Einführung in Matlab

Autoren: Prof. Dr. J. Bitzer
Prof. Dr. M. Hansen
Ulrik Kowalk (M.Sc.)

Stand: 13. September 2016

Institut Technik und Gesundheit für Menschen (TGM)
Jade Hochschule Oldenburg/Wilhelmshaven/Elsfleth
Fachbereich Bauwesen und Geoinformation (B+G)
Ofener Str. 16
D-26121 Oldenburg

JADEHOCHSCHULE
Wilhelmshaven **Oldenburg** Elsfleth

TGM Technik und Gesundheit für Menschen

Inhaltsverzeichnis

1 Was ist Matlab und wozu wird Matlab genutzt?

Matlab steht für *Matrix Laboratory* und ist ein Programmpaket von Mathworks. In der heutigen Form kann man Matlab als Programmiersprache der 3.-4. Generation verstehen. Die Abarbeitung der Programme erfolgt im Normalfall Zeile für Zeile, somit ist das Hauptprogramm ein Interpreter für die verwendeten Programme.

Als Studierende haben Sie die Möglichkeit eine Matlab Student Edition zu erwerben, die im Vergleich zum Originalpreis von Matlab sehr günstig ist. Dies kann interessant für Sie sein, da Sie im weiteren Studium immer wieder mit Matlab arbeiten werden und sich Matlab in vielen Branchen als Industriestandard bei der Prototypentwicklung durchgesetzt hat. Wer die Kosten sparen will, kann auch sog. Matlab-Clones einsetzen. Der bekannteste Clone ist Octave (www.octave.org). Die grundsätzliche Handhabung ist ähnlich, aber Octave unterstützt nicht *sämtliche* Matlab-Features und ist nicht ganz so komfortabel zu bedienen. Der größte Nachteil ist die andere Plot-Syntax von Octave im Vergleich zu Matlab.

Zur Erweiterung der Standardfunktionalität von Matlab kann man sogenannte Toolboxes erwerben, die es ermöglichen, Matlab in vielen Spezialgebieten einzusetzen. So gibt es unter anderem eine Financial Toolbox für Aktienkursanalysen. Für die Nachrichtentechnik und die Signalverarbeitung ist insbesondere die Signal Processing Toolbox wichtig. Ebenso interessant für H+A und AT ist die Statistik-Toolbox.

Um Matlab etwas besser gegen andere Programmiersprachen abzugrenzen, sind in der folgenden Tabelle einige besondere Eigenschaften festgehalten:

Eigenschaft	**Matlab**	**C / C++**
Hauptanwendung	Prototypentwicklung, Forschung	Applikationsentwicklung
Programmausführung	Interpreter	Compiler
Speichermanagment	automatisch	komplex
Unterstützung Vektoren / Matrizen	interne Repräsentation	Nur durch Zusatzbibliotheken
Rechenzeit	z.T. langsam	optimiert, sehr schnell
Kosten	für Firmen sehr hoch	unter 1000 Euro
Freie Software Version	Octave	GNU

Die Stärke und gleichzeitig das Hauptproblem von Matlab ist die sehr einfache Bedienung und eine gewisse Fehlertoleranz: Sie bekommen, außer bei sehr groben Fehlern, meistens ein Ergebnis berechnet. Dies führt leider oft zu einem **Programmieren ohne Nachdenken** darüber, ob dies das Ergebnis ist, das Sie tatsächlich berechnen wollten. Dies sollten Sie vermeiden! Auch in Matlab ist es unumgänglich, vor der Implementierung der Problemlösung nachzudenken und ein Lösungskonzept (Struktogramm, Programmablaufplan, usw.) zu entwerfen.

2 Erste Schritte

2.1 Die Programmoberfläche

Nach dem Start sollten Sie eine Programmoberfläche sehen, welche der in Abbildung 1 ähnelt. Diese ist standardmäßig in 4 Teile aufgeteilt (sonst evtl. unter dem Menüpunkt `Home` → `Layout` fehlende Teile auswählen).

1. **Explorer**: Zeigt den aktuellen Pfad und die darin befindlichen Files
2. **Kommandozeile**: Dient der Eingabe der Matlab-Befehle sowie der Darstellung des Outputs
3. **Workspace**: Zeigt alle gerade genutzten Variablen inklusive ihres Typs und ihrer Dimensionen
4. **Command History**: Liste der zuletzt benutzten Befehle

Es ist möglich, Matlab allein mittels der Kommandozeile als einziges Fenster zu bedienen. Es empfiehlt sich, diese Art der Bedienung zügig zu lernen. Die weiteren Fenster dienen je nach Geschmack dem Komfort und der Übersicht. Die Erfahrung zeigt, dass die Bedienung mittels Tastatur allein, also ohne Maus, für fast alle Nutzer schneller und effizienter ist als häufiges Mausklicken in verschiedenen Unterfenstern.

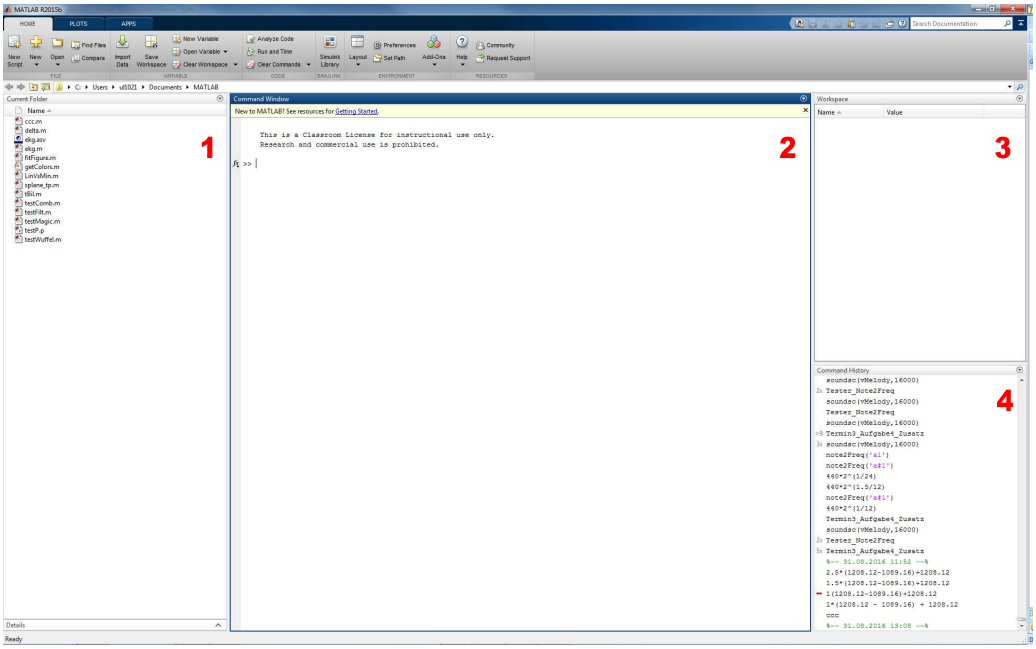

Abbildung 1: Typische Startoberfläche von Matlab (Version R2015b)

2.2 Der Editor

Einen sehr wichtigen Programmteil stellt der Editor dar, in dem die einzelnen Programme erarbeitet und geschrieben werden können. Da die Programme, sogenannte Skripts und Funktionen, in Matlab reine ASCII-Files mit der file-extension `.m` sind, lässt sich jeder beliebige Editor verwenden. Matlab verfügt auch über einen eingebauten Editor. Er wird mit dem Befehl `edit IhrProgrammName.m` aufgerufen (und das File `IhrProgrammName.m` automatisch erstellt). Das Editieren eines Programms in Matlab erfolgt in gleicher Weise, wie man es von jedem anderen Editor fürs Programmieren gewohnt ist. Beachten Sie, dass der aktuelle Pfad in Matlab und im Editor unterschiedlich sein können. Deshalb empfiehlt sich, zunächst in Matlab einen Pfad einzustellen z.B. `d:\User\IhrName\Versuch1` und erst dann den Editor mit `edit` ohne Argument zu starten. Um Matlab erst einmal kennen zu lernen, kann man ohne den Editor Kommandos auf der Kommandozeile eingeben. Abbildung 2 zeigt den Matlab-Editor.

Die Funktionen des Editors sind in drei Kartei-Reiter unterteilt: *Editor*, *Publish* und *View*. Die wichtigsten Menüpunkte finden Sie unter Editor. Hier sehen Sie alles, was direkt mit Ihrem Code zu tun hat, allem voran der *Run*-Button. Ein Druck auf diesen startet die Abarbeitung Ihres Skripts. Desweiteren finden Sie hier Vergleichsfunktionen, Kommentarfunktionen, Speichern des Programms und so weiter. Der Reiter *Publish* beinhaltet Hilfen zur Veröffentlichung und

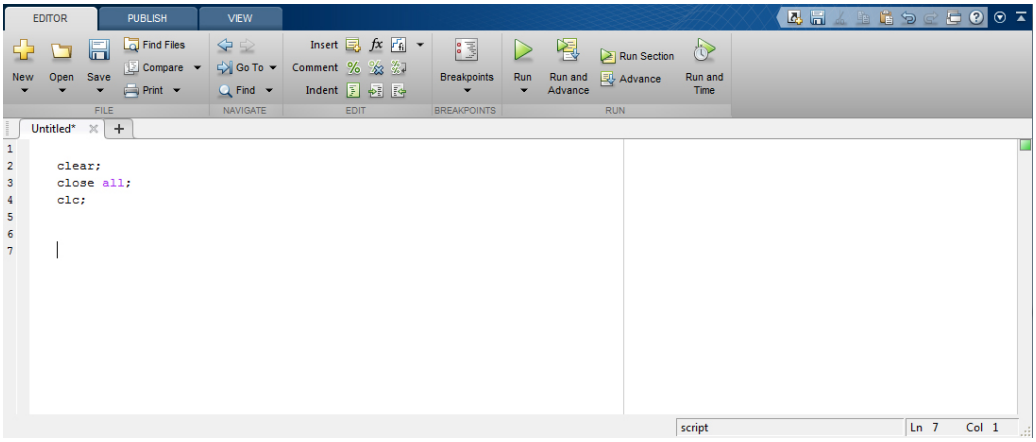

Abbildung 2: Matlab-Editor (Version R2015b)

Aufbereitung Ihres Codes. Dies ist an dieser Stelle aber zunächst sekundär. Unter *View* können Sie das Editorfenster konfigurieren, z.B. in zwei nebeneinander liegende Fenster aufteilen um auf diese Weise zwei Skripte oder Funktionen miteinander zu vergleichen.

2.3 Shortcuts

Für ein effizienteres Arbeiten mit Matlab ist es empfehlenswert, sich einige Tastaturkürzel anzueignen. Neben den gängigen Shortcuts aus der Textverarbeitung sind folgende Kombinationen wichtig:

- Strg + R: Kommentiert die ausgewählten Zeilen aus
- Strg + T: Kommentiert die ausgewählten Zeilen ein
- F5: Startet die Abarbeitung des Skripts, analog zum *Run*-Button
- Strg + I: Rückt selektierte Zeilen korrekt ein (am besten den gesamten Code selektieren)
- Strg + D: Bei Auswahl einer Funktion öffnet Matlab diese automatisch

2.4 Speziallösung im TGM

Zur Vereinfachung der Kommentierung und um immer ein Grundgerüst für ein neues Skript oder eine neue Funktion zu haben, bietet das TGM zwei Hilfsfunktionen an (siehe Moodle), mit denen sich sehr einfach neue Funktionen oder Skripte erstellen lassen. Die beiden Funktionen lauten

```
newScript('ScriptName','AutorenName')
newFunction('FunctionName','AutorenName')
```

Sie sollten diese beiden Funktionen mit in den Suchpfad von Matlab übernehmen (siehe help PATH, oder Path Editor aufrufen) oder die Funktionen in ihr Arbeitsverzechnis (z.B. `D:\Users\Student\matlab`) kopieren. Nach dem Aufruf wird automatisch der Editor geöffnet und Sie können mit der Programmierung und Kommentierung beginnen (vorher natürlich nachdenken, was programmiert werden soll und wie ein guter Name aussieht.)

2.5 Hilfe finden

Um schnell einen Überblick über die Funktionsweise oder Parameter eines Skripts oder einer Funktion zu erhalten, wird der Begriff `help <Name>` verwendet. Gerade für neue Matlab-UserInnen empfiehlt ist es empfehlenswert, auf diese Weise einen Teil des Funktionsumfangs von Matlab besser kennen zu lernen.

3 Vektoren und Matrizen

Matlab ist eine Programmiersprache, die vektoriell bzw. Matrix-orientiert arbeitet. Anders als in C/C++ sind alle Variablen in Matlab automatisch Matrizen. Vektoren haben dabei die Dimensionen $N \times 1$ (Spalten-Vektor) oder $1 \times N$ (Zeilen-Vektor). Skalare Größen sind Matrizen der Dimension 1×1. Einige Beispiele, die sie direkt eingeben sollten:

`x = 2`	weist `x` den Wert 2 zu.
`a = [3 4 5]`	weist `a` einen Zeilenvektor (row vector) der Dimension 1×3 zu.
`b = [6;7;8]`	weist `b` einen Spaltenvektor (column vector) der Dimension 3×1 zu.
`c = [1 2 3; 4 5 6]`	weist `c` eine Matrix der Dimension 2×3 zu.

Sie werden feststellen, dass Ihre Eingaben immer wiederholt werden. Die Ausgabe des Resultats auf dem Bildschirm kann durch das Semikolon `;` am Ende des Kommandos verhindert werden. Grundsätzlich kommt uns die vektorielle Arbeitsweise von Matlab in der Signalverarbeitung sehr entgegen, da Signale oft als Vektoren darstellbar sind. Auf der anderen Seite bedeutet dies aber auch, dass Sie sich bei jedem Arbeitsschritt über die Dimension ihrer Variablen im klaren sein müssen. Zusätzlich sollten Ihnen die grundlegenden Rechenregeln für Matrizen bekannt sein. Hier noch einmal eine kleine Wiederholung und die dazugehörige Matlab-Schreibweise:

3.1 Rechenregeln für Matrizen und ihre Matlab-Anwendung

Addition: Die Addition erfolgt elementweise, d.h. die Matrixdimensionen müssen exakt übereinstimmen. Für die oben eingegebenen Beispielmatrizen gilt: `d=a+a` ist erlaubt, dagegen führt `e=a+c` zu einer Fehlermeldung. Es ist möglich, einen Skalar und eine Matrix zu addieren. Beispielsweise ergibt `f=x+a` den neuen Vektor `[5 6 7]`

Subtraktion: siehe Addition.

Elementweise Multiplikation: Bei der Multiplikation gibt es zwei verschiedene Modi. Bei der elementweisen Multiplikation müssen die Dimensionen der beiden Multiplikanden exakt gleich sein. Die elementweise Multiplikation wird in Matlab durch einen vorangestellten Punkt angezeigt. Beispiel: `h=a.*a` ist erlaubt und ergibt den Vektor `[9 16 25]`. Dagegen ist `a.*b` nicht definiert.

Ebenso wie bei der Addition ist es auch möglich, einen Skalar mit einer Matrix zu multiplizieren. Das Ergebnis ist elementweise wie bei der Skalarmultiplikation von Vektoren definiert. In diesem Fall kann gleichwertig der Multiplikationspunkt `*` allein oder vorangestelltem Punkt `.*` für „elementweise“ verwendet werden. Der Ausdruck `x*a` ergibt dasselbe wie `x.*a`, nämlich den Vektor `[6 8 10]`

Matrizenmultiplikation: Die Matrizenmultiplikation entspricht der aus der linearen Algebra bekannten Multiplikation zweier Matrizen. Hierbei gilt, dass die beiden Matrizen in ihren Dimensionen zueinander passen müssen. Die Anzahl der Spalten der linken Matrix müssen gleich der Anzahl der Zeilen der rechten Matrix in der Gleichung

sein. Beispiel: `k=a*b` ergibt den skalaren Wert 86. `l=c*b` ist ebenfalls erlaubt und ergibt den Spaltenvektor `[44; 107]` der Dimension 2×1. In umgedrehter Reihenfolge der Multiplikanden ist das Produkt `m=b*c` nicht definiert. Auch `n=c*a` führt zu einer Fehlermeldung.

Elementweise Division: Die Division ist im allgemeinen nicht für Matrizen definiert. Bei der elementweisen Division gilt das gleiche wie für die Multiplikation.

Inverse Matrix (Division): Einer der wichtigen Punkte, warum Matlab so erfolgreich ist, ist die Implementierung des Matrix-Inversions-Problems, die auch für schlecht konditionierte Matrizen noch gute Ergebnisse liefert. (Dieses Skript ersetzt nicht Teile der Mathematikvorlesung. Falls Ihnen nicht bekannt ist, was singuläre Matrizen sind, so gehen Sie zunächst einfach über den Begriff „schlecht konditioniert" hinweg.) Somit ist das Lösen von linearen Gleichungssystem mit Matlab sehr einfach und elegant. Als simples Beispiel seien folgende Gleichungen gegeben:

$$2z_1 + 3z_2 + 4z_3 = 11 \tag{1}$$

$$z_1 - z_2 + z_3 = 9 \tag{2}$$

$$-z_1 + 2z_2 + 3z_3 = 4 \tag{3}$$

Bestimmen Sie z_1, z_2 und z_3. Normalerweise rechnet man an einer solchen Aufgabe 5 bis 10 Minuten mit Hilfe des Gleichsetzungs- und Einsetzverfahrens. In Matlab wird das Problem wie folgt gelöst. Zunächst ist die Umformulierung in Matrixschreibweise notwendig:

$$\mathbf{Az} = \mathbf{d} \tag{4}$$

mit

$$\mathbf{A} = \begin{pmatrix} 2 & 3 & 4 \\ 1 & -1 & 1 \\ -1 & 2 & 3 \end{pmatrix}, \qquad \mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}, \qquad \mathbf{d} = \begin{pmatrix} 11 \\ 9 \\ 4 \end{pmatrix} \tag{5}$$

Angenommen die Inverse $\mathbf{A}^{-1}$ zu $\mathbf{A}$ existiert. Dann folgt aus Gl. (4) durch Multiplikation *von links* mit $\mathbf{A}^{-1}$:

$$\mathbf{A}^{-1}\mathbf{Az} = \mathbf{A}^{-1}\mathbf{d} \tag{6}$$

$$\Rightarrow \mathbf{z} = \mathbf{A}^{-1}\mathbf{d}, \tag{7}$$

da $\mathbf{A}^{-1}\mathbf{A}$ die Einheitsmatrix ist.

Die Berechnung der Lösung **z** aus dieser Aufgabe in Matlab ergibt sich als `z = A \ d`. Der Backslash symbolisiert dabei das Lösen des linearen Gleichungssystems. Er soll symbolisch andeuten, dass **A** *unter* dem Bruchstrich steht, also dass sozusagen **d** durch **A** geteilt wird (!), also mit $\mathbf{A}^{-1}$ multipliziert wird. Weil **A** links von **d** steht, wird also **d** von links mit $\mathbf{A}^{-1}$ multipliziert wird. Es ist auch möglich die Lösung wie oben direkt zu nutzen: `z=A^-1 *d`. `z=inv(A)*d`.

Im allgemeinen Fall sollte aber beim Lösen von Gleichungssystemen auf die explizite Inversion der Matrix verzichtet werden, da andere Lösungen oft numerisch effizienter sind. Dies gilt insbesondere für große Gleichungssysteme.

Elementweise Potenzierung siehe Multiplikation, die Matlabnotation ist `p=a.^2`.

Transponierung: Zur Umwandlung von Spalten- in Zeilenvektoren oder zur Transponierung von Matrizen wird das Hochkomma `'` verwendet. Bei komplexen Zahlen wird gleichzeitig eine Konjugation durchgeführt. Möchten Sie nur Transponieren wird dies durch einen vorangestellten Punkt deutlich gemacht, als `.'`. Für rein reelle Matrizen haben also ' und .' dieselbe Wirkung. Beispiel: `b'` ergibt den Zeilenvektor `[6 7 8]`.

3.2 Index-Zugriff auf Matrizen

Häufig möchte man auf bestimmte Elemente einer Matrix zugreifen. Dies gelingt durch den Index-Operator `(·,·)`, wobei zuerst die Zeile, dann die Spalte angegeben werden. So ergibt sich beispielsweise für `aa = A(1,3)` eine 4. Es ist auch möglich, Teile aus Matrizen oder Vektoren zu extrahieren. Dies geschieht durch Index-Vektoren im Index-Operator. Zusammenhängende Index-Vektoren erhält man durch den `:` („bis“). So entspricht `3:6 = [3 4 5 6]`. Als Beispiel extrahieren wir die oberen rechten 4 Elemente der Matrix **A**. `B = A(1:2,2:3)`. Als Ergebnis ergibt sich `B = [ 3 4 ; -1 1]`. Ganze Spalten oder Zeilen können durch Eingabe des Doppelpunktes ohne weitere Angaben extrahiert werden. Manchmal kennt man die Größe eines Vektors oder einer Matrix nicht, aber man weiß, dass alle Endelemente angesprochen werden sollen. In diesem Fall kann das Schlüsselwort `end` verwendet werden, das immer auf das letzte Element zeigt. Für das zuvor genutzte Beispiel hätte also auch `B = A(1:2,2:end)` funktioniert.

3.3 Spezielle Matrizen

Der Aufbau größerer Matrizen kann in Handarbeit sehr mühevoll sein, deshalb sind einige besonders oft verwendete Matrizen bereits vordefiniert.

`I = eye(N,M)` erzeugt eine eine $N \times M$ große Einheitsmatrix:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

.

`B = zeros(N,M)` erzeugt eine $N \times M$ Matrix mit lauter Nullen als Elementen.

`C = ones(N,M)` erzeugt eine $N \times M$ Matrix mit lauter Einsen.

`X = randn(N,M)` erzeugt eine $N \times M$ Matrix, deren Elemente aus lauter Zufallszahlen bestehen, denen eine *Gauß'sche Standard-Normalverteilung* zugrunde liegt, d.h. die Varianz beträgt 1 und der Mittelwert 0.

`Y = rand(N,M)` erzeugt ebenfalls eine $N \times M$ Matrix, deren Elemente aus Zufallszahlen bestehen. Jedoch entstammen diese einer *Gleichverteilung* mit Werten aus dem offenen Intervall $0 < y_{n,m} < 1$.

⚠ **Achten Sie auf den gewaltigen Unterschied zwischen `rand` und `randn`! Das `n` am Ende von `randn` steht für Normalverteilung.**

3.4 Einfache Operationen mit Matrizen

Es gibt eine Reihe von Operationen, mit denen sich ausgehend von einfachen Matrizen kompliziertere erzeugen lassen. Die folgenden Beispiele gehen von den beiden Vektoren `x = ones(1,6)` und `y = (3:8)` aus:

`diff`: Gliedweise Differenzenbildung benachbarter Elemente. Dies ist die diskrete Entsprechung zum Differenzieren von Funktionen. Achtung, der Ergebnisvektor ist um 1 Element kürzer als der Eingabevektor.

Beispiel: `diff(y)` ergibt `[1 1 1 1 1]`

`cumsum`: Kumulierte Summenbildung. Dies ist die diskrete Entsprechung zum Integrieren von Funktionen. Achtung, hier hat der Ergebnisvektor dieselbe Länge wie der Eingabevektor.

Beispiel: `cumsum(x)` ergibt `[1 2 3 4 5 6]` und `cumsum(y)` ergibt `[3 7 12 18 25 33]`

`sign`: Vorzeichenfunktion.

Beispiel: `a = (-2:3); b = sign(a)` ergibt `[-1 -1 0 1 1 1]`

`abs`: Absolutwert.

Beispiel: `a = (-2:3); b = abs(a)` ergibt `[2 1 0 1 2 3]`

`y > 5`: Elementweiser logischer Vergleich.

Beispiel: `y > 5` liefert *nicht* die Zahlen `[6 7 8]`, die in diesem Beispiel größer sind als 5, sondern elementweise eine 1 für logisch-wahr bzw. eine 0 für logisch-falsch, hier also: `[0 0 0 1 1 1]`. Entsprechend ergibt `x > 10` einen reinen Nullen-Vektor: `[0 0 0 0 0 0]`

`find`: Auffinden der *Indizes* derjenigen Matrixelemente, die eine logische Bedingung erfüllen.

Beispiel: `find(y>5)` ergibt `[4 5 6]`, denn für diese Indizes gilt, dass $y > 5$. Deshalb ergibt der Ausdruck `find(x>10)` eine leere Matrix als Ergebnis, denn kein Element erfüllt die Bedingung.

4 Komplexe Zahlen

Für die Signalverarbeitung und die Nachrichtentechnik ist die Verarbeitung komplexer Zahlen von elementarer Bedeutung. Der komplexe Anteil einer Zahl wird in Matlab durch ein nachgestelltes `i` kenntlich gemacht. In der Nachrichtentechnik üblicher ist die Verwendung des *j*, welches ebenfalls in Matlab verwendet werden kann. Dies ist auch der Grund, warum in den vorherigen Beispielen das `i` und `j` nicht als Variablennamen genutzt wurden. Sie sollten darauf achten, dass Sie `i` oder `j` *nicht* überladen, also z.B. `i = 1` schreiben. Um `i` erneut den richtigen Wert zuzuweisen hilft z.B. `i = sqrt(-1)`, was die Quadratwurzel aus -1 berechnet und in Matlab den gewünschten Wert der imaginären Einheit liefert.

Alle bisherigen Betrachtungen gelten auch für komplexe Zahlen. Somit kann ein einfacher komplexer Vektor z. B. durch `z = [1+4i 2-3i 4+5i]` definiert werden. Wichtig ist dabei, dass kein Leerzeichen zwischen dem + und − und den einzelnen Zahlen vorkommt, da Matlab dies sonst als zwei Zahlen – eine rein reellwertige und eine rein imaginärwertige – interpretiert. Eine weitere Darstellung komplexer Zahlen, die sehr häufig in der Nachrichtentechnik verwendet wird, arbeitet mit dem Radius und der Phase. Abbildung 3 zeigt einen Wertepunkt `z` in der komplexen Ebene (Darstellung von Imaginärteil über Realteil). Dieser Punkt kann ebenso über dessen Radius r und den Winkel φ angegeben werden.

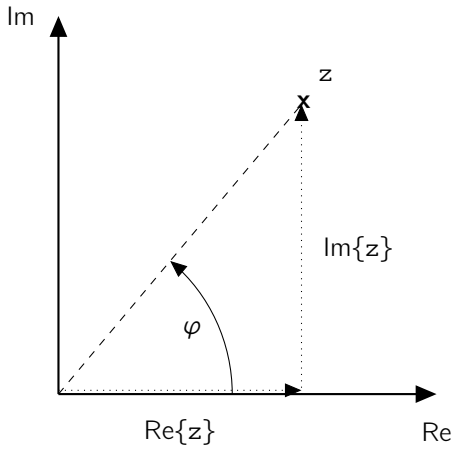

Abbildung 3: Alternative Darstellungen einer komplexen Zahl `z` in der komplexen Ebene

Zur Umrechnung beider Darstellungsarten sind folgende Formeln zu benutzen:

$$\mathrm{Re}\{z\} = r \cdot \cos(\varphi) \tag{8}$$

$$\mathrm{Im}\{z\} = r \cdot \sin(\varphi) \tag{9}$$

$$r = \sqrt{\mathrm{Re}\{z\}^2 + \mathrm{Im}\{z\}^2} \tag{10}$$

$$\varphi = \begin{cases} \arctan\left(\frac{\mathrm{Im}\{z\}}{\mathrm{Re}\{z\}}\right) & \text{bei} \quad \mathrm{Re}\{z\} \geq 0 \\ \arctan\left(\frac{\mathrm{Im}\{z\}}{\mathrm{Re}\{z\}}\right) + \pi & \text{sonst} \end{cases} \tag{11}$$

Für die Darstellung von z lässt sich auch schreiben:

$$z = \mathrm{Re}\{z\} + j \cdot \mathrm{Im}\{z\} \tag{12}$$

$$= r \cdot (\cos(\varphi) + j \cdot \sin(\varphi)) \tag{13}$$

Durch Umwandlung mit der Eulerschen Formel erhalten wir die sehr häufig verwendete Form:

$$z = r \cdot e^{j\varphi}. \tag{14}$$

Die Matlab-Funktionen für die Umrechnungen lauten `real(·)`, `imag(·)`, `abs(·)`, `angle(·)`.

5 Matlab Programme: Skripts

Die Eingabe jedes einzelnen Befehls in der Kommandozeile ist eher mühsam, insbesondere wenn man eine längere Folge von Kommandos mehr als einmal durchführen möchte oder wenn zwischendurch Fehler auftraten, da dann häufig von vorne begonnen werden muss. Eine elegante Methode zur Lösung dieses Problems stellt die Verwendung eines sog. „Skripts“ oder „m-files dar“. Diese Files bestehen aus normalem ASCII Text. In ihnen können Matlab-Befehlsfolgen in einem Editor vorab geschrieben und abgespeichert werden. Die Datei-Endung lautet `*.m`, daher auch die Bezeichnung „m-file“. Der Aufruf dieser Befehlsfolge erfolgt über den Namen des gespeicherten Files. Ein sehr einfaches Programm, welches beispielsweise eine komplexe Zahl in die Radius-Winkel Darstellung überführt, würde folgendermaßen aussehen:

```
% Sehr Kurzes Beispielskript
%      Das Prozentzeichen (%) ist das Kommentarzeichen in Matlab
n_Z = 3+4i;           % Zuweisung einer komplexen Zahl
n_Betrag = abs(n_Z)   % Berechnung des Betrags. Ohne Semikolon -> Ausgabe auf dem Monitor
n_Winkel = angle(n_Z) % Berechnung der Phase, ebenfalls ohne ;
n_Real = real(n_Z);   % Berechnung des Realteils
n_Imag = imag(n_Z);   % Berechnung des Imaginaerteils
n_Quadrat = n_Z*n_Z;  % Berechnung des Quadrats

% End of file beispiel1
```

Obiger Text würde in ein File namens `beispiel1.m` gespeichert. Auf der Kommandozeile ruft man das neu entstandene Kommando `beispiel1` auf. Wünscht man, die Berechnungen ebenfalls für andere Zahlen durchzuführen, so ist es nur nötig, jeweils die Zuweisung von `z` zu ändern, das File erneut zu speichern und ein weiteres Mal aufzurufen.

6 Ein-und Ausgaberoutinen

Ein sehr wichtiger Teil beim Programmieren und Lösen von Aufgaben liegt in der Kommunikation mit der Außenwelt und anderen Programmen. Als Ausgabegerät haben wir zur Zeit nur den Monitor verwendet. Als Eingabe nur die Tastatur. Unser Programm aus dem vorherigen Abschnitt musste für eine Veränderung der komplexen Zahl immer wieder editiert werden. Eine alternative Möglichkeit wäre eine direkte Eingabe. Solche einfachen Eingaben sind durch den `input` Befehl möglich. Zum Beispiel bewirkt `n_Z = input('Geben Sie eine Zahl ein')`, dass Matlab die Abarbeitung der Befehle anhält, bis der Benutzer eine Eingabe gemacht hat. Diese wird der Variablen `z` zugewiesen und danach werden die folgenden Befehle abgearbeitet.

6.1 File-I/O

Für den Schwerpunkt Audiologie, sind insbesondere Audio-Dateien, die häufig im Microsoft *.wav Format vorliegen, die wichtigsten Ein- und Ausgabedaten. Zum lesen gibt es den Befehl `v_Signal = audioread(s_Filename)`, wobei s_Filename ein String ist, welcher den Namen der zu öffnenden Datei enthält. Dieser Befehl kann optional auch erweitert werden zu `[v_Signal, n_Fs] = audioread(s_Filename)`. Dies hat den Vorteil, dass man über den Parameter der Samplingfrequenz der Wave-Datei informiert ist (siehe `help audioread`). Nach dem Einlesen stehen die Abtastwerte in der Variable mit dem Namen `v_Signal` zur Verfügung. Im Falle eines Stereosignals besteht `v_Signal` aus zwei Spalten, bei einem Monosignal wird nur ein Spaltenvektor gelesen. Das Speichern eines Vektoren bzw. Matrix mit Abtastwerten als Wave-File erfolgt mit dem Befehl `audiowrite(s_Filename, v_Signal, n_Fs)`. Zusätzlich zur Samplingfrequenz können weitere Parameter festgelegt und übergeben werden, Einzelheiten dazu finden Sie unter `help audiowrite`. Möchten Sie nur Informationen über die Sounddatei erhalten ohne dafür das gesamte File vorher einzulesen, bietet sich der Befehl `st_Info = audioinfo(s_Filename)` an. Dabei wird ein Struct (hier `st_Info`) erzeugt, welches sämtliche abrufbaren Informationen über die Datei enthält.

Zur Speicherung von Daten, die keine spezielle Verbindung zu Audio haben, stehen die Funktionen `load` und `save` zur Verfügung, die ein einfaches Speichern einiger oder aller zur Zeit vorhandenen und definierten Variablen in einem Binärformat ermöglichen. Da in Matlab auch objektorientierte Programmierung möglich ist, ist der `import`-Befehl von großer Bedeutung. Mit ihm werden Klassen importiert und so der Funktionsumfang des eigenen Programms erweitert. Weitere Hilfe hierzu finden Sie unter `help import`.

6.2 Diagramme

6.2.1 Erstellen von Diagrammen

Betrachtet man Audiosignale, so handelt es sich z. B. bei einer CD um 2 Datenvektoren (Links und Rechts für Stereo), die aus 44100 Werten pro Sekunde bestehen. Eine solch lange Zahlenkolonne ist nicht sehr übersichtlich. Deshalb werden größere Datenmengen i. allg. auf grafische Weise dargestellt. Diese Visualisierung von Daten ist die Hauptstärke von Matlab. Dementsprechend mächtig sind die dazugehörigen Befehle. Zunächst wollen wir uns auf die wichtigsten Funktionalitäten beschränken.

Der zentrale Befehl für zwei-dimensionale Darstellungen ist `plot`. Er akzeptiert eine variable Anzahl von Eingabeparametern, die das Aussehen der gezeichneten Daten beeinflussen. Mit nur einem Parameter gibt Matlab die Daten als verbundene Linie aus. Die x-Achse ist der (ganzzahlige) Index der Daten, beginnend bei 1.

Beispiel:

```
v_y = [-1 2 1 3 -1 5 4 3 2 1 3];
plot(v_y);
```

erzeugt

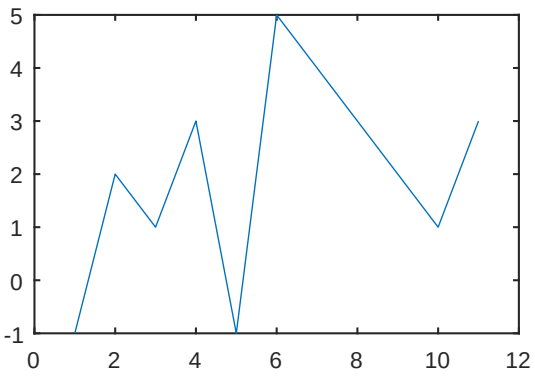

Abbildung 4: Einfacher Matlab-Plot eines Vektor gegen die Element-Indizes

Die Übergabe zweier (gleichlanger) Vektoren an `plot` führt dazu, dass Matlab den ersten Vektor als x-Werte und den zweiten Vektor als y-Werte interpretiert und jeweils Werte y_i als Funktion von x_i für alle i darstellt:

Beispiel:

```
v_x = [-5 -1 1 2 2.5 3 4 8 10 16 27];
v_y = [-1  2 1 3 -1  5 4 3  2 1  3];
plot(v_x,v_y);
```

erzeugt

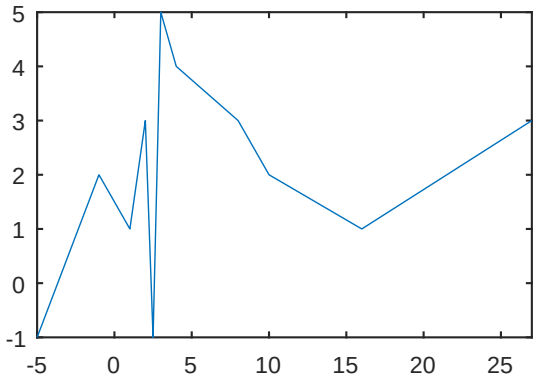

Abbildung 5: Einfacher Matlab-Plot zweier Vektoren gegeneinander

`plot` stellt die Daten jeweils in dem aktuellen „Figure"-Fenster dar. Um einen neuen Plot in einem neuen Fenster zu erhalten, kann mit `figure` ein neues Fenster geöffnet und gleichzeitig zum aktuellen Fenster gemacht werden. Dies hilft z.B. um zwei Darstellungen besser vergleichen zu können. Die einzelnen Figures werden mittels einer Integerzahl angesprochen. Um die Figure mit einer bestimmten Nummer `n` zu eröffnen bzw. wieder zur aktiven Figure im Vordergrund zu machen dient der Befehl `figure(n)`.

6.2.2 Formatierung von Diagrammen

Das dritte Argument von `plot` dient dazu, die Darstellungsfarbe, die Linienform und die Art der Punktmarker festzulegen. Diese Parameter werden durch einen String festgelegt, der in Hochkommata eingeschlossen wird. Beispiel: `'--gx'` erzeugt eine grüne (`g`) gestrichelte Linie (`--`), wobei jeder Datenpunkt (Marker) durch ein kleines x (`x`) gekennzeichnet wird. Folgende Optionen stehen zur Verfügung:

Linienart		**Farbe**		**Punktform**	
.-	Punkt-Striche	b	blau	x	Kreuze
-	durchgezogen	r	rot	o	Kreise
- -	gestrichelt	g	grün	.	Punkte
:	gepunktet	m	magenta	+	Plus-Zeichen
		y	gelb	s	Quadrate
		c	cyan	v	Dreiecke (Spitze unten)
		k	schwarz	^	Dreiecke (Spitze oben)
				<	Dreiecke (Spitze links)
				>	Dreiecke (Spitze rechts)
				p	Pentagramme
				h	Hexagramme
				d	Diamanten

Beispiel:

```
figure(1)  % plotte stets in genau DIESE Figure
v_y = randn(20,3);
plot(v_y(:,1),'-gd');
hold on;  % hinzufuegen statt ueberschreiben
plot(v_y(:,2),'--k>');
plot(v_y(:,3),'m:+');
hold off;
grid on;
xlabel('X-Werte');
ylabel('Y-Werte');
title('Name der Grafik');
```

erzeugt eine Figure wie in Abb. 6.

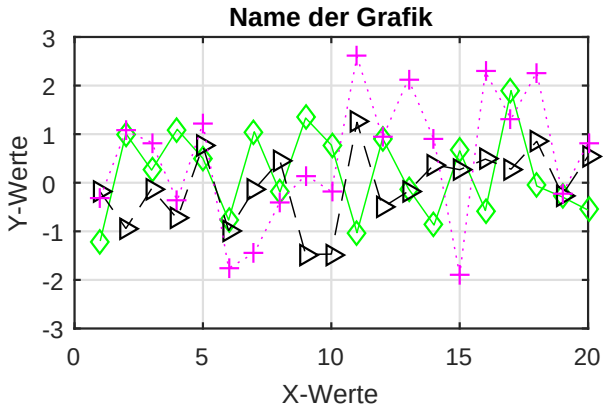

Abbildung 6: Drei verschiedene Kurven, gekennzeichnet durch Marker, Farbe und Strichart.

Einige weitere wichtige Befehle im Zusammenhang mit Diagrammen sind:

`xlabel ('LabelText')` erzeugt einen Beschriftung der x-Achse, z. B. `xlabel('Zeit in s')`.

`ylabel ('LabelText')` erzeugt eine Beschriftung der y-Achse, z. B. `ylabel('Amplitude')`.

`title ('TitelText')` erzeugt einen Titel über der Grafik, z.B. `title('Wellenform des Sprachsignals')`.

`grid on` erzeugt ein Gitternetz zum einfacheren Ablesen der Datenwerte.

`axis([xmin xmax ymin ymax])` ermöglicht es, nur Ausschnitte innerhalb gewählter minimaler und maximaler *x* und *y*-Werte zu betrachten. Dies ist ebenfalls nützlich, um z.B. die Darstellung mehrerer Diagramme zu vereinheitlichen, und eine bessere Vergleichbarkeit der Daten zu gewährleisten.

`xlim([xmin xmax])`, `ylim([ymin ymax])` ermöglichen ebenfalls das Betrachten eines Ausschnittes, beschränken sich jedoch speziell auf jeweils eine Achse.

`zoom on` Ermöglicht durch Ziehen eines Rechteckes mit der Maus die Auflösung des Diagramms grafisch zu verändern

6.2.3 Speichern von Diagrammen

Wichtig ist, die Diagramme in andere Programme übernehmen zu können. Für WYSIWYG-Programme („What you see is what you get.“) ist das Kopieren über die Menü-Leiste der Figure möglich. Alternativ kann man die Grafik mit `print -dmeta FileName.emf` in eine Datei im Windows-Vektorgrafik-Format drucken. Um die Abbildung in LaTeXeinbinden zu können, ist das Drucken in eine eps-Datei (Encapsulated Postscript) notwendig. Durch `print -deps FileName.eps` geschieht dies in Schwarz-Weiß, `print -depsc FileName.eps` macht dasselbe in Farbe. Für `pdflatex` eignet sich das eps-Format leider nicht. Stattdessen verwenden Sie hier den Befehl `print -dpdf FileName.pdf`. Alternativ verwenden Sie das LaTeX-Package eps2pdf, ein Programm dass die notwendige Umwandlung durchführt. In modernen Latex-Distributionen sind hierfür auch schon automatische Lösung vorgesehen. Weitere Druckformate liefert Ihnen `help print`.

6.3 Sound Ausgabe

Um Audiosignale hörbar zu machen, bietet Matlab mehrere Befehle. Bei der Ausgabe mittels `sound(v_Vektor, n_SamplingFrequenz)` interpretiert Matlab die einzelnen Elemente des Vektors als aufeinanderfolgende Abtastwerte des Signals. Dabei ist es **wichtig**, dass die Abtastwerte als ein *Spalten*-Vektor vorliegen. Für den Fall eines Mono-Signal wandelt `sound` einen Zeilenvektor automatisch vor der Ausgabe in einen Spaltenvektor um. Beim Versuch der Ausgabe eines falsch herum orientierten Stereo- oder Mehrkanal-Signals (Zeilen statt Spalten) mittels `sound` ergibt sich jedoch eine Fehlermeldung.

Es gilt die Konvention, dass ein Abtastwert von 1.0 (floating point) der maximal vollausgesteuerten Amplitude des D/A-Wandlers entspricht. Entsprechend ist -1.0 die betragsmäßig größte negative Amplitude. Für den Fall, dass die Abtastwerte des Signale viel zu groß oder viel zu klein sind im Vergleich zum Bereich des DA-Wandlers zwischen $-1 \leq x \leq 1$, gibt es den Befehl `soundsc(v_Vektor, n_SamplingFrequenz)`. Dabei wird der Vektor zunächst auf maximalen bzw. minimalen Wert 1 bzw. -1 normalisiert und erst dann via Soundkarte wiedergegeben.

⚠ **Achtung: Dies führt in der Regel dazu, dass ein wiedergegebenes Signal sehr laut wird!**

7 Programmierstrukturen

In der Informatik-Vorlesung haben Sie gelernt, dass jeder Algorithmus sich in drei Grundbausteine zerlegen lässt. Diese drei Grundbausteine sind auch in Matlab implementiert.

7.1 Sequenz

Die Sequenz ist eine Abfolge von Befehlen.

7.2 Iteration

Als Iteration wird die Wiederholung eines bestimmten Programmteils bezeichnet. Hierzu sind aus anderen Programmiersprachen Schleifen bekannt. Genau dieses Konzept setzt auch Matlab um.

7.2.1 `for` Schleife

Ein sehr wichtiges Konstrukt ist die einfache Zählschleife, die durch das Befehlspaar `for` und `end` angezeigt wird. Zur Erklärung der Syntax, nehmen wir das folgende Beispiel:

```
% Script zur Summation von ganzen Zahlen
n_OberGrenze = 10;
n_Summe = 0;
for kk = 1:n_OberGrenze
    n_Summe = n_Summe+kk;
end
```

In diesem Beispiel wird der Bereich zwischen `for` und `end` genau `n_OberGrenze`-mal wiederholt.

7.2.2 `while` Schleife

Eine andere Möglichkeit stellen Schleifen mit Abbruchsbedingungen dar. Die bekannteste Art ist die `while`-Schleife, die ebenfalls durch `end` beendet wird. Nehmen wir an, wir möchten wissen, wieviele aufsteigende ganze Zahlen notwendig sind, um eine bestimmte Summe zu erreichen oder zu überschreiten. Dies könnte man durch die folgende `while`-Schleife lösen.

```
n_ZuErreichendeSumme = 100;
n_Summe = 0;
kk = 1;
while (n_Summe < n_ZuErreichendeSumme)
    n_Summe = n_Summe+kk;
    kk = kk + 1;
end
kk
```

7.2.3 `break`: Schleifenabbruch

Die Abarbeitung einer Schleife endet im Normalfall, wenn die vorgesehene Anzahl der Durchläufe der `for`-Schleife erreicht ist oder die Bedingung der `while`-Schleife erfüllt ist.

Die weitere Durchführung einer Schleife kann jedoch auch explizit vorzeitig durch den Befehl `break` abgebrochen werden. Die Programmausführung verlässt bei einem `break`-Befehl nur die jeweils innerste Schleife. Dies kann z.B. als eine zwischenzeitliche Maßnahme während einer Fehlerbehandlung hilfreich sein.

7.3 Selektion

Die Selektion ist die Auswahl unter bestimmten Bedingungen. Dies wird in Matlab durch die Befehle `if`, `elseif` und `else` sowie `switch` erreicht. Eine einfache Abfrage könnte zum Beispiel so aussehen.

```
if (kk==10)
    % Aktion
end
```

Soll eine Alternative zur Verfügung stehen, wird dies durch `else` angezeigt.

```
if (kk==10)
    % Aktion
else
    % Default Aktion
end
```

Sollen mehrere Auswahlblöcke zur Verfügung stehen, können diese durch `elseif` getrennt werden.

```
if (kk==10)
    % Aktion 1
elseif (kk ==11)
    % Aktion 2
else
    % Default Aktion
end
```

Bei einer größeren Anzahl von Entscheidungsmöglichkeiten empfiehlt sich die Verwendung der `switch`-Selektion:

```
% Je nach Inhalt der Variable "Bedingung" werden verschiedene Aktionen ausgefuehrt
switch Bedingung
    case 1
        % Aktion 1 falls der Inhalt "1" entspricht
    case 2
        % Aktion 2 falls der Inhalt "2" entspricht
    case 3
        % Aktion 3 falls der Inhalt "3" entspricht
    otherwise
        % Default Aktion bei anderem Inhalt
end
```

7.4 Logische Operatoren

In Abfragen für die Iteration oder Selektion werden Bedingungen logisch geprüft. Diese können durch logische Operatoren auch verknüpft werden.

Operator	logische Verknüpfung
&	UND
|	ODER
~	NICHT

Als Bedingungen werden i. allg. Abfragen nach >,>=,<=, < und == genutzt, welche der Notation in C entsprechen. Die Abfrage auf *Ungleichheit* lautet in Matlab jedoch abweichend davon ~=.

Die logischen Operatoren `&`, | und ~ sind elementweise zu verstehen, falls einer die Operanden links und/oder rechts davon Vektoren bzw. Matrizen sind. Beispiel:

```
a = (1:6)
b = a > 3
c = a < 6
d = b & c
```

ergibt `d = [0 0 0 1 0 0]`.

```
e = b | d
```

ergibt `e = [0 0 0 1 1 1]`.

```
f = ~e
```

ergibt `f = [1 1 1 0 0 0]`.

7.5 Verknüpfung von Bedingungen

Sollen zwei Bedingungen wie im nachfolgenden Beispiel miteinander verknüpft werden, geschieht dies mit doppelten Operatoren. Dargestellt ist einmal der Fall in dem sowohl die erste, als auch die zweite Bedingung erfüllt sein müssen → **UND**. Im zweiten Fall reicht die Erfüllung einer der beiden Bedingungen (wobei auch beide Bedingungen erfüllt sein dürfen → **UND/ODER**):

```
% UND:
if (Bedingung1 && Bedingung2)
    % Aktion
end

% UND/ODER:
if (Bedingung1 || Bedingung2)
    % Aktion
end
```

Konstruktionen mit mehr als zwei Bedingungen sind ebenfalls möglich.

8 Programmieren von Funktionen

Bisher haben wir nur sog. Skript-Files kennen gelernt, die immer alle Daten im globalen Speicher (in Matlab Workspace genannt) zur Verfügung stellen. Bei vielen Aufgaben ist es aber sinnvoll, kleine Programme zu schreiben, die für sich in einem separaten Speicher ablaufen. Solche Unterprogramme werden wie auch in den meisten anderen Programmiersprachen Funktionen genannt. In Matlab werden diese Funktionen durch das Schlüsselwort `function` gekennzeichnet. Man speichert diese Funktionen in einem eigenem File, das den selben Namen, wie die Funktion trägt.

⚠ **Achtung: der Filename ist für Matlab entscheidend dafür, unter welchem Namen die Funktion in Matlab bekannt ist, nicht der Name innerhalb der Funktion.**

Ein Beispiel für ein solche Funktion ist der folgende Sinuston-Generator.

```
function v_SinSignal = generate_sinus(n_F0, n_Fs, n_Dur_sec)

% generate a sinusoidal waveform
% ------------------------------------------------
% Usage: v_SinSignal = GenerateSinus(n_F0, n_Fs, n_Dur_sec)
% Input Paramter
%     n_F0          : desired frequency in Hertz
%     n_Fs          : sampling frequency in Hertz
%     n_Dur_sec     : duration of the signal in seconds
% Output Paramter
%     v_SinSignal : vector containing the sinus of desired duration and frequency

% Author:  J. Bitzer, M. Hansen
% Date:    22.09.2003, 28.9.2005
% History: Version 1.0 (JB) Ersterstellung
%                  1.1 (JB) Angepasst an Matlab
%                  1.2 (MH) Wahl eines Spalten-Vektors fuer k, v_Time, SinSignal
```

```
i_NrSamples = round(n_Fs*n_Dur_sec); % number of samples during duration dur_sec
v_K         = (0:i_NrSamples-1)';    % counter v_K from 0 to N-1, column vector(!)
n_Dt        = 1/n_Fs;                % time interval or sampling period

v_Time      = v_K*n_Dt;              % vector containing the time, starting at t=0
n_Amplitude = 1;                     % amplitude
n_Phi       = 0;                     % starting phase 0

v_SinSignal = n_Amplitude*sin(2*pi*n_F0*v_Time + n_Phi); % sine with start phase phi

% End of file generate_sinus.m
```

Von besonderer Bedeutung ist dabei u.a. eine passende Wahl *der ersten Kommentarzeile* im File. Diese sollte unbedingt eine kurze prägnante Zusammenfassung darüber enthalten, was die Funktion kann. Der Inhalt dieser Zeile (nur EINE Zeile, der sogenannte „H1-Header“) wird von der Matlab-Funktion `lookfor` ausgewertet. Dieser Aufruf sucht in allen im Matlab-Pfad verfügbaren Funktionen nach Stichworten. Zur Verwendung der weiteren Kommentarzeilen, siehe u.a. in Abschnitt 10.2.

Es können, im Gegensatz zu C, auch mehr als nur eine einzige Variable von der aufgerufenen Funktion zurückgegeben werden, z. B. könnte im vorherigen Fall der Zeitvektor ebenfalls nützlich sein, um die Diagramm-Darstellung zu erleichtern. Die Funktionsdeklaration im File `generate_sinus.m` würde dann folgendermaßen aussehen:

```
function [v_SinSignal,v_Time] = generate_sinus(n_F0,n_Fs,n_Dur_sec)
```

Ein möglicher Aufruf wäre dann:

```
n_GrundFreq         = 100;
n_SampRate          = 44100;
n_LenInSec          = 0.1;
[v_Signal,v_TimeVek]  = generate_sinus(n_GrundFreq,n_SampRate,n_LenInSec);
plot(v_TimeVek,v_Signal,'r');
```

Hier ist sehr gut der Unterschied zwischen den inneren Variablenbezeichnungen einer Funktion und den Variablennamen außerhalb der Funktion zu erkennen. Für die Funktion ist nur wichtig, wie viele Variablen welchen Typs sie empfängt und wie viele sie zurück gibt. Wie diese heißen bevor sie in die Funktion geschickt werden, bzw. unter welchem Namen sie am Ausgang der Funktion abgegriffen werden, ist für sie unerheblich.

9 Breakpoints

Um Ihren Code zur Laufzeit überprüfen zu können, bietet Matlab eine Breakpoint-Funktion an. Diese ermöglicht es Ihnen, die Ausführung des Codes an einer beliebigen Stelle anzuhalten um beispielsweise Variablen auf ihren Inhalt zu überprüfen. Sie können die Ausführung ebenfalls Schritt für Schritt verfolgen um Funktionsaufrufe nachzuvollziehen. Zu diesem Zweck setzen Sie einen Breakpoint an einer (oder mehreren) gewünschten Stelle(n) im Code. Sobald Matlab während der Ausführung an dieser Stelle angelangt ist, wird die Ausführung des Programms unterbrochen. Wenn Sie nun den Mauszeiger über eine Variable bewegen, werden Ihnen Informationen zu dieser Variable angezeigt. Auch können Sie über die Commandline Werte abfragen oder Inhalte plotten.

Normaler Breakpoint: Einen normalen Breakpoint setzen Sie, indem Sie im Editor am linken Rand in die Spalte zwischen Zeilennummer und eigentlichem Code klicken. Alle Stellen, an welchen ein Breakpoint gesetzt werden kann, sind mit einem „-“ gekennzeichnet. Ein weiterer Klick löscht den Breakpoint wieder.

Conditional Breakpoint: Die zweite Art von Breakpoints ist jeweils an eine Bedingung geknüpft, welche Sie aufstellen. Ist der Inhalt einer Variable beispielsweise an der gewünschten Stelle gleich Null, so soll der Code angehalten werden. Hat die Variable einen anderen Wert, so wird der Breakpoint einfach übergangen. Erstellen können Sie diese Art von Breakpoints mit einem Klick der rechten Maustaste auf das „-“ und nachfolgender Selektion der Option „Set Conditional Breakpoint“. In dem sich dann öffnenden Fenster wird die Bedingung eingegeben.

10 Style-Guide für Matlab-Programme

Das Erstellen von Programmen hat grundsätzlich das Ziel, bestimmte Aufgaben durch einen automatischen Ablauf zu lösen. Zur Lösung können häufig schon bekannte Lösungen mit einbezogen werden. Dies erfordert meistens, dass man sich mit einem Programmteil einer anderen Person auseinander setzen muss. Genauso häufig ist der Fall, dass man seinen eigenen Programmcode in seiner Funktionalität erweitern möchte. Um diese Wieder- und Weiterverwendung zu vereinfachen, werden in vielen Firmen sog. Style-Guides eingesetzt, die sehr genau festlegen in welchem Stil programmiert werden soll. Für Matlab-Programme in den Studiengängen des TGM sollten ebenfalls einige Regeln genutzt werden, um eine möglichst gute Austauschbarkeit und Kompatibilität zu gewährleisten.

Es gibt es viele verschiedene Programmierstile, jeweils mit ihren spezifischen Vorteilen und vehementen Fürsprechern, sowie auch Nachteilen. Im folgenden wird *ein* gewisser Style-Guide beschrieben.

10.1 Namenskonventionen

Bei der Namenskonvention sind einige Grundregeln zu beachten:

- Benutzen Sie *Telling Names*. Dies bedeutet, jede Variable sollte einen Namen erhalten, der ihrem Sinn entspricht. Beispielsweise sind `a,aa,aaa,aaaa,u,uu,uuu` keine guten Variablennamen. Für die Samplingfrequenz `fs` zu verwenden ist grenzwertig. Besser ist `sampfreq`.

- Benutzen Sie englische Ausdrücke. Die Wahrscheinlichkeit, dass Sie selbst in einem internationalen Umfeld arbeiten oder Ihr Programm dort verwendet wird, ist sehr hoch. Deshalb sollten Sie sich schon jetzt daran gewöhnen, auch beim Programmieren englische Namen und Kommentare zu verwenden.

- Zur Kenntlichmachung von Vektoren und Matrizen bietet es sich an, dieses durch den Variablennamen auszudrücken. Häufig wird durch einen vorangestellten Buchstaben kenntlich gemacht, um welchen Datentyp es sich handelt. Also `v_name` für einen Vektor, `m_name` für ein Matrix. Eine Zeichenkette oder String wird oft durch `s_name` gekennzeichnet. Zusätzlich können z.B. Variablen mit konstant bleibenden Inhalt durch `c_name` gekennzeichnet sein.

10.2 Dokumentation

Einen sehr wichtigen Faktor zum Verständnis einer Funktion stellt die Dokumentation dar. Hierzu zählen Kommentare, Funktionserklärungen und Schnittstellenbeschreibungen. Die Schnittstellenbeschreibung dient auch als automatischer Hilfe-Text in Matlab, da Matlab beim `help` Befehl alle Zeilen ab Beginn des Files ausgibt, die bis zur ersten nachfolgenden Leerzeile als Kommentar gekennzeichnet sind. In diesem Block sollte der Funktionsaufruf und alle Ein- und Ausgabeparameter kurz erläutert sein. Die allererste Kommentarzeile eines m-files dient einer Kurzbeschreibung. Der Inhalt dieser Zeile sollte so kurz und prägnant wie möglich sein. Die Zeile wird u.a. von `lookfor` ausgewertet, welches eine Stichwortsuche in allen verfügbaren Funktionen und Skripten innerhalb des Matlab-Baum ermöglicht.

Nehmen wir an, wir haben die Funktion `meanValue()` programmiert. Diese könnte aussehen wie folgt:

```
function n_MeanValue = meanValue(v_InputData)
% function to compute the mean value of given input data
%
% Usage: n_MeanValue = meanValue(v_InputData)
%
% Input Parameter:
%    v_InputData:           Incoming data vector
% Output Parameter:
%    n_MeanValue:           Mean value scalar of input data
%
%-------------------------------------------------------------------------
%
% Example: 3 = meanValue([1,3,5])

% Author: Peter Zwegat
% Version History:
% Ver. 0.01 initial create  06-Sep-2016  Initials (eg. PZ)

% Mean value is the sum of all data divided by the number of elements
n_MeanValue = sum(v_InputData)/length(v_InputData);

% End of file
```

Wird nun `help meanValue` in der Kommandozeile eingegeben, erscheint folgende Information:

```
function to compute the mean value of given input data

Usage: n_MeanValue = meanValue(v_InputData)

Input Parameter:
    v_InputData:           Incoming data vector
Output Parameter:
    n_MeanValue:           Mean value scalar of input data

-------------------------------------------------------------------------

Example: 3 = meanValue([1,3,5])
```

Zusätzlich sind stets Informationen zur Erstellung der Funktion anzugeben. Hierbei insbesondere der Autor, und Erstellungsdatum wichtig, weiterhin eventuelle Copyright-Vermerke, Versionsnummern und eine kurze History, wie die aktuelle Version entstanden ist. Falls eine Funktion aus einem Buch, Text oder einer anderen Quelle nachpro-

grammiert wurde, oder aus einer Vorgängerversion hervorgegangen ist, ist diese unbedingt vollständig aufzuführen und/oder entsprechende Vorarbeiten sind angemessen zu würdigen. Die Information zur Autorschaft können z.B. im Kommentar nach der ersten Leerzeile auftauchen, wodurch Sie für den Benutzer beim Aufruf von `help` nicht sichtbar sind, wohl jedoch für die Programmiererin.

Bei der Kommentierung des Programmiercodes gilt, dass nur die wenigsten Zusammenhänge selbsterklärend sind. Diese Selbsterklärung gilt zusätzlich meist nur für die Autorin selbst, und das auch nur für eine kurze Zeit (ca. 1 Woche bis 1 Monat). Ein nicht dokumentiertes Programm ist so gut wie wertlos, selbst für den Autor. Deshalb sollten die meisten Programmierzeilen einen kurzen Kommentar enthalten, der die Wirkung und Absicht dieser Programmierzeile beschreibt.

10.3 File Separator

Da nicht alle Betriebssysteme die gleichen Bezeichnungen für ihre Ordnerstruktur benutzen, sollten Sie anstelle des Slash „\“ in Matlab konsequent mit dem Betriebssystem-spezifischen *File Separator* arbeiten. Der Befehl hierfür lautet `filesep` und wird von jedem Betriebssystem nach eigener Vorgabe interpretiert, so dass sich Ihre Programme plattformübergreifend ausführen lassen. Technisch gesehen gibt der Aufruf `filesep` einen String mit dem jeweiligen Symbol zurück, welches Sie an der Stelle einfügen, an der es benötigt wird. Um einzelne Strings zu einem einzigen großen String zusammen zu fügen, werden übrigens eckige Klammern benutzt. Dies mag auf den ersten Blick umständlich erscheinen, zahlt sich jedoch auf lange Sicht **immer** aus. Beispiel für eine Ordnerstruktur:

```
% Herkoemmliche Struktur:
s_Folder = 'C:\Users\ab1234\Test';
% Mit filesep:
s_Folder = ['C:',filesep,'Users',filesep,'ab1234',filesep,'Test'];
```

Und nun viel Erfolg beim Schreiben Ihrer eigenen Programme!