`clear`** `x` löscht nur die Variable `x`, während **`clear`** ohne weitere Angaben den ganzen Workspace löscht.

Neben Variablen, die vom Nutzer angelegt werden können, gibt es in MATLAB auch einige vordefinierte Konstanten. Die wichtigsten für uns sind in Tabelle 1 zusammengefasst.

Konstante	**Symbol in Matlab**	**Erläuterung**
Kreiszahl π	**`pi`**	
Imaginäre Einheit i, j	`i, j, 1i, 1j`	`1i` und `1j` sind die bevorzugten Zeichen
Unendlich ∞	**`Inf`**, `inf`	
Not a number	**`NaN`**	Beispiel: **`Inf - Inf`**

Tabelle 1: Vordefinierte Konstanten in MATLAB.

Die folgenden Rechnungen zeigen einige Beispiele für die Verwendung der vordefinierten Konstanten:

```
>> f=1000;
>> omega=2*pi*f
omega =
   6.2832e+03
>> z=1+2j
z =
   1.0000 + 2.0000i
>> x=2+3*1j
x =
   2.0000 + 3.0000i
>> y=1/0
y =
   Inf
>> u=1/Inf
u =
```

```
     0
>> v=Inf-Inf
v =
   NaN
>> w=1^Inf
w =
   NaN
>> p=1.01^Inf
p =
   Inf
```

Hinweis: Ein häufiger Fehler beim Umgang mit MATLAB ist es, vordefinierte Konstanten (oder Funktionen) mit eigenen Variablen (oder Funktionen) zu überschreiben. Wenn man z.B. mit `i=5`, die Variable `i` definiert, dann hat `i` nicht mehr den Wert $\sqrt{-1}$, sondern den Wert 5. Das folgende Beispiel verdeutlicht das.

```
>> x=2+3*i
x =
   2.0000 + 3.0000i
>> i=5;
>> y=2+3*i
y =
    17
```

4 Operatoren, Funktionen und `help`-Befehl

In MATLAB existieren alle gängigen arithmetischen Operatoren, z.B. "+" (Addition), "−" (Subtraktion), '∗" (Multiplikation) und '/" (Division). Darüber hinaus gibt es noch eine Vielzahl weiterer Operatoren. Eine gute Möglichkeit, sich einen Überblick über die Operatoren in MATLAB zu verschaffen bietet der **`help`**-Befehl. Wenn Sie auf der Kommandozeile **`help`** `ops` eintippen, erscheint im Command Window eine Übersicht über die Operationen und Operatoren in MATLAB.

Hinweis: Ausführlichere Erklärungen bieten die eingebaute Hilfe (? in der Symbolleiste) sowie die online-Hilfe von MathWorks (Unternehmen, welches MATLAB entwickelt hat), die unter folgendem Link erreichbar ist: `http://de.mathworks.com/help/`.

Neben den Operatoren gibt es auch eine große Menge vordefinierter Funktionen. Die wichtigsten elementaren Funktionen für uns sind: **`sin`**, **`cos`**, **`exp`**, **`log`**, **`log10`**, **`real`**, **`imag`**, **`abs`**, **`angle`**. Einen Überblick über alle elementaren MATLAB-Funktionen erhält man, wenn man **`help`** `elfun` auf der Kommandozeile eingibt. In den folgenden Versuchen werden wir noch etliche Funktionen kennenlernen, die speziell für die Signalverarbeitung interessant sind.

Durchführung

Aufgabe D1:

Berechnen Sie die Werte der folgenden Ausdrücke mit MATLAB.

a) $\sin\left(\frac{\pi}{3}\right)$

b) $(2+3j)\cdot e^{j\frac{\pi}{6}}$

c) $10\cdot\log_{10}(1000)$

5 Vektoren und Matrizen

Matrizen sind die Grundelemente aller Berechnungen in MATLAB. In MATLAB werden alle Variablen als Matrizen betrachtet. Die skalaren Variablen, die wir bisher kennengelernt haben, werden intern als 1×1-Matrizen interpretiert. Das wird z.B. bei der Ausgabe des **whos**-Befehls deutlich:

```
>> x=5
x =
     5
>> whos
  Name       Size             Bytes  Class     Attributes
  x          1x1                  8  double
```

Spaltenvektoren sind $N \times 1$-Matrizen und Zeilenvektoren sind $1 \times N$-Matrizen. Beide Arten von Vektoren lassen sich durch Aufzählung ihrer Elemente innerhalb von rechteckigen Klammern "`[]`"erzeugen, z.B. werden durch die nachfolgenden beiden Befehle der Zeilenvektor `x` und der Spaltenvektor `y` erzeugt:

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

Auch Matrizen können durch Aufzählen ihrer Elemente innerhalb eckiger Klammern erzeugt werden, z.B.

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

Darüber hinaus gibt es eine Vielzahl nützlicher Befehle zum Erzeugen von Vektoren und Matrizen. Einen Überblick über die wichtigsten Befehle erhält man mit dem Befehl **help** `elmat`. Die folgenden Beispiele zeigen wichtige Anwendungen dieser Befehle:

```
>> a = 1:10
a =
     1     2     3     4     5     6     7     8     9    10
>> b = 1:0.2:2
b =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
>> c = zeros(3,1)
c =
     0
     0
     0
>> d = zeros(2,2)
d =
     0     0
     0     0
>> e = ones(3,3)
```

```
e =
     1     1     1
     1     1     1
     1     1     1
>> f = eye(3)
f =
     1     0     0
     0     1     0
     0     0     1
>> g = linspace(1,2,5)
g =
    1.0000    1.2500    1.5000    1.7500    2.0000
```

Vorbereitung

Aufgabe V1:
Vollziehen Sie alle Beispiele aus dem Abschnitt 5 nach. Befehle die nicht selbsterklärend sind, können Sie beispielsweise auf den Online-Hilfeseiten von MathWorks nachschauen.
http://de.mathworks.com/help/

Durchführung

Aufgabe D2:
Erzeugen Sie in MATLAB die folgenden Vektoren und Matrizen:

a) Einen Zeilenvektor `a` der alle Vielfachen der Zahl 4 von 4 bis 20 enthält.
b) Eine 4×4-Diagonalmatrix `D`, deren Elemente auf der Hauptdiagonale jeweils den Wert 4 haben. Diagonalmatrix bedeutet, dass Elemente, die nicht auf der Hauptdiagonalen liegen, 0 sind.
c) Eine 5×6-Matrix `A`, deren erste beide Zeilen nur Nullen und deren übrige 3 Zeilen nur Einsen enthalten.

Die meisten Matlab-Operatoren führen Matrixoperationen durch. So führen z.B. "'+", "−" und "∗" eine Matrixaddition, eine Matrixsubtraktion bzw. eine Matrixmultiplikation durch, wobei jeweils die Dimensionen der Operanden passen müssen. Dazu einige Beispiele:

```
a = [1 2 3];
>> b = [4;5;6];
>> A = a*b
A =
    32
>> B = b*a
B =
     4     8    12
     5    10    15
     6    12    18
>> C = B * b
C =
   128
   160
```

```
   192
>> D = B*a
Error using  *
Inner matrix dimensions must agree.
```

Vorbereitung

Aufgabe V2:
Warum kommt es beim letzten Beispiel zu einem Fehler?

Neben den Matrixoperationen gibt es aber auch sogenannte elementweise Operationen. Beispielsweise führt der Operator "`.*`" eine elementweise Multiplikation zweier Matrizen (oder zweier Vektoren) durch. Dazu müssen natürlich die Dimensionen der beiden Matrizen übereinstimmen. Der Operator "`.^`" führt zu einer elementweisen Potenzierung. Viele MATLAB-Funktionen können nicht nur auf skalare Werte, sondern auch auf Vektoren und Matrizen angewendet werden. Die Funktionen werden dann in der Regel auch elementweise angewendet. Zwei weitere wichtige Operatoren für Matrizen sind die Transponierung "`.'`" sowie die komplex konjugierte Transponierung "`'`". Bei reellwertigen Matrizen führen beide Operatoren zu den gleichen Ergebnissen. Im Folgenden einige Beispiele:

```
>> a = [1 2 3]
a =
     1     2     3
>> b1 = [4;5;6]
b1 =
     4
     5
     6
>> b = b1'
b =
     4     5     6
>> c = a.*b
c =
     4    10    18
>>
>> d = a.^2
d =
     1     4     9
>> t = linspace(0,pi,5)
t =
         0    0.7854    1.5708    2.3562    3.1416
>> x = sin(t)
x =
         0    0.7071    1.0000    0.7071    0.0000
>> t = linspace(pi/4,pi/2,3)
t =
    0.7854    1.1781    1.5708
>> z = exp(1j*t)
z =
   0.7071 + 0.7071i   0.3827 + 0.9239i   0.0000 + 1.0000i
>> z1 = z'
```

```
z1 =
   0.7071 - 0.7071i
   0.3827 - 0.9239i
   0.0000 - 1.0000i
>> z2 = z.'
z2 =
   0.7071 + 0.7071i
   0.3827 + 0.9239i
   0.0000 + 1.0000i
```

Vorbereitung

Aufgabe V3:
Warum wird kein Operator "`.+`" benötigt?

Wenn man nach einer Matrix runde Klammern "`()`" mit einem Index anhängt, kann man auf einzelne Matrixelemente zugreifen. Beispielsweise steht `A(2)` für das 2. Element der Matrix `A`. Im Gegensatz zur Programmiersprache C zählt man in MATLAB die Elemente beginnend mit 1 (In C beginnt die Zählung bei 0). Das ist eine häufige Fehlerquelle, wenn man MATLAB-Code in C-Code umsetzt. In Kombination mit dem "`:`"-Operator kann man auch Untermatrizen herausgreifen. Beispiele:

```
>> a = 10*[1:9]
a =
    10    20    30    40    50    60    70    80    90
>> A = reshape(a,3,3)
A =
    10    40    70
    20    50    80
    30    60    90
>> A(1)
ans =
    10
>> A(5)
ans =
    50
>> A(:,1)
ans =
    10
    20
    30
>> A(2:end,1)
ans =
    20
    30
>> A(1:2,2:3)
ans =
    40    70
    50    80
>> A([1;3],:)
ans =
```

```
    10    40    70
    30    60    90
```

Mit Hilfe von rechteckigen Klammern, lassen sich Matrizen aus Teilmatrizen zusammenbauen:

```
>> A = [ones(4,1) zeros(4,3)]
A =
     1     0     0     0
     1     0     0     0
     1     0     0     0
     1     0     0     0
>> B = [eye(2); zeros(3,2)]
B =
     1     0
     0     1
     0     0
     0     0
     0     0
```

Durchführung

Aufgabe D3:

a) Erstellen Sie in MATLAB eine 11×11-Matrix `M`, die das kleine Einmaleins mit einer Überschriftenzeile und einer Überschriftenspalte darstellt. Die Matrix soll wie folgt aussehen:

```
M =
   NaN     1     2     3     4     5     6     7     8     9    10
     1     1     2     3     4     5     6     7     8     9    10
     2     2     4     6     8    10    12    14    16    18    20
     3     3     6     9    12    15    18    21    24    27    30
     4     4     8    12    16    20    24    28    32    36    40
     5     5    10    15    20    25    30    35    40    45    50
     6     6    12    18    24    30    36    42    48    54    60
     7     7    14    21    28    35    42    49    56    63    70
     8     8    16    24    32    40    48    56    64    72    80
     9     9    18    27    36    45    54    63    72    81    90
    10    10    20    30    40    50    60    70    80    90   100
```

Hinweis: Ziel ist es, die Matrix `M` durch mathematische Operationen zu erstellen und nicht durch Aufzählen eines jeden Elements.

b) Greifen Sie jetzt aus `M` den Zeilen-Vektor `V5` des Fünfer-Einmaleins heraus (ohne Überschriften). `V5` soll also folgendermaßen aussehen:

```
V5 =
     5    10    15    20    25    30    35    40    45    50
```

c) Greifen Sie jetzt aus `M` die Matrix `M5` des Fünfer-Einmaleins mit Überschriften heraus. `M5` soll also folgendermaßen aussehen:

```
M5 =
   NaN     1     2     3     4     5     6     7     8     9    10
     5     5    10    15    20    25    30    35    40    45    50
```

6 MATLAB-Skripte und MATLAB-Funktionen

MATLAB kann Befehle nicht nur von der Kommandozeile entgegennehmen, sondern auch Befehle oder Befehlsfolgen aus einem File (Programm) lesen. Die Verwendung von Files ist vor allem bei der Bearbeitung komplexer Aufgaben empfehlenswert. Files, die MATLAB-Befehle enthalten, nennt man M–Files, da ihr Filetyp, also die Endung ihres Filenamens `.m` ist.

M–Files bestehen aus Folgen normaler MATLAB–Befehle, die ein Programm bilden. Um die Klarheit und Wartbarkeit eines Programm zu erhöhen, sind Kommentare im Programmcode hilfreich. Kommentare werden in MATLAB mit einem Prozentzeichen "`%`" eingeleitet. MATLAB ignoriert in diesem Fall den Rest der Programmzeile ab dem Kommentarzeichen.

Prinzipiell können M–Files mit jedem beliebigen Texteditor erstellt werden. Die MATLAB-Benutzeroberfläche bietet einen maßgeschneiderten Editor, mit dem man nicht nur M–Files erstellen, sondern auch starten und debuggen kann. Der MATLAB-Editor kann beispielsweise durch das große gelbe Pluszeichen links oben in der Benutzeroberfläche (im sogenannten MATLAB-Toolstrip) gestartet werden.

Es gibt zwei Arten von M–Files: *Skripte* und *Funktionen*:

- **Skripte:**
  Skripte enthalten eine Folge von MATLAB–Befehlen, die nacheinander abgearbeitet werden, als wären diese auf der Kommandozeile eingegeben worden. Von der Kommandozeile aus wird das Skriptfile durch die Eingabe des Dateinamens ohne die Erweiterung `.m` gestartet, also z.B. `meinskript`, wenn ein File `meinskript.m` im aktuellen Arbeitsverzeichnis vorhanden ist. Alternativ kann das Skript auch durch Drücken des großen grünen Pfeils im Toolstrip des MATLAB-Editors gestartet werden.

  Das folgende Beispiel zeigt ein MATLAB-Skript `Multiplikation.m`, welches eine Matrix und einen Vektor definiert und anschließend beide miteinander multipliziert:

```
%Definition der Variablen
A = [1 2; 3 4];
V = [5;6];
%Multiplikation
C = A*V
```

  Wenn das Skript auf der Kommandozeile gestartet wird, liefert es folgendes Ergebnis:

```
>> Multiplikation
C =
    17
    39
```

- **Funktionen:**
  Wenn die erste Zeile eines M–Files das Schlüsselwort **`function`** enthält ist das File eine Funktionsdatei. Eine Funktion unterscheidet sich von einem Skript dadurch, dass Argumente übergeben und Ergebnisse zurückgeliefert werden können. Außerdem sind alle Variablen, die im File verwendet werden, lokal. Das heißt, sie werden aus dem Arbeitsspeicher entfernt, sobald die Funktion abgearbeitet ist. Mit Funktionen kann der Benutzer MATLAB nach seinen Bedürfnissen erweitern. Funktionen können sowohl auf der Kommandozeile als auch von Skripten aufgerufen werden. Der Start von Funktionen mit dem grünen Pfeil des MATLAB-Editors führt in der Regel zu einem Fehler, weil auf diese Weise keine Argumente übergeben werden können.

  Das folgende Beispiel zeigt eine MATLAB-Funktion, welche den Umfang und den Flächeninhalt eines Rechtecks berechnet.

```
function [U,A]=Rechteck(l,b)
%[U,A] = Rechteck(l,b)
```

```
%Berechnet den Umfang U und die Flaeche A
%eines Rechtecks mit der Laenge l und der
%Breite b
U = 2*[l+b];
A = l*b;
end
```

Starten der Funktion `Rechteck` auf der Kommandozeile mit den Aktualparametern 4 und 5 liefert:

```
[Umfang,Flaeche]=Rechteck(4,5)
Umfang =
    18
Flaeche =
    20
```

Durchführung

Aufgabe D4:

a) Schreiben Sie eine MATLAB-Funktion `Summe`, welche alle ganzen Zahlen von 1 bis `n` addiert und als Ergebnis die berechnete Summe ausgibt. Eingabeargument ist die Variable `n`.
Hilfreicher Befehl: `sum`.

b) Schreiben Sie ein MATLAB-Skript `TestSumme`, welches die Funktion `Summe` mit den folgenden Werten aufruft: `n=10, n=100, n=2000` und testen Sie es.

7 Grafische Darstellung

Mit MATLAB lassen sich sehr gut Grafiken erstellen und Probleme veranschaulichen. Im Folgenden betrachten wir die für uns wichtigsten Möglichkeiten zur grafischen Darstellung etwas genauer.

7.1 Zweidimensionale Darstellung

Mit Hilfe des **`plot`**-Befehls lassen sich auf einfache Weise Funktionen einer Variable veranschaulichen. Der Befehl **`plot`**`(x,y)` zeichnet einen Graphen der Funktion $y = f(x)$. Dabei ist `x` ein Vektor mit allen Werten der unabhängigen Variable x und `y` ist ein Vektor mit allen Werten der abhängigen Variable y. Die beiden Vektoren müssen unbedingt die gleiche Länge haben. Der **`plot`**-Befehl zeichnet für jedes Wertepaar (x_i, y_i) einen Punkt und verbindet alle Punkte durch eine Linie. Das folgende Beispiel zeigt, wie sich in MATLAB die Funktion $y = x^2$ grafisch darstellen lässt. Neben dem **`plot`**-Befehl enthält das Beispiel noch eine Reihe weiterer Befehle zur Beschriftung und Verschönerung der Grafik.

```
%Erzeugung der Variablen x und y
x = linspace(-3,3,200); %200 Werte im Intervall [-3;3]
y = x.^2; %.^ da jedes Element aus x einzeln quadriert werden soll

%Graphische Darstellung
figure(1);  %Erzeugt ein neues Fester fuer den Graphen
clf;        %Loescht den Inhalt des neuen Fensters
plot(x,y);  %Stellt den Vektor y ueber x dar
```

```
title('Graph der Funktion y=x.\^2'); %Ueberschrift
xlabel('x \rightarrow') %Beschriftung x-Achse
ylabel('y \rightarrow') %Beschriftung y-Achse
grid on;    %Einschalten von Gitterlinien
```

Abbildung 2 zeigt die grafische Darstellung zum obigen Beispiel. Mit Hilfe der Werkzeuge im oberen Teil des Grafikfensters kann man beispielsweise in die Grafik hineinzoomen oder interaktiv die Grafik weiter bearbeiten.

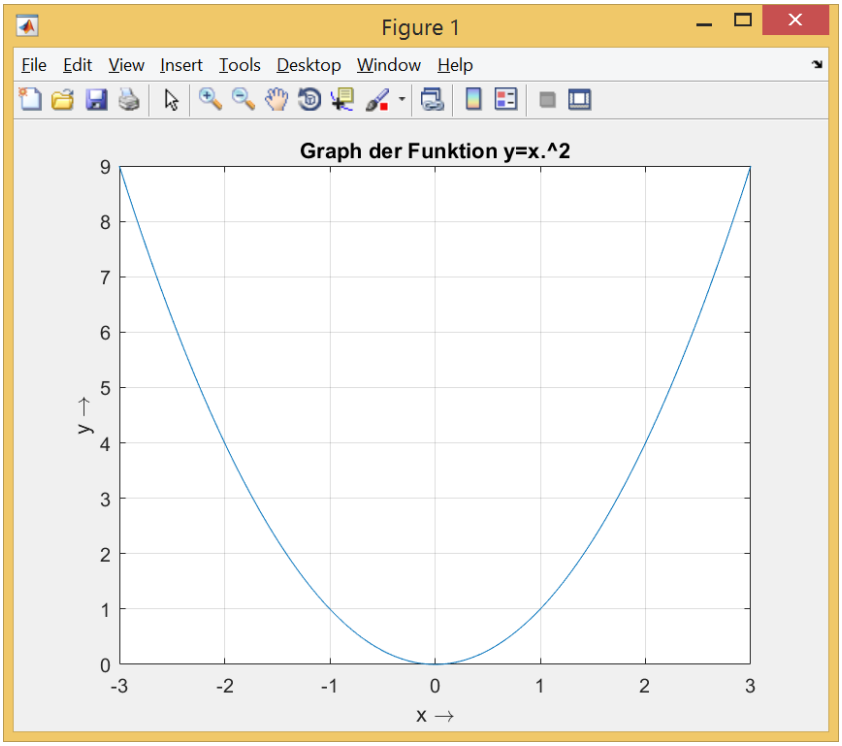

Abbildung 2: Grafische Darstellung der Funktion $y = x^2$.

Mit dem `plot`-Befehl lassen sich auch mehrere Graphen in ein Koordinatensystem zeichnen, wenn die folgende Form benutzt wird: `plot(x1,y1,x2,y2,...)`. Das folgende Beispiel zeigt dies anhand der parametrisierten Funktion $y = (2x - a)^2 + a$

```
%Erzeugung der Variablen x und y1, y2, y3
x = linspace(-4,4,200);
y1 = (2*x+1).^2-1;
y2 = (2*x).^2;
y3 = (2*x-1).^2+1;

%Graphische Darstellung
figure(1);
clf;
plot(x,y1,x,y2,x,y3,'LineWidth',2);  %3 Kurven in einem Fenster
title('Graph der Funktion y=(2*x-a).\^2+a');
xlabel('x \rightarrow')
ylabel('y \rightarrow')
grid on;
legend('a=-1','a=0','a=1')  %Hinzufuegen einer Legende
axis([-4 4 -2 10]) %Ausschneiden eines Teilbereiches (Zoom in)
```

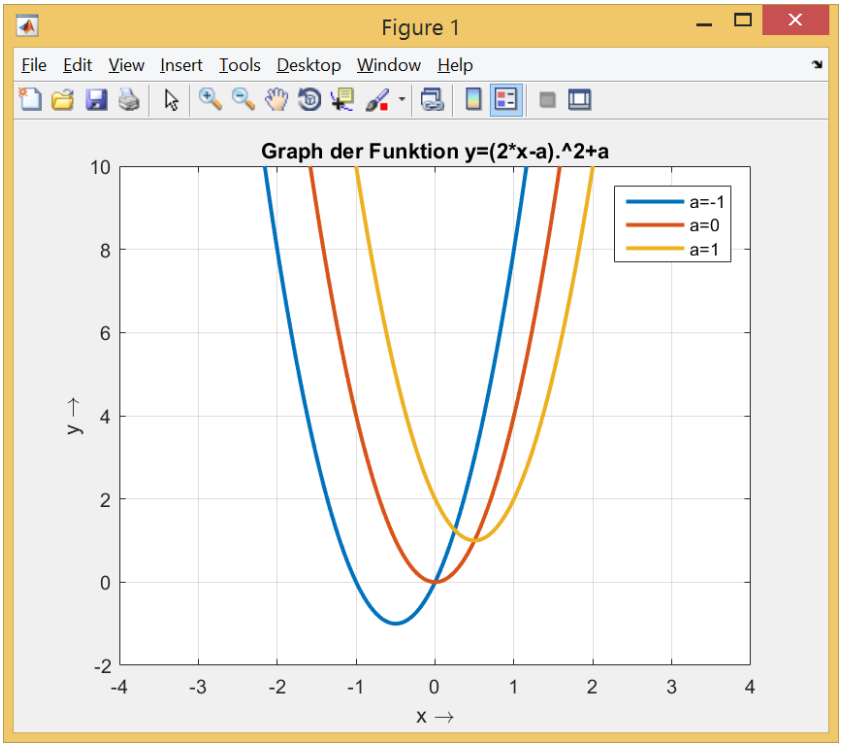

Abbildung 3: Grafische Darstellung der Funktion $y = (2x - a)^2 + a$.

Durchführung

Aufgabe D5:
Stellen Sie die Funktion $y = sin(x - a)$ im Bereich $x = -4\pi$ bis $x = 4\pi$ mit MATLAB grafisch dar. Stellen Sie dazu 3 Kurven für $a = 0$, $a = \frac{\pi}{4}$ und $a = \frac{\pi}{2}$ in einem Koordinatensystem dar. Variieren Sie die Anzahl der Stützstellen, also die Anzahl der Vektorelemente im Vektor `x` zwischen 10 und 1000 Werten. Beschriften Sie die Grafik.

7.2 Dreidimensionale Darstellung

Auch dreidimensionale Grafiken lassen sich sehr einfach mit MATLAB erzeugen. Das schauen wir uns näher an dem Beispiel $z = f(x, y) = x^2 - y^2$ an. Wir wollen hier also den Graphen einer Funktion von 2 unabhängigen Variablen zeichnen. Dazu müssen wir zunächst zwei Vektoren `x` und `y` erzeugen, welche die x-Achse und die y-Achse definieren. Mit Hilfe des Befehls **`meshgrid`** erzeugen wir dann aus `x` und `y` die Matrizen `X` und `Y`. Mit Hilfe der Formel $z = x^2 - y^2$ können wir aus `X` und `Y` die Matrix `Z` berechnen und diese dann mit dem **`mesh`**-Befehl oder dem **`surf`**-Befehl grafisch darstellen. Der folgende Code und die Abbildung 5 zeigen die genaue Vorgehensweise sowie die resultierende Grafik.

```
%Erzeugung der Variablen x und y
x = linspace(-2,2,100); %Vektor mit 100 Werten zwischen -1 und 1
y = linspace(-2,2,100); %Vektor mit 100 Werten zwischen -1 und 1
[X,Y]=meshgrid(x,y);    %Erzeugung der Matrizen X und Y aus x und y
Z = X.^2 - Y.^2;        %Berechnung der Funktion

%Graphische Darstellung
figure(1);  %Erzeugt ein neues Fester fuer den Graphen
clf;        %Loescht den Inhalt des neuen Fensters
surf(X,Y,Z);  %Stellt den Vektor y ueber x dar
```

```
shading interp; %kontinuierliche Interpolation der Farben
xlabel('x \rightarrow') %Beschriftung x-Achse
ylabel('y \rightarrow') %Beschriftung y-Achse
zlabel('z \rightarrow') %Beschriftung y-Achse
grid on;     %Einschalten von Gitterlinien
print ('-dpng', '../Bilder/Quadrat3D.png'); %png-File erstellen
```

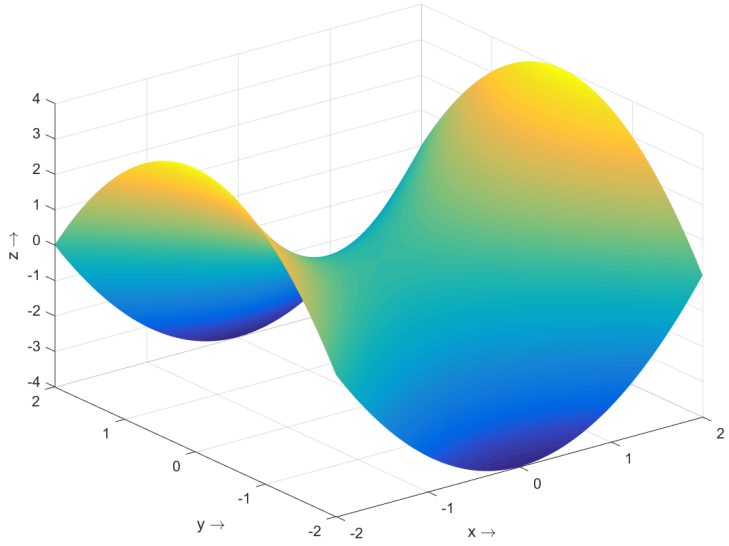

Abbildung 4: Grafische Darstellung der Funktion $y = x^2 - y^2$.

Durchführung

Aufgabe D6:
Nun soll die Funktion

$$f(x, y) = \mathrm{sinc}(2x) + \mathrm{sinc}(2y)$$

mit $\mathrm{sinc}(x) = \sin(\pi x)/(\pi x)$ dargestellt werden.
Erzeugen Sie die zu einem Stützpunktraster von $\Delta x = \Delta y = 0.1$ passenden Matrizen X und Y. Berechnen Sie damit die Matrix `F` der Funktionswerte für die Funktion $f(x, y)$. Verwenden Sie dabei den MATLAB–Befehl `sinc`. Welches Problem würde auftreten, wenn wir statt dem `sinc`-Befehl direkt mit der Rechenvorschrift $\sin(\pi x)/(\pi x)$ arbeiten würden? Stellen Sie nun die Funktion mit Hilfe des Befehls **`surf`** dar.

7.3 Darstellung von Bildern

Im Folgenden sollen Bilder mit Hilfe von MATLAB dargestellt und manipuliert werden. Dabei machen wir uns zu Nutze, dass ein digitales Bild einfach durch Matrizen repräsentiert werden kann. Während zur Beschreibung eines Farbbildes 3 Matrizen nötig sind, nämlich jeweils eine für Rot, Grün und Blau, kann ein digitales Schwarz-Weiß-Bild durch eine einzige Matrix repräsentiert werden. Dabei entspricht jedes Matrixelement einem Bildpunkt und kann Werte zwischen 0 (entspricht Schwarz) und 1 (entspricht Weiß) annehmen. Die Zwischenwerte ergeben die entsprechenden Graustufen. In der folgenden Aufgabe werden wir Schwarz-Weiß-Bilder darstellen und manipulieren.

Durchführung

Aufgabe D7:

Laden Sie mit dem Befehl `load('some_images.mat')` die folgenden Matrizen in den Workspace: `lena`, `einstein` und `noise`.

a) Stellen Sie mit den Befehlen `imshow(lena)` und `imshow(einstein)` die Matrizen `lena` und `einstein` als Bilder dar.

b) Transponieren Sie die Matrizen `lena` und `einstein` und stellen Sie dann die transponierten Matrizen als Bilder dar. Was beobachten Sie?

c) Multiplizieren Sie die Matrizen `lena` und `einstein` jeweils mit einem skalaren Wert und stellen Sie dann die skalierten Matrizen als Bilder dar. Experimentieren Sie mit unterschiedlichen skalaren Werten im Bereich 0.1 bis 2.0. Wie wirkt sich die Skalierung auf die Bilder aus?

d) Addieren Sie zu den Matrizen `lena` und `einstein` jeweils einen skalaren Wert. Experimentieren Sie mit unterschiedlichen skalaren Werten im Bereich -1.0 bis 1.0. Wie wirkt sich die Addition einer Konstanten auf die Bilder aus?

e) Addieren Sie die beiden Matrizen `lena` und `einstein` und stellen Sie dann die Ergebnisse als Bild dar. Wie sieht das Summenbild aus?

f) Stellen Sie die Matrizen `einstein`, `noise` und `einstein+noise` als Bilder dar. Wie verändert die Addition von `noise` das Bild von `einstein`?

8 Lineare Gleichungssysteme

Ein lineares Gleichungssystem mit n Gleichungen und n Unbekannten $x_1, x_2, \ldots, x_n$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{1}$$

kann kompakt in Matrixform

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{2}$$

geschrieben werden. Ein solches System heißt regulär, wenn seine Koeffizientenmatrix $\boldsymbol{A} = [a_{ij}]$ invertierbar ist. Ein reguläres System hat eine eindeutige Lösung $\boldsymbol{x} = [x_i]$, die in MATLAB entweder mit Linksdivision

```
x=A\b
```

oder durch Multiplikation mit der inversen Matrix

```
x=inv(A)*b
```

berechnet werden kann, wobei die erste Methode ein effizientes Eliminationsverfahren verwendet und deshalb vorzuziehen ist. Im Folgenden soll das anhand eines Beispiels durchgeführt werden. Außerdem veranschaulichen wir grafisch die Lösbarkeit von linearen Gleichungssystemen.

Vorbereitung

Aufgabe V4:

Berechnen Sie die Lösung des nachfolgenden linearen Gleichungssystems:

$$\begin{aligned} 2x_1 + 3x_2 + x_3 &= -4 \\ 4x_1 + x_2 + 4x_3 &= 9 \\ 3x_1 + 4x_2 + 6x_3 &= 0 \end{aligned} \tag{3}$$

Durchführung

Aufgabe D8:

Lösen Sie das lineare Gleichungssystem der obigen Vorbereitungsaufgabe mit Hilfe von MATLAB.

Aufgabe D9:

Betrachtet wird das lineare Gleichungssystem ($a \in \mathbb{R}$)

$$\begin{aligned} 2x_1 + + x_3 &= -1 \\ \tfrac{1}{4}x_1 + \tfrac{15}{4}x_2 + x_3 &= -3 \\ (3+6a^2)x_1 + (1-16a^2)x_2 + x_3 &= 10a - 3 \end{aligned} \tag{4}$$

dessen Lösung(en) nun grafisch veranschaulicht werden soll(en).

a) Betrachten Sie zunächst nur die erste Gleichung des Gleichungssystems. Lösen Sie diese nach x_3 auf und stellen Sie x_3 als 3D–Grafik über x_1 und x_2 dar, z.B. mittels

```
>> [x1, x2] = meshgrid( -2:.1:2, -2:.1:2 );
>> x3_Gl1 = ...
>> surf( x1, x2, x3_Gl1, 1*ones(size(x3_Gl1)) ); shading flat;
>> hold on; axis(2*[-1 1 -1 1 -1 1]); caxis([1 3]);
```

Welches geometrisches Gebilde ist zu sehen?

b) Betrachten Sie nun die zweite Gleichung des Gleichungssystems. Lösen Sie diese ebenfalls nach x_3 auf und stellen Sie x_3 als zusätzlichen Plot in Ihrer 3D–Grafik dar.

```
>> x3_Gl2 = ...
>> surf( x1, x2, x3_Gl2, 2*ones(size(x3_Gl2)) ); shading flat;
```

Wo liegen die Punkte $[x_1, x_2, x_3]^\mathsf{T}$, welche beide Gleichungen gleichzeitig erfüllen?

c) Betrachten Sie nun die dritte Gleichung des Gleichungssystems. Lösen Sie diese ebenfalls nach x_3 auf und stellen Sie x_3 für $a = 0$ als weiteren Plot in Ihrer 3D–Grafik dar. Wo liegt die Lösung des Gleichungssystems?

d) Ändern Sie nun a in kleinen Schritten von 0 nach 1 und stellen Sie jeweils die 3D–Grafik dar. Was beobachten Sie? Wo liegt/liegen jeweils die Lösung(en)?

e) Ändern Sie nun a in kleinen Schritten von 0 nach -1 und stellen Sie jeweils die 3D–Grafik dar. Was beobachten Sie? Wo liegt/liegen jeweils die Lösung(en)?

9 Komplexe Zahlen

MATLAB ist ideal für Berechnungen mit komplexen Zahlen geeignet. Alle arithmetischen Operationen und alle Standardfunktionen sind auch für komplexe Zahlen definiert. Zur Eingabe von komplexen Zahlen gibt es in MATLAB mehrere Möglichkeiten, die imaginäre Einheit einzugeben, nämlich `i`, `j`, `1i`, und `1j`. Prinzipiell sind alle vier Darstellungen äquivalent und entprechen der Wurzel aus -1. Allerdings sind die beiden Darstellungen `1i`, und `1j` der komplexen Einheit numerisch günstiger und führen dazu, dass die entsprechenden Ausdrücke schneller ausgewertet werden können. Deshalb sollte man zumindest für umfangreiche Berechnungen mit komplexen Zahlen die Darstellungen `1i` oder `1j` verwenden.

Wichtige Funktionen für den Umgang mit komplexen Zahlen sind: **`abs`**, **`angle`**, **`real`**, **`imag`**, und **`conj`**. Die folgenden Beispiele zeigen, wie in MATLAB mit komplexen Zahlen gerechnet werden kann.

```
>> x = 2+3j
x =
   2.0000 + 3.0000i
>> y = 3+5*1j
y =
   3.0000 + 5.0000i
>> z = x*y
z =
  -9.0000 +19.0000i
>> xr = real(x)
xr =
     2
>> xi = imag(x)
xi =
     3
>> xc = conj(x)
xc =
   2.0000 - 3.0000i
>> x_betrag = abs(x)
x_betrag =
    3.6056
>> x_phase = angle(x)
x_phase =
    0.9828
>> x-x_betrag*exp(1j*x_phase)
ans =
   0.0000e+00 + 4.4409e-16i
```

MATLAB kann auch als Hilfsmittel für die komplexe Wechselstromrechnung eingesetzt werden. Besonders interessant ist die Möglichkeit, komplexwertige Funktionen zu veranschaulichen. Das folgende Beispiel zeigt, wie die komplexe Impedanz Z einer Parallelschaltung eines ohmschen Widerstandes $R = 100\,\Omega$ und einer Kapazität $C = 10\,\mu F$ für eine Frequenz von $f = 1\,kHz$ berechnet werden kann. Anschließend wird noch die Ortskurve der Impedanz als dreidimensionale Grafik mit dem Befehl **`plot3`** dargestellt.

```
%Parallelschaltung von R und C
%Berechnung der komplexen Impedanz Z fuer f = 1kHz
f=1000;   %Frequenz
w=2*pi*f; %Kreisfrequenz
R=100;    %Widerstandswert
C=10e-6;  %Kapazitaet
```

```
Z=1/(1/R+j*w*C) %Berechnung der komplexen Impedanz

%Erstellung einer Ortskurve fuer Z
f = linspace(0,10000,1000); %Frequenz: 1000 Werte zwischen 0 und 10kHz
w = 2*pi*f;  %Kreisfrequenz: Vektor mit 1000 Werten
Z=1./(1/R+j*w*C)  %./ ist hier notwendig, da w jetzt ein Vektor ist
ZR = real(Z);  %Realteil von Z
ZI = imag(Z);  %Imaginaerteil von Z
plot3(w,ZR,ZI,'LineWidth',2)  %Grafische Darstellung der Ortskurve
xlabel('\omega in 1/s')
ylabel('Realteil von Z in \Omega');
zlabel('Imaginaerteil von Z in \Omega');
grid on
print ('-dpng', '../Bilder/ParallelschaltungRC.png'); %png-File erz.
```

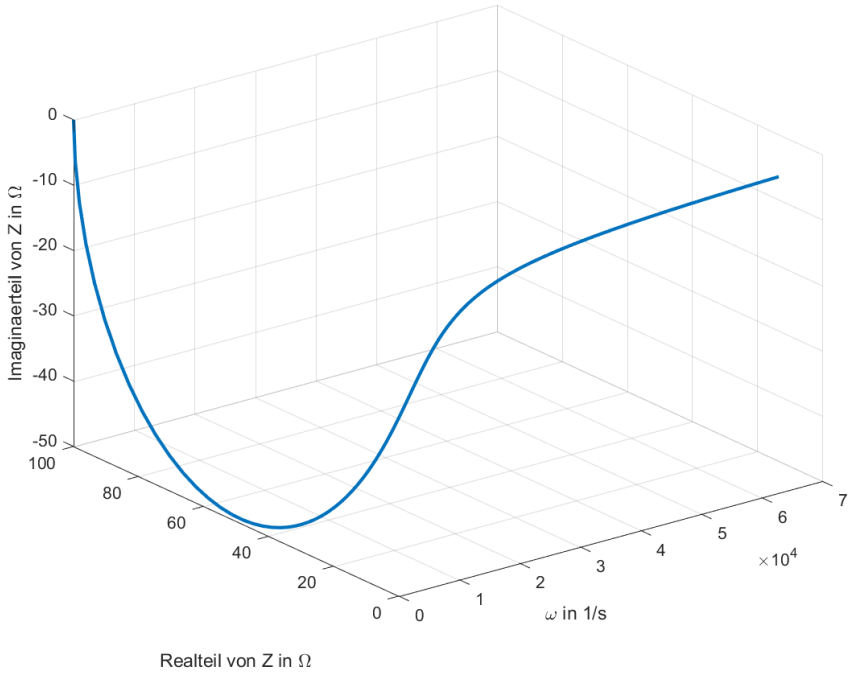

Abbildung 5: Ortskurve von Z für die Parallelschaltung von R und C .

Durchführung

Aufgabe D10:

Wir betrachten die Reihenschaltung eines Widerstandes $R = 100\,\Omega$ mit einer Kapazität $C = 400\,\mathrm{nF}$ und einer Induktivität $L = 0,1\,\mathrm{H}$. Zeichnen Sie mit dem MATLAB-Befehl **plot3** eine dreidimensionale Darstellung der Ortskurve für die komplexe Impedanz Z des Reihenschwingkreises.

Aufgabe D11:

In dieser Aufgabe wollen wir die Basisfunktionen der Fourier- und Laplace-Transformation veranschaulichen. Die Basisfunktionen der Laplace-Transformation ist eine allgemeine komplexe Exponentialfunktion $b(t) = e^{st} = e^{(\sigma+j\omega)t}$. Für $\sigma = 0$ erhalten wir die Basisfunktion der Fourier-Transformation. Zeichnen Sie mit Hilfe des Befehls **plot3** den Graphen von $b(t)$. Erzeugen Sie sich dazu zunächst eine Zeitachse, die von 0 bis 10 reicht. Variieren Sie die Werte für σ und ω und beobachten Sie, wie sich die

Basisfunktion verändert.
Hinweis: Sie sollten für beide Parameter sowohl positive als auch negative Werte und den Wert 0 wählen.

Weiterführende Literatur

[1] Robert Fischer. *Skript zum Praktikum Simulationstools*. FAU Erlangen-Nürnberg, 2009.

[2] Martin Werner. *Digitale Signalverarbeitung mit MATLAB*. Vieweg und Teubner Verlag, 2012.

[3] Ottmar Beucher. *MATLAB und Simulink – Eine kursorientierte Einführung für Studierende der Natur- und Ingenieurwissenschaften*. MITP, 2013.

[4] Anne Angermann, Michael Beuschel, Martin Rau, and Ulrich Wohlfarth. *MATLAB – Simulink – Stateflow*. Oldenbourg Verlag, 2014.

[5] Rüdiger Kutzner and Sönke Schoof. *MATLAB/Simuling*. RRZN Handbuch, 2014.

[6] Wolfgang Schweizer. *MATLAB kompakt*. Oldenbourg Verlag, 2016.