Kapitel 1

MATLAB Einstieg

1.1 Einführung

MATLAB eignet sich für die Durchführung von Berechnungen der Lineare Algebra als Grundlage weitere numerischen Berechnungen anderer mathematische Gebiete.

MATLAB verarbeitet, analysier und visualisiert aus wissenschaftlichen Experimenten und Simulationen und erzeugt durch deren Daten Modelle.

MATLAB ist mit einer Vielfältigkeit von Funktionen ausgerüstet, welche die Durchführung sämtlicher Operationen der Lineare Algebra und die Darstellung der Resultate in graphischer Weise erzeugt. Im Laufe der Zeit wurden gruppierten Funktionen in verschiedene Anwendungsbereiche kreiert. Eine Toolbox eintspricht einer solchen speziellen Gruppe von sogenannten M-Files, explizit zusammengesetzt zur Lösung einer besonderen Klasse von Problemen.

MATLAB wurde geschrieben, umden raschen Zugang zu Matrix-Software zu erlauben, welche im Umfeld der LINPACK- und EISPACK-Projekte entwickelt wurde. Matrizen aus reellen oder komplexen Zahlen sind dementsprechend die Grundlemente für alle Operationen.

MATLAB verfügt über ein Online-Portal, welches zusätzliche Informationen erhält. Diese sind unter zu finden.

1.2 Einstieg

MATLAB wird durch Doppelklicken der entsprechende Ikone gestartet und und durch den Befehl im Command Window `quit` oder `exit` verlassen.

Um den Pfad zu den M-Files zu zeigen, schreibt man im Command Window den Befehl `pwd`. Muss einen neuen Pfad hergestellt werden, so muss man den neuen Path neu setzen durch den Befehl `cd` zum Wechseln zwischen Directories. Bei aufwendige Berechnungen ist es von grossem Vorteil, wenn man die

Befehlsabfolge in ein M-File schreibt, anstatt die einzelnen Befehle im Command Window einzutippen.

M-Files sind nichts anderes als gewöhnliche Text-Files, deren Namen mit `.m` enden. M-Files werden auch als "Scripts" bezeichnet. Die Ausführung von bestimmten Zeilen können im Command Window untersucht werden. Die Ausführung von M-Files erfolgt linear. Die Namen von M-Files müssen keine reservierte Wörter bzw. Variable enthalten.

Kommentare in Scripts sind sehr nützlich und erfolgt am Anfang einer Zeile durch ein Prozentzeichen `%` und somit MATLAB ignoriert den Inhalt dieser Zeile. Die Ausgabe von Resultate im Command Window kann man unterbinden, indem man am Ende einer Zeile einen Strichpunkt `;` anbringt. Die Befehle werden ausgeführt, aber nicht angezeigt. Der Wert einer bestimmten Variablen erfolgt durch den Befehl `disp` sowie Texte `disp('Text')` durch im Command Window. Der Befehl `input('any text')` verlangt eine Eingabe über die Tastatur. Mit `pause` stoppt man die laufende Evaluation des M-Files bis man eine beliebige Taste gedrückt hat. Mit dem Befehl `figure` öffnet man ein neues, für Graphiken reserviertes Fenster.

Mit M-Files kann man neue Funktionen definieren. Funktionen sind Prozeduren, welche allfällige Argumente verarbeitet und/oder eine Reihe von Befehlen ausführt um ein Resultat zu liefern. Die Name der Funktion muss mit der Name des M-Files übereinstimmen.

Der Name einer Variablen darf fast beliebig gewählt werden. Der Variablenname darf nicht mit dem Namen eines M-Files übereinstimmen. Der Name muss ein einziges Wort ohne Leerzeichen sein, aus maximal 31 Zeichen bestehen und mit einem Buchstaben beginnen. Variablennamen dürfen Zahlen und Underscores "_" enthalten. In MATLAB sind die Variablen von in M-Files definierten Funktionen lokal. Wenn der Variablenname "test" sowohl in einer Funktion als aich direckt im Command Window exsitieren soll, dann addressiert man diese Namen zwei verschiedene Werte im Speicher. Man kann in einer Funktion definierte Variable `global` deklarieren.

1.3 Allgemeine Befehle

1.3.1 Hilfe

Mit dem Befehl `help data` wird die zur Verfügung stehende Kommentare im File "`data.m`" angezeigt. Jeder Funktion in MATLAB beinhaltet einen Help-Abschnitt, welcher meistens aus einer ausführlichen Befehlssyntax besteht.

Mit dem Befehl `help` wird allein eine Reihe von Themen aufgelistet, aus welchen dann einzelne Begriffe abgefragt werden können.

Mit dem Befehl `demo` wird das MATLAB-eigene Demonstrationsprogramm gestartet. Nebst einer kurzen Einführung beinhaltet es eine praktische Beispiele.

1.3.2 Workspaces

Die Befehle `who` werden die Namen aller während der laufenden Session definierten Variablen aufgelistet. Mit `whos` werden ausführliche Informationen gezeigt.

Den Befehl `clear` löscht alle vom Benutzer definierten Variablen, es ist ratsam am Anfang jedes M-Files diesen Befehl einzuführen, um ältere Variablen aus dem Speicher zu entfernen. Mit der Variablen `clear var` löscht man nur die Variable `var`. Mit dem Befehl `clear na*` löscht man alle Variablen, die mit `na` beginnen. `clear all` entfernt alle Variablen, globale Variablen, Funktionen und MEX-Verbindungen.

Mit dem Befehl `load` lädt man Variablen oder Daten aus einem File in dem Workspace. Mit `load filename` werden die in einem MAT-File namens "filename.mat" abgespeicherte Variablen in den Arbeitsspeicher eingelesen. Mit dem Befehl `load filename X Y` werden nur die Variablen `X` und `Y` aus dem File eingelesen.

Den Befehl `save` speichert alle in der Arbeitsoberfläche definierten Variablen in ein File. Mit `save filename` werden alle definierten Varaiblen in einem MAT-File namens "filename.mat" abgespeichert. Der ganze Path muss angegeben werden, falls das File nicht im aktuellen Directory gespeichert werden soll. Der Befehl `save filename X` speichert nur die Variable `X` ab.

Mit den Befehlen `quit` oder `exit` wird MATLAB verlassen. Die definierten Variablen werden nicht automatisch gespeichert.

1.3.3 Funktionen suchen und editieren

Mit dem Befehl `edit filename` wird vom default Texteditor das File `filename` aufgemacht. Der ganze Path muss angegeben werden, falls das File sich nicht im aktuellen Directory befindet. Mit dem Befehl `lookfor string` wird in allen zur Verfügung stehenden M-Files und in allen Beschreibungen von eingebauten Funktionen nach der Zeichenfolge "string" gesucht. Eine Liste von möglicherweise zutreffenden Begriffen wird anschliessend im Command Window editiert. Der Befehl `lookfor` eignet sich zur Suche von Funktionen, wenn nur eine Vorstellung vorhanden ist, welche Begriffe, Wörter oder Wortabschnitte in Beschreibungsteil vorhanden sein könnten.

Mit dem Befehl `which filename` wird der Path des M-Files `filename` gezeigt. Mit `which xyz` wird beschrieben, was für ein Objekt `xyz` (Variable, Funktion, usw.) ist.

1.3.4 Ausgabeformat

Mit dem Befehl `format` wird das Ausgabeformat von numerischen Resultaten definiert: Fliesskommaformat mit 4 oder 14 Nachkommastellen oder wissenschaftliche Notation mit 5- oder 16-stelliger Mantissa und 2-stelligem Exponenten. Möglicher Auswahl sind: `format short`, `format long`, `format short e`, `format long e`.

1.3.5 Systembefehle

Mit dem Befehl `cd path` wird der Path gesetzt und damit das laufende Directory geändert. Mit dem Befehl `pwd` wird den aktiuellen Path angezeigt.

Mit dem Befehl `dir` werden die Directories und Files im aktiellen Directory aufgelistet. Der Befehl `dir .m*` listet alle M-Files im aktuellen Directory. Der Befehl `delete filename` löscht das File `filename`. Analog der Befehl `delete .m*` löscht alle M-Files im aktuellen Directory.

1.3.6 Spezielle Tasten

MATLAB führt eine Liste der während einer Session durchgeführten Befehle. Mit `Ctrl-P` für previous und `Ctrl-N` für next. Mit `Ctrl-B` für backward und `Ctrl-F` für forward läuft der Cursor rück- oder vorwärts über eine Zeile. Damit können Änderungen an einem schon eingetippten Befehl durchgeführt werden, ohne dass der Befehl gelöscht werden muss. Mit `Ctrl-A` mit dem Cursor an den Zeilenanfang bzw. mit `Ctrl-E` an den Zeilenende. Mit `Ctrl-U` wird eine Zeile gelöscht und mit `Ctrl-C` wird eine laufende Rechnung oder die Durchführung eines Befehls unterbrochen.

1.4 Zeichen und Operatoren

1.4.1 Spezielle Zeichen

Befehle und Funktionen, die Daten in Form von Vektoren, Matrizen oder Strings benötigen, sind mit einem Klammerpaar `( )` zu versehen. Runde Klammern haben eine höhere Priorität bei der Rechenoperationen.

Mit den eckigen Klammerpaar `[ ]` werden Vektoren und Matrizen definiert. Jede Zeile wird mit einem Strichpunkt abgeschlossen, die Kolonnen mit einem Komma oder Leertaste.

Mit einem Punkt `.` werden die Dezimalstellen einer rationalen Zahl angegeben. Ein Punkt vor den Operatoren bewirkt, dass die Multiplikation, die Links- und Rechtsdivision und die Exponentialfunktion elementweise abläuft.

Zwei Punkte `..` werden nur in Verbindung mit dem befehl `cd` gebraucht. Mit dem Befehl `cd ..` wechselt der Path vom aktuellen Directory in das hierarchisch nächst höherliegende Directory.

Drei Punkte `...` am Ende einer Zeile verbinden das Ende der Zeile mit dem Anfang der folgenden Zeile. Für eine übersichtliche Darstellung werden lange Zeilen oft mit drei Punkten zwei oder dreigeteilt.

Bei der Definition von Matrizen werden die Kolonnen durch Kommas `,` oder Leerzeichen getrennt. Befehle oder Funktionen können mehrere Eingabeargumente haben. Diese werden mit Kommas voneinander getrennt. Mehrere befehle auf einer Zeile werden ebenfalls mit Kommas getrennt. Dies sollte jedoch vermieden werden.

Bei der Definition von Matrizen werden die Zeilen mit einem Strichpunkt `;` abgeschlossen. Die Ausgabe von Zwischenresultaten wird im Command Window unterdrückt, wenn die entsprechende Befehlzeile mit einem Strichpunkt abgeschlossen wurde.

Buchstaben und Zeichen können innerhalb von zwei halben Anführungszeichen `' '` als Zeichenreihe dargestellt werden. 'Ein Text' ist ein Vektor. Jedes einzelnen Zeichen dieses Vektors wird intern im entsprechenden ASCII-Code abgespeichert. Ein halbes Anführungszeichen innerhalb eines beliebigen Textes wird mit zwei halben Anführungszeichen definiert.

Programme sollten ausführliche Kommantare `%` enthalten, damit das Programmierte zu einem späteren Zeitpunkt leichter nachvollzogen werden kann. Programme ohne Kommentare sind nutzlos. Ein Kommentar besteht aus einem Prozentzeichen und einem darauffolgenden Text. Es ist ratsam, ein M-File mit iner kurzen Inhaltsangabe hinter Prozentzeichen zu versehen. Insbesondere sollte vor jedem einzelnen Programm-Abschnitt ein kurzer Kommentar stehen.

Das Gleichheitszeichen `=` weist einer Variable einen bestimmten Zahlenwert zu.

1.4.2 Arithmetische Operatoren

Das `+`-Zeichen addiert Zahlen, Vektoren oder Matrizen. Bei der Addition von Vektoren oder Matrizen müssen die Dimensionen übereinstimmen.

Das `-`-Zeichen subtrahiert Zahlen, Vektoren oder Matrizen. Die Matrixdimensionen müssen bei der Subtraktion ebenfalls übereinstimmen.

Mit `*` werden die Matrizen oder Vektoren miteinander multipliziert. Dabei ist zu beachten, dass bei der Multiplikation zweier Vektoren oder Matrizen `A*B` die Anzahl Kolonnen von `A` mit der Anzahl Zeilen von `B` übereinstimmen. Mit `.*` multipliziert man Vektoren oder Matrizen elementweise. Die Anzahl Zeilen und Kolonnen von `A` und `B` müssen identisch sein.

Die linke Division `\` wird für das Lösen von linearen Gleichungssystemen verwendet. Für das lineare Gleichungssystem `Ax=b` lautet die nach `x` aufgelöste Gleichung $x=A^{-1}b$. Dafür wird der Befehl `A\b` verwendet. `A` ist eine quadratische Matrix und `b` ein Kolonnenvektor.

Falls das lineare Gleichungssystem die Form `xA=b` hat, kommt die rechte Division `/` zur Anwendung. Die Lösung ist $x=bA^{-1}$, und der dazugehörige MATLAB-Befehl lautet `b/A`. Wie bei der Multiplikation können die Vektoren und Matrizen elementweise dividiert werden. Vor das Divisionszeichen ist dafür ein Punkt zu setzen.

Mit dem Zeichen `^` kann auf quadratische Matrizen angewandt werden, um $A \wedge n$ zu berechnen. `n` ist eine Integer. Für `n>0` wird die Matrix `A` `n` mal mit sich selbst multipliziert. `n=0` ergibt die Identitätsmatrix. `n<0` entspricht der Inversen von `A`.

1.4.3 Logische Operatoren

Das logische "und" `&` vergleicht zwei gleich grosse Objekte miteinander. Der logische Vergleich erfolgt elementweise. Somit sind auch Vektoren und Matrizen untereinander vergleichbar.

Haben die Matrizen `A` und `B` dieselbe Grösse, so ergibt sich aus `A&B` wiederum eine Matrix derselben Grösse mit Nullen und Einsen. Ist sowohl das eine Element von `A` als auch das entsprechende von `B` ungleich Null, so erhält die Antwort-Matrix an derjenigen Stelle eine Eins.

Ist sowohl das eine Element von `A` als auch das entsprechende von `B` ungleich Null, so erhält die Antwort-Matrix an derjenigen Stelle eine Eins. Ist irgendein Element `A` oder `B` gleich >Null, dann schreibt MATLAB an derselben Stelle eine Null.

Für das logische "oder" `|` gelten die gleichen Vorschriften wüe für das logische "und". Ein Element der resultierenden Matrix ist jedoch nur dann Null, falls das Element von `A` und das entsprechende von `B` ebenfalls Null sind.

Für das logische "nicht" `~` gelten dieselben Vorschriften wie für das logische "und". Der Befehl `~A` wandelt alle Elemente der Matrix `A`, die den Wert Null haben, in eine Eins und alle anderen in eine Null um.

Das doppelte Gleichheitszeichen `==` vergleicht zwei Matrizen miteinander. Es kann aber auch ein Vektor und eine Matrix oder ein Skalar und ein Vektor gegenübergestellt werden. Der Vektor ird mit den einzelnen Zeilen oder Kolonnen der Matrix verglichen und der Skalar mit den einzelnen Elementen des Vektors. Dieser Befehl wird häufig bei der Programmierung von "while" Schlaifen, "if-else-end" Beziehungen oder "switch-case" Konstruktionen verwendet.

`>`, `<`, `<=`, `>=` sind Vergleichsoperatoren zweier Argumente. Gleichheit liefert aber die Antwort "falsch" bzw. den Wert 0.

1.5 Elementare Mathematik

1.5.1 Trigonometrie

Die Funktionen `sin(x)`, `cos(x)`, `tan(x)`, `cot(x)` berechnen die trigonometrischen Funktionen von x in Radians, wobei x eine beliebige Zahl ist.

1.5.2 Exponential- und Logarithmusfunktionen

Der Befehl `exp(x)` berechnet die Exponentialfunktion von x. Der Befehl `ln(x)` berechnet den natürlicher Logarithmus von x. Der Befehl `sqrt(x)` berechnet die Quadratwurzel von x, wobei x eine beliebige Zahl ist.

1.5.3 Komplexe Zahlen

Der Befehl `abs(z)` berechnet den Absolutwert von z. Falls x eine komplexe Zahl ist, wird den Betrag von x ermittelt. Der Befehl `angle(z)` berechnet den Winkel von z in Radians, wobei z eine komplexe Zahl ist. Die komplexe Zahl besteht aus einem Real- und einem Imaginäranteil z=x+yi. Der Winkel ergibt sich aus dem tangens von Imaginär- über Realteil. Der Befehl `conj(z)` berechnet die Konjugiertkomplexe der Zahl z, d.h. $\bar{z}$. Der Befehl `imag(z)` gibt den Imaginärteil der komplexen Zahl z wieder. Der Befehl `real(z)` gibt den Realteil der komplexen Zahl z wieder.

1.5.4 Runden

Der Befehl `fix(x)` rundet in Richtung Null auf die nächste ganze Zahl. Der Befehl `floor(x)` rundet die natürliche Zahl x auf den nächstkleineren ganzzahligen Wert. Mit `ceil(x)` wird die natürliche Zahl x auf den nächstgrösseren ganzzahligen Wert gerundet. Mit dem Befehl `round(x)` kann die Zahl x auf den nächstliegenden ganzzahligen Wert runden. Der Befehl `sign(x)` gibt das Vorzeichen von x an.

Listing 1.1: Einige Befehle zum Kapitel

```
>> help quit;
>> A = ones(3,2); %Matrix aus Einsen
>> whos(A); %Informationen zur Variable A
>> clear A %Loescht Variabel A
>> load /home/user/Daten/test %Laden einer Datei
>> save test; %Speichert Datei
>> lookfor corr %Suche von Funktionen
>> which A; %A ist eine Variable
>> which sin; %sin ist eine Funktion
>> format short; %Kurzer Format
>> format long; %Langer Format
>> format short e; %Exponentieller Format
>> format long e; %Exponentieller Format
>> pwd; %Aktueller Verzeichnis
>> sqrt(9); %Wurzel aus 9
>> plus(3,3); %Summe
>> mtimes(3,3); %Produkt
```

```
>> A(3,1)=0; %Ersetzen einer Komponente
>> A(:,1)=0; %Ersetzen der ersten Spalte
>> A(:,1)=[]; %Loeschen der ersten Spalte
>> A=[1 2 3]; %Zeilenvektor
>> B=[1; 2; 3]; %Spaltenvektor
>> C=[1 2 3; 4 5 6; 7 8 9] %Matrix
>> A*B; %Produkt von Vektoren: Zahl
>> B*A; %Produkt von Vektoren: Matrix
>> A*.B; %Elementenweises Produkt, A und B gleiche Dimension!
>> 7+12-3; %Arithmetische Operationen
>> A=[1 0 1; 0 1 1; 1 1 0]; B=[1;1;1]; %Zuweisungen
>> A\b; %Gleichungssystem loesen
>> inv(A); %Inverse Matrix
>> A^2; %Matrix Exponentialfunktion
>> A=[4 0 2 -3 0 1]; B=[2 1 0 5 0 4]; %Zeilenvektoren
>> A&B; %Logisches Und: [1 0 0 1 0 1]
>> A|B; %Logisches Oder: [1 1 1 1 0 1]
>> ~A; %Logisches Nicht: [0 1 0 0 1 0]
>> A==B; %Logisches Gleichheitszeichen: [0 0 0 0 1 0]
>> sin(pi); %1.224e-16
>> sin(pi/2); %1
>> exp(1); %2.781
>> log(0); %0, Log of zero, ans -Inf
>> sqrt(-1); %0 + 1.0000i
>> abs(-pi); %3.1416
>> abs(1-i); %1.4242
>> angle(1); %0
>> sqrt(-1); %0+1.0000i
>> angle(sqrt(-1)); %1.5708
>> conj(sqrt(-1)); %0-1.0000i
>> imag(sqrt(-1)); %1
>> real(sqrt(-1)); %0
>> fix(1.99); %1
>> fix(-1.99); %-1
>> floor(1.99); %1
>> floor(-1.01); %-2
>> ceil(1.1); %2
>> ceil(-1.9); %-1
>> round(1.5); %2
>> round(1.49); %1
>> sign(2); %1
>> sign(0); %0
>> sign(-2); %-1
```

Kapitel 2

Vektoren und Matrizen

2.1 Matrix-Manipulationen

2.1.1 Elementare Matrizen

Der Befehl `zeros` setzt alle Koeffizienten einer Matrix gleich Null. Der Befehl `zeros(n)` bildet eine `n`×`n` Matrix, deren Elemente alle den Wert Null besitzen. Der Befehl `zeros(n,m)` steht für eine Rechtecksmatrix mit `n` Zeilen und `m` Kolonnen mit lauter Nullen.

Der Befehl `ones` weist allen Koeffizienten einer Matrix den Wert Eins zu. Der Befehl `ones(n)` bildet eine quadratische Matrix, deren Elemente alle den Betrag Eins haben. Der Befehl `ones(n,m)` bildet eine Matrix aus `n` Zeilen und `m` Kolonnen. Jeder Variable muss ein numerischer Wert zugewiesen werden.

Der Befehl `eye(n)` bildet die quadratische `n`×`n`-Identitätsmatrix. Der Befehl `eye(n,m)` definiert eine Rechtsmatrix mit `n` Zeilen und `m` Kolonnen. Die Diagonalelemente haben den Betrag Eins.

Mit `diag(a)` wird der Vektor `a` in der Diagonale einer quadratischen Matrix eingebettet. Die Koeffizienten ausserhalb der Diagonale sind Null. Der `diag(A)` bildet die Diagonale einer beliebigen Matrix in einem Kolonnenvektor ab.

Der Befehl `rand(n)` weist allen Koeffizienten einer quadratischen Matrix mit gleichmässig verteilte Zufallszahl zwischen Null und Eins. Der Befehl `rand(n,m)` hat `n` Zeilen und `m` Kolonnen mit gleichmässig verteilten Zufallszahlen. Der Befehl `rand(n,m,p)` erzeugt `p` Matrizen mit `n` Zeilen und `m` Kolonnen.

Der Befehl `linspace(xstart, xend)` erzeugt einen Vektor zwischen `xstart` und `xend`, der in 99 gleiche Intervalle unterteilt wird. Der Vektor besteht somit aus 100 linear, gleichmässig verteilten Punkten. Der Befehl `linspace(xstart, xend, n)` bildet einen Vektor mit `n` linear unterteilten Punkten zwischen `xstart` und `xend`.

Der Befehl `[X,Y]=meshgrid(x,y)` formt aus den Vektoren `x`$\in \mathbb{R}^m$ und `y`$\in \mathbb{R}^n$

die Matrizen X und Y mit je n×m Elementen. Die Matrizen X und Y werden für das Plotten von Funktionen mit zwei Varaiblen und für dreidimensionale Oberflächen Graphiken verwendet.

Der Befehl `[X,Y,Z]=meshgrid(x,y,z)` formt ein dreidimensionaler Gitter. Die Matrizen X, Y und Z werden für das Plotten von Funktionen mit drei Varaiblen und für dreidimensionale Volumen Graphiken verwendet.

2.1.2 Informationen über die Dimension

Der Befehl `size(A` informiert über die Dimension der Matrix A. Die erste Zahl des zweizeiligen Ausgabevektors steht für die Anzhl Zeilen von A, die zweite für die Anzahl Kolonnen. Mit `[M,N]=size(A)` werden die Zeilen- und kolonnenzahl von A den Variablen M und N zugewiesen. Der Befehl `size(A,1)` gibt Auskunft über die Anzahl Zeilen der Matrix A und `size(A,2)` über die Anzahl Kolonnen.

Der Befehl `length(a)` ermittelt die Anzahl Zeilen des Kolonnenvektors a bzw. die Anzahl Kolonnen des Zeilenvektors a.

Der Befehl `disp(X)` editiert im Command Window die in X definierte Zeilenfolge. Dabei wird der Namen der Zeilenfolge bei der Ausgabe unterdrückt. Ist X ein String, so erscheint ein beliebiger Text im Command Window. Damit können in M-Files Kontrollpunkte eingefügt werden. Sobald der Rechner im M-File eine bestimmte Teilaufgabe erfolgreich gelöst hat, kann dies mit `disp('test')` im Command Window angezeigt werden. Für diese Kommunikation zwischen M-File und Command Window eignen sich die Befehle `disp` und `input`.

2.1.3 Spezielle Variablen und Konstanten

Die zuletzt berechnete Ausgabe wird der Variable `ans` zugewiesen, falls kein anderer Namen definiert wurde. Die Konstante `eps` gibt den Abstand an, der zwischen der Zahl Eins und der nächstmöglichen Fliesskomma-Stelle liegt. Es ist der kleinste Wert, der zu Zahl Eins addiert werden kann, damit sich die Summe von Eins unterscheidet. Mit `realmax` eruiert MATLAB die grösstmögliche positive reelle Zahl, die der Computer berechnen kann. Mit `realmin` eruiert MATLAB die kleinstmögliche positive reelle Zahl, die der Computer berechnen kann. Die Konstante `pi` gibt das Verhältnis zwischen dem Kreisumfang und dem Kreisdurchmesser und kann mit bis 16 Stellen ermittelt werden. MATLAB antwortet mit dem Ausdruck `inf`, falls ein Wert gegen unendlich geht. Strebt während einer Berechnung ein bestimmter Wert gegen unendlich, so wird dies im Command Window angezeigt.

2.2 Lineare Algebra

2.2.1 Grundoperationen

Die linke Division `\` von A\B entspricht der Multiplikation der Inversen der regulären Matrix (regulär: `det(A)`$\neq$`0`) A mit der Matrix B, d.h. `inv(A)*B`. Dasselbe

gilt für `A\B` und `B*inv(A)`. `A\b` ist die Lösung des linearen Gleichungssystems `Ax=b`, falls `A`$\in \mathbb{R}^{n\times n}$ und `Rang(A)=n`. Für ein unterbestimmtes Gleichungssystem `n<m` gibt es unendlich viele Lösungen. MATLAB sucht sich selber eine aus. Bei einer Rechtsmatrix `A`$\in \mathbb{R}^{n\times m}$ mit `n>m` handelt es sich um ein überbestimmtes Gleichungssystem. Das System hat keine exakte Lösung. Mit `A\b` liefert MATLAB jene approximierte Lösung, für die der totale Fehler `e` für alle `n` Gleichungen am kleinsten ist. Dieser entspricht im MATLAB der Summe aller Fehler im Quadrat.

$$e = \sum_{i=1}^{n} \left(b_i - \sum_{j=1}^{m} a_{i,j} x_j \right)^2 = \left\| b - Ax \right\|_2^2 \tag{2.1}$$

Sei `A`$\in \mathbb{R}^{n\times m}$ eine reelle Matrix mit den Einträgen `[a_ij]`. Ihre Transponierte ist definiert durch $A^T = [a_{ji}]$. In MATLAB wird das halbe `'` Anführungszeichen für die transponierte `A'` der Matrix `A` verwendet. Ist `A` symmetrisch, so ist `A'=A`. `A.'` ist der Befehl für die nicht-konjugiert-transponierte Matrix einer komplexen Matrix `A`.

Der Befehl `inv(A)` berechnet die inverse Matrix der quadratischen regulären Matrix `A`. Die quadratische Matrix heisst regulär, wenn `det(A)`$\neq$`0` ist. Falls sie singulär ist, erscheint im Command Window eine Fehlermeldung. Die Definition der inversen Matrix von `A` lautet

$$X = A^{-1} = \frac{1}{\det(A)} [(-1)^{i+j} \det(A_{ji})] \tag{2.2}$$

Der Befehl `cond(A)` gibt mit einer reellen Zahl Auskunft über die Kondition des Systems, das durch Matrix `A` beschrieben wird. Sie ist grösser oder gleich Eins, und misst die Empfindlichkeit der Lösung `x` auf Störungen im linearen Gleichungssystem `Ax=b`. Die Kondition ist das Verhältnis zwischen dem grössten und dem kleinsten Singulärwert einer Matrix. Die Singulärwerte σ von `A` lassen sich mit der Euklidischen Norm berechnen. Sie sind die positiven Quadratwurzeln der grössten Eigenwerte der Hermiteschen Matrix `A`^{H}`A`.

$$\sigma_i(A = \sqrt{\lambda_i(A^H A)} \tag{2.3}$$

wobei A^H ist die konjugiert-transponierte Matrix von `A`$\in \mathbb{C}^{n\times n}$. Bei regulären Matrizen ist der kleinste Singulärwert immer grösser Null. Singuläre Matrizen sind nicht invertierbar, und haben einen Singulärwert bei Null. Die Kondition einer sungulären Matrix geht somit gegen unendlich. Eine grosse Zahl zeigt an, dass sich die betreffende Matrix nahe bei einer singulären Matrix befindet, d.h. das System ist schlecht konditioniert.

Der Rang einer Matrix `A` wird durch den Befehl `rank(A` bestimmt. Der Rang berechnet die Anzahl linear unabhängiger zeilen oder Kolonnen der Matrix `A`.

Die Norm eines Vektors ist ein Skalar. Er misst die Grösse bzw. die Länge eines Vektors. Die Euklidische Norm kann man berechnen durch `norm(a)`. Dies

entspricht `sum(abs(a).^2)^(1/2)`

$$\left\|a\right\|_2 = \sqrt{\sum_k \left|a_k\right|^2} \tag{2.4}$$

Der Befehl `norm(a,1)` berechnet die Summe der absoluten Beträge von jedem Vektorelement, `sum(abs(a))`

$$\|a\|_1 = \sum_k \left|a_k\right| \tag{2.5}$$

Der Befehl `norm(a, inf)` ermittelt die ∞-Norm von `a`. Sie steht für das betragsmässig grösste Zeilenelement von `a`, `max(abs(a))`.

Die Norm einer Matrix misst die Grösse bzw. den Betrag einer Matrix. Der Befehl `norm(A)` eruiert die Euklidische Norm der Matrix `A`. Sie entspricht dem grössten Singulärwert von `A`, `max(svd(A))`.

Der Befehl `norm(A,1)` berechnet die betragsmässig grösste Summe der absoluten Beträge von jedem Kolonnenelement von `A`, `max(sum(abs(A)))`.

Der Befehl `norm(A, inf)` ermittelt die betragsmässig grösste Summe der absoluten Beträge von jedem Zeilenelemente von `A`, `max(sum(abs(A')))`.

Der Befehl `det(A)` berechnet die Determinante einer Matrix `A`. Für eine Matrix `A`$\in \mathbb{R}^{n\times n}$, wobei `n`$>1$ ist und `j`-te Kolonnen, gilt für die Determinante von `A`

$$\det(A) = \sum_{k=1}^{n} (-1)^{j+k} a_{kj} \det(A_{kj}) \tag{2.6}$$

Die Spur einer Matrix `A` ist die Summe der Diagonalelemente. Der Befehl `trace(A)` berechnet die Sput einer Matrix.

Der Befehl `orth(A)` erzeugt die orthonormierte Basisvektoren einer Matrix `A`, die denselben Raum wie die Matrix `A` spannen. Die Anzahl Kolonnen der orthonormierten Basismatrix entspricht dem Rang einer Matrix `A`. Die Identitätsmatrix erhält man durch folgende Rechnung `(orth(A))'*orth(A)`

2.2.2 Eigenwerte und Singulärwerte

Die Eigenwerte λ berechnet man durch den Befehle `eig(A)` und die dazu gehörenden Eigenvektoren `x` einer Matrix `A`$\in \mathbb{R}^{n\times n}$ durch `[eigv, eigw]=eig(A)` und weist der Variable `eigv` die Eigenvektoren von `A` zu. In der Diagonalmatrix `eigw` befinden sich die dazugehörenden Eigenwerte.

Der Befehl `svd(A)` ermittelt alle Singulärwerte der Matrix `A`. Die Singulärwerte σ der Matrix `A` die positiven Quadratwurzeln der grössten Eigenwerte der Hermiteschen Matrix A^H. Die Singulärwerte geben an, um welchen Faktor sich

ein Vektor `x` durch die Abbildung mit der Matrix `A` in der Länge und der Richtung verändert. Sie charakterisieren das Verstärkungsverhalten einer beliebigen Matrix.

Listing 2.1: Einige Befehle zum Kapitel

```
>> A=[1 2 3; 4 5 6]; %Matrix
>> zeros(2,3); %Matrix aus Nullen
>> ones(2,3); %Matrix aus Einsen
>> eye(2,3); %Einheitsmatrix
>> diag(2); %Diagonalmatrix
>> diag(A); %1;5 Diagonalalemente
>> rand(2,4,3); %3 Matrizen aus Zufallselemente zwischen 0 und 1
>> linspace(0, 10, 11); %Vektor mit 11 Komponenten gleicher
    Abstand zwischen 0 und 10
>> x=[0 1 0]; y=[0 0.5 1]; %Zeilenvektoren
>> [X,Y]=meshgrid(x,y); %Zwei Matrizen X und Y
>> A=[1 2; 3 4; 5 6];
>> [M,N]=size(A); %M=3, N=2. M:Anzahl Zeilen, N: Anzahl Spalten
>> length([1 2 3 4]); %4
>> length([1; 2; 3]); %3
>> disp(['Die Matrix wurde invertiert']); %Widergabe einer
    Zeilenfolge, String oder Matrix
>> A=[1+i 2-2i; 3+3i 4-4i];
>> A'; %Transponierte Matrix
>> A.'; %Nicht konjugiert komplexe Matrix
>> A=[1 3 2; 0 4 2; 1 -1 -2]; %Neue Matrix A
>> cond(A); %Kondition einer Matrix
>> rank(A); %Rang einer Matrix
>> norm(A); %Norm einer Matrix
>> det(A); %Determinante einer Matrix
>> trace(A); %Spur einer Matrix
>> orth(A); %Orthonormierte Basisvektoren einer Matrix
>> [eigv, eigw]=eig(A); %eigv: Eigenvektoren, eigw: Eigenwerte
    einer Matrix
>> svd(A); %Singularwerte einer Matrix
```

Kapitel 3

Graphik

3.1 Zweidimensionale Graphik

3.1.1 Elementare zweidimensionale Graphik

Der Befehl `plot` öffnet ein Graphikfenster namens "figure" mit einer Nummer, in das eine Graphik engebettet werden kann. Falls für die Abszisse und für die Ordinate keine Schranken gesetzt werden, passt sich die Skalierung des Koordinatensystems den Daten automatisch an. Für jedes weitere Bild mus mit dem Befehl `figure` ein neues Graphikfenster geöffnet werden, es erhält eine neue Nummer. Andernfalls wird das alte Bild im Graphikfenster durch das neue Bild überschrieben.

Bekanntlich basiert die Grundstruktur von MATLAB auf einer `n`×`m`-Matrix aus reellen oder komplexen Elementen. Auch Daten werden ja in Matrizen abgelegt. Für ein zweidimensaionales Bild benötigt MATLAB also mindestens zwei Kolonnenvektoren gleicher Länge.

Der Befehl `plot(x,y)` zeichnet den Datensatz `y` in Funktion von Datensatz `x` auf. Wie üblich wird jedem Wert von `x` ein Wert von `y` zugeordnet. `x` sind die Werte der Abszisse und `y` diejenigen auf der Ordinate. Die daraus resultierende Punkte werden mit geraden Linien verbunden (lineare Interpolation). Beide Achsen haben eine lineare Skala.

Mit `plot(x,y,s)` werden im String `s` der Linientyp und die Farbe der Kurve definiert. Der Befehl `plot(x,y,'c+:')` plottet eine rot punktierte Linie, die jedem Datenpunkt ein rotes Plus-Zeichen hat. Der Befehl `plot(y)` enthält nur einen Kolonnenvektor `y` Element von $\mathbb{R}^{n\times 1}$. In diesem Fall generiert MATLAB für die `x`-Achse automatisch Werte, nämlich 1 bis `n`, die Indizes der `n` Kolonnenwerte.

Handelt es sich jedoch bei `y` um einen Vektor mit komplexen Zahlen, so werden beim Befehl `plot(y)` die Realteile auf der `x`-Achse und die Imaginärteile auf der `y`-Achse aufgetragen.

Der Befehl `plot(A)` zeichnet für jede eine Kurve aus `n` Punkten. Die Matrix `A` Element von $\mathbb{R}^{n\times m}$ je eine Kurve aus `n` Punkten. Die `x`-Achse zeigt wieder die Indizes 1 bis `n`. Die LInien werden zur Unterscheidung verschiedene Stile bzw. verschiedene Farben haben.

Beim Befehl `plot(x,A)` wird jede der `m` Kolonnen der Matrix `A` Element von $\mathbb{R}^{n\times m}$ gegen die gemeinsame unabhängige Variable `x` aufgezeichnet. Der Vektor `x` muss die Dimension `n` haben: `x` ist ELement von $\mathbb{R}^{n\times 1}$

Beim Befehl `plot(A,B)` nilden je eine Kolonne der Matrix `A` Element von $\mathbb{R}^{n\times m}$ und der Matrix `B` Element von $\mathbb{R}^{n\times m}$ ein `x`-`y`-Vektorpaar.

Der Befehl `subplot(n,m,p)` unterteilt ein Graphikfenster in `n` Zeilen von je `m` Bildern. Damit können `n`×`m` Bilder in ein Graphikfenster eingebettet werden. `p` ist der Laufindex der `n`×`m` Bilder, wobei die Numerierung zeilenweise von links nach rechts erfolgt. Für jedes neue Bild im Graphikfenster wird der Befehl `subplot` wiederholt, jedesmal mit dem neuen Index `p`. Der eigentliche Befehl `plot` mit seinen Parametern muss dann natürlich auc noch kommen.

Die Befehle `semilogx` und `plot` sind bis auf die Skalierung der `x`-Achse identisch. Beim Befehl `plot` hat die Abszisse eine lineare, beim Befehl `semilogx` aber eine logarithmische Skala (Basis 10). Der Befehl `semilogx(x,y)` entspricht dem Befehl `plot(log10(x), y)`, doch MATLAB gibt bei `semilogx` für `x=0` keine Warnung "log for zero". Der Nullpunkt der `x`-Achse wird unterdrückt, er kann nicht gezeichnet werden. Mit dem Befehl `semilogy` wird die Ordinate mit dem Zehnerlogarithmus skaliert.

Der Befehl `loglog` plottet eine zweidimensionale Graphik in einem doppeltlogarithmischen Koordinatensystem (Basis 10). Der Befehl `loglog(x,y),log10(y)` entspricht dem Befehl `plot(log10(x))`, doch MATLAB gibt bei `loglog` keine Warnung "log of zero", falls `x` oder `y` gleich Null ist. Der Koordinatennullpunkt wird unterdrückt.

Der Befehl `polar(phi,r)` zeichnet die beiden Vektoren `phi` und `r` in ein polares Koordinatensystem. Die Werte des Vektors `phi` sind im Bogenmass angegeben. Die WErte des Vektors `r` entsprechen dem Radius, d.h. dem Abstand zwischen dem Ursprung und dem betreffenden Punkt der Funktion.

Der Befehl `plot(x1,x2,y1,y2)` versieht die Graphik mit zwei Ordinaten. Die linke `y`-Achse bezieht sich auf `y1` in Funktion von `x1` und die rechte `y`-Achse auf `y2` in Funktion von `x2`.

3.1.2 Massstab

Mit dem Befehl `axis([xmin xmax ymin ymax])` lassen sich die Grenzen der `x`- und der `y`-Achse neu definieren. Mit dem Befehl `axis auto` setzt für die Achsen wieder dir ursprünglichen Werte ein. Mit dem Befehl `axis equal` erhalten alle Achsen die gleiche Skalierung. Mit dem Befehl `axis ij` wechselt das Vor-

zeichen der y-Achse. Die positive y-Achse zeigt nun nach unten. Der Befehl `axis xy` macht `axis ij` wieder rückgängig. Der Befehl `axis tight` passt die Achsenlänge exakt dem Bild an. Der Befehl `axis off` schaltet alle axis-Definitionen, die "tick marks" und den Hintergrund aus. Mit dem Befehl `axis on` werden sie wieder aktiviert.

Der Befehl `zoom on` aktiviert in der aktuellen Graphik die Zoom-Funktion, er kann direkt im Command Window eingegeben werden. Mit "clic and drag" wird dann im Bild ein beliebiger Ausschnitt näher herangebracht. Mit der linken Maustaste wird der Graphikausschnitt vergrössert und mit der rechten Maustaste verkleinert. Der Befehl `zoom(factor)` zoomt die Achsen um den für "factor" gewählten Wert. Der Befehl `zoom out` führt die Graphik in ihre default-Fenstergrösse zurück. Der Befehl `zoom xon` bzw. `zoom yon` aktiviert in re aktuellen Graphik die Zoom-Funktion nur für die betreffenden Achsen. Der Befehl `zoom(figurename, option)` versieht die Graphik "figurename" mit einer Zoom-Funktion. Als Option kann eine der oben genannten Zoom-Funktionen gewählt werden.

Der Befehl `grid on` versieht die aktuelle Graphik mit ienem Liniennetz. Mit dem Befehl `grid off` werden die Linien wieder deaktiviert. Der Befehl `box on` umrahmt das aktuelle Bild mit einem Rahmen aus dünnen, schwarzen LInien, der Befehl `box off` entfernt ihn wieder.

Der Befehl `hold on` fixiert das aktuelle Graphikfenster, so dass weitere Funktionen im bereits bestehenden Graphikfenster positioniert werden können. Die ursprünglichen Achseinstellungen bleiben unverändert, selbst dann, wenn die neue Funktion nicht gut in den Rahmen passt. Der Befehl `hold off` führt zum Normalbetrieb zurück, d.h. ein neuer plot-Befehl löscht die aktuelle Funktion und fügt die neue Funktion in das Fenster ein, falls nicht mit dem Befehl `figure` ein weiteres Graphikfenster geöffnet wurde.

Der Befehl `axes('position', [links unten Breite Höhe])` beschreibt im Graphikfenster die POsition der linken unteren Ecke des Bildes und seine Abmessung. Mit Werten zwischen 0 und 1 kann nun die Position und dir Grösse des Bildes festgelegt werden. Das default-Graphikfenster hat eine Breite und eine Höhe 1. Der Befehl `axex` generiert in einem Graphikfenster ein Koordinatensystem, in das mit `plot` ein beliebiges Bild hineingelegt werden kann. Über mehrere `axex`-Definitionen können im Fenster mehrere Bilder erzeugt werden.

3.1.3 Beschriften von Bilder

Der Befehl `legend('st1', 'st2', ...)` versieht im Fenster ein Bild mit einer Legende. Er erhält die einzelnen Strings "st1", "st2", usw. Für jedes Bild in einem Graphikfenster kann ein eigener Titel gewählt werden. Der zu schreibende Text wird in Anführungszeichen genommen. Der Befehl `legend off` entfernt die Legende aus dem aktuellen Bild. Der Befehl `legend('st1', 'st2', ..., position)` positioniert die Legende mit der Angabe einer Position an einen definierten Ort in der Graphik: 0-beste, 1-oben-rechts, 2-oben links, 3-unten-links, 4-unten-rechts, -1-rechts. Die Legende kann verschoben werden, indem die Le-

gende mit der rechten Maustaste angeklickt und an den gewünschten Ort gezogen wird.

Der Befehl `title('text')` fügt einen Titel oberhalb der Graphik hinzu. Ein Text kann hoch (`"^"`) und tiefgestellte `"_"`, kursiv `it"` geschrieben oder griechische Zeichen enthalten.

$$A_1 e^{-\alpha t} \sin \beta t \quad \texttt{'\itA\_\{1\}e\^{}\{\alpha\itt\}sin\beta\itt'} \tag{3.1}$$

Der Befehl `xlabel('text')` schreibt die Zeile "text" unter die Abszisse. Der Befehl `ylabel` ist für die Beschriftung der Ordinate.

Der Befehl `text(x,y,'text')` kann innerhalb des Bildrahmes eine beliebige Textzeile an der Stelle `(x,y)` angebracht werden. `x` und `y` sind in den Koordinaten der Achsen anzugehen. Dagegen verwendet der Befehl `text(x,y,'text,'sc')` die Koordinaten des Graphikfensters, nämlich (0,0) in der unteren linken Ecke und (1,1) ub der oberen rechten Ecke. Geht ein Text über mehrere Zeilen, so kann er in eine Text-Variable geschrieben werden. Ein Text kann hoch- und tiefgestellte, kursiv geschriebene oder griechische Zeichen enthalten.

Mit dem Befehl `gtext('text')` kann mit der Maus im Bild irgendein Text an beliebigen Ort eingefügt werden. Im Graphifenster erscheint der Mauspfeil als Fadenkreuz. An der gewünschten Stelle kann durch Betätigung einer Maustaste der Text eingefügt werden.

3.1.4 Graphiken speichern oder drucken

Der Befehl `print` sendet eine Kopie des aktuellen Graphikfensters an den Drucker. Der Befehl `print filename` speichert eine Kopie des aktuellen Graphikfensters als PostScript-Datei im aktiven Directory unter dem Dateinamen "filename". Mit dem Befehl `print path` kann der genaue Pfad angegeben werden, wo das aktuelle Graphikfenster als POst-Script-Datei abgelegt werden soll. Der Befehl `print[-ddevice][-options]<filenmae>` speichert die aktuelle Graphik im Format des speziell gewählten Druckertreibers und de rzusätzlichen Option im aktiven Directory unter "filename" ab. Der Befehl `help print` zeigt im Command-Window alle möglichen `[-ddevice]` und `[-options]`.

Der Befehl `orient landscape` druckt bei print-Befehlen die Graphikfenster im Querformat. Mit dem Befehl `orient portrait` wird das Graphikfenster im Hochformat grdruckt. Der Befehl `orient tall` setzt das Blattformat auf Hochformat. Zusätzliche wird das Graphikfenster auf das ganze Papierblatt vergrössert bzw. verkleinert. Der Befehl `orient` sagt im Command Window, welche Orientierung momentan aktiv ist.

3.2 Dreidimensionale Graphik

3.2.1 Elementare dreidimensionale Graphik

Der Befehl `plot3(x,y,z)` plottet im dreidimensionalen Raum die Graphen, die durch die Vektoren `x`, `y` und `z` gegeben sind. Die Vektoren müssen alle dieselbe Länge haben. Mit `plot3(X,Y,Z)` zeichnet pro Kolonne einen Graphen. Die Matrizen müssen alle dieselbe Grösse haben. Bei `plot3(x,y,z,'style')` können zusätzlich noch der Linientyp, die Plot Symbole und die Farbe des Graphen geändert werden. `help plot` listet im Command Window eine Answahl von möglichen Liniendefinitionen auf.

Bei `mesh(Z)` entsprechen die Werte der Matrix `Z` Element von $\mathbb{R}^{n\times m}$ den z-Werten des Netzes. Für die `x`- und `y`-Werte verwendet `mesh` die Kolonnen- bzw. die Zeilennummer.

Mit dem Befehl `mesh(X,Y,Z,C)` zeichnet ein Netz un der Vogelperspektive mit `Z` als Funktion von `X` und `Y`. Es handelt sich hierbei um eine FUnktion mit zwei Variablen. `X`, `Y` und `Z` sind Matrizen mit den Werten für die `x`-, `y`- und `z`-Koordinaten. `X` und `Y` können aber auch Vektoren der Länge `m` und `n` sein. Jeder `z`-Koordinate aus der Matrix `Z` werden dann die entsprechenden WErte des `x`- und `y`-Vektors zugewiesen. `C` ist ebenfalls eine Matrix und beinhaltet die Farbskala für die Graphik. Ohne `C` wird `C=Z` gesetzt.

Mit dem Befehl `fill(X,Y,Z,C)` wird in den Farben der Matrix `C` ein dreidimensionales Polygon geplottet. Sind `X`, `Y` und `Z` Vektoren, so wird die Fläche unterhalb des Graphen mit Farbe ausgefüllt. Ist `C` ein Skalar, so wird die Fläche monochrom. Folgende Farben sind möglich: `'r'`, `'g'`, `'b'`, `'c'`, `'m'`, `'y'`, `'w'` und `'k'`. Mit dem Vektor `[rot grün blau]` kann eine neue Farbe gemischt werden. Die Werte von liegen zwischen 0 und 1. Je nach Anteil ergibt sich eine Farbkombination. Ist `C` ein Vektor, so hat er die gleiche Länge wie `X`, `Y` und `Z`. Wird für `C` einer der drei Vektoren gewählt, dann ist die Farbabstufung der momentan aktive `colormap` proportional zur betreffenden Koordinatenachse.

Sind `X`, `Y` und `Z` Matrizen, so zeichnet `fill3` pro Kolonne ein Polygon und füllt es mit der entsprechenden Farbe aus. Die Farbgebung bleibt gleich. Falls `C` ein Zeilenvektor ist, dann hat das Polygon die Schattierung `shading flat`. Für eine Matrix wird sie `shading interp`. `shading` shattiert die Objekt-Oberfläche, `shading flat` berechnet für jede Teilfläche einer Oberfläche, die mit den Befehlen `surf`, `mesh`, `polar`, `fill` oder `fill3` gebildet wurden, die entsprechende Farbabstufung. `shading interp` interpoliert über die Farbabstufung. `shading faceted` entspricht dem `shading flat`. Die 3D Graphik wird jecoh zusätzlich mit schwarzen Linien versehen.

3.2.2 Projektionsarten einer Graphik

Mit `view(az,el)` kann in einem dreidimensionalen Plot der Blickwinkel beliebig eingestellt werden, bzw. die Graphik-Box ist um zwei Achsen drehbar. `az` steht für azimuth und definiert die horizontale Rotation im Grad. Für einen positiven Winkel dreht sich die Graphik entgegen dem Uhrzeigersinn um die z-Achse. `el` beschreibt die Anheben bzw. Senken der Gtraphik in Grad. Bei einem

positiven Winkel befindet sich der Betrachter in der Vogelperspektive, bei negativem Winkel in der Froschperspektive. `view([x y z])` setzt den Blickwinkel in kartesischen Koordinaten. `view(2)` stellt für die 2D Ansicht den vordefinierten Blickwinkel `view(0, 90)` ein. `view(3)` stellt für die 3D Ansicht den vordefinierten Blickwinkel `view(-37.5,30)` ein. `T=view` speichert diew `view` der aktuellen Graphik in der Variable `T` als 4x4-Matrix. `view(T)` weist einer aktuellen Graphik die in der Variable `T` gespeicherte view zu.

`T=viewmtx(az,el)` weist wie bei `T=view(az, el)` der Variable `T` die 4x4-Transformationsmatrix zu. Die Ansicht der aktuellen Graphik wird dabei nicht verändert. Mit `T=viewmtx(az, el, phi)` wird die Graphik durch ein Objektiv betrachtet. Der Linsenwinkel wird in Grad angegeben. `phi=0` Grad definiert die orthogonale Projektion. 10 Grad entspricht einem Teleobjektiv, 25 Grad einem Normalobjektiv und 60 Grad einem Weitwinkelobjektiv. Bei `T=viewmtx(az, el, phi, tp)` wird mit `tp=[xp, yp, zp]` einen Fluchtpunkt gesetzt.

`rotate3d on` aktiviert in der aktuellen Graphik die Maus gesteuerte 3D-Rotation. Die Graphik-Ansicht kann damit beliebig verändert werden. Mit `rotate3d off` wird sie wieder deaktiviert.

3.2.3 Dreidimensionale Graphik beschriften

`zlabel('text')` versieht die z-Achse mit der Aufschrift "text". Mit dem Befehl `colorbar('vert')` erscheint in der aktuellen Graphik eine vertikale Farbskala. In einem 3D-Plot bezieht sie sich auf die Werte der z-Achse. `colorbar('horiz')` zeichnet eine vertikale Farbskala. Im 3D-Plot ist die Farbgebung ebenfalls auf die z-Achse abgestimmt. `colorbar` alleine fügt der Graphik entweder eine vertikale Farbskala hinzu, oder die bestehende Farbskala wird aktualisiert.

3.3 Spezielle Graphen

`fill(x,y,c)` füllt diejenige Fläche mit der Farbe `c` aus, welche von der Geraden, die den Endpunkt von `f(x)` mit dem Anfang verbindet, und der Funktion `y=f(x)` selbst umgeben wird. Ist `c` ein Vektor derselben Länge wie `x` und `y`, dann verwendet MATLAB entweder die im Vektor `c` definierten Farben, oder für `c=x` bzw. `c=y` die Farbpalette der aktuellen `colormap`. Sind in `fill(X,Y,C)` `X` und `Y` Matrizen derselben Grösse, so wird pro Kolonne ein Polygon gezeichnet. `C` kann ein Vektor aber auch eine Matrix sein. Beim Vektor ist die Schattierung der Fläche `shading flat`, bei der Matrix `shading interp`.

Mit dem Befehl `fplot('f',lim)` zeichnet eine beliebige Funktion `f=f(x)` im Bereich `lim=[`x_{min} x_{max}`]`. Mit `lim=[`x_{min} x_{max} y_{min} y_{max}`]` werden zusätzliche Schranken gesetzt. Mit dem Befehl `fplot('f', lim, tol)` mit `tol`<1 definiert die Toleranz des relativen Fehlers. Die voreingestellte Toleranz ist 2e-3 bzw. 0.2%. Mit dem Befehl `fplot('f', lim, N)` berechnet zwischen x_{min} für die Funktion `f=f(x)` `N+1` Punkte. Mit dem Befehl `fplot('f', lim, 'LineSpec')` definiert mit LineSpec den Linien-Typ der Funktion. Alle möglichen "line specifications" werden mit `help plot` aufgelistet.

Mit dem Befehl `hist(x)` zeichnet MATLAB ein Histogramm mit den in `x` gespeicherten Daten. Es ist in 10 gleichmässig verteilte Intervalle unterteilt. Pro Intervall gibt es die Anzahl Elemente an, die es enthält. Wenn `x` eine Matrix ist, plottet `hist` pro Kolonne ein Histogramm. Mit dem Befehl `hist(x,n)` definiert man zusätzlich die Anzahl `n` Intervalle. Mit dem Befehl `hist(x,y)` berechnet MATLAB die Verteilung von `x` bezüglich `y`. `y` ist ein Vektor, deren Elemente in aufsteigender Ordnung aufgelistet sind. Jedes einzelne Element von `y` entspricht einem Zentrum.

Mit dem Befehl `pie(x)` stellt die Daten aus dem Vektor `x` in einem Kuchendiagramm dar. Die Elemente von `x` werden mit der Summe der `x`-Werte dividiert. Damit ist die Grösse von jedem einzelnen Kuchenstück in % gegeben. Mit dem Befehl `pie(x,explode)` zieht mit "explode" die gewünschte Stücke aus dem Kuchen, `explode` ist ein Vektor derselben Länge wie `x`. Seine Elemente haben entweder den Betrag 0, d.h. das entsprechende Stück von `x` verbleibt im Kuchen, ode rden Betrag 1, d.h. das dazugehörende Stück von `x` wird aus dem Kuchen herausgezogen.

Mit dem Befehl `stem(y)` zeichnet eine Verteilung der Daten aus dem Vektor `y`. Jeder Wert aus `y` im Plot mit einem Kreis und einer Linie versehen. Auf der Abszisse wird jedes Element aus `x` fortlaufend eingereiht. Auf der Ordinate kann sein Wert abgelesen werden. Der entsprechende Wert ist mit ienem Kreis gekennzeichnet. Bei `stem(x,y)` entsprechen die Elemente aus dem `x`-Vektor den `x`-Werten und diejenigen aus dem `y`-Vektor den `y`-Werten. Mit dem Befehl `stem(..., 'filled')` malt den Kreis mit der entsprechenden Farbe aus. Mit dem Befehl `stem(..., 'linespec')` kann der Linien-Typ gewählt werden. Alle möglichen "line specifications" werden mit `help plot` aufgelistet.

Mit dem Befehl `contour(Z)` zeichnet einen Konturplot mit den WErten aus der Matrix `Z`. Die Elemente der Matrix `Z` entsprechen den Werten auf der `z`-Achse. Die Kolonnenzahlen ergeben die Koordinaten auf der `x`-Achse und die Zeilenzahlen die WErte auf der `y`-Achse. Die Höhen für die einzelnen Höhenlinien werden automatisch ausgewählt. Mit `contour(x,y,z)` werden die `x`- und `y`-Koordinaten explizit mitgeliefert. `contour(z,N)` und `contour(x,y,z,N)` verwendet für den Konturplot `N` Höhenlinien. `contour(Z,v)` und `contour(x,y,Z,v)` zeichnet all jene Höhenlinien, deren Höhen im Vektor `v` angegeben werden. Mit `contour(Z,[v v])` plottet MATLAB eine einzige Höhenlinie bei der Höhe `v`. Mit `contour(...,'linespec')` kann der Linien-Typ gewählt werden. Alle möglichen "line specifications" werden mit `help plot` aufgelistet...

Listing 3.1: Figure einer Funktion

```
>> figure
>> x=0:0.1:1;
>> y=x.^2;
>> plot(x,y)
```

Listing 3.2: Figure mehrerer Funktionen

```
>> figure
```

```
>> subplot(3,2,1);
>> x=[0:0.2:1];
>> plot(x,sqrt(x))

>>subplot(3,2,2)
>> y=[0 0.4 0.16 0.36 0.64 1];
>> plot(y)

>> subplot(3,2,3)
>> z=[1+i; 2-i; 1.5-0.5i; -0.2+1.9i; -1.3-2i];
>> plot(z, '*')

>>subplot(3,2,4)
>> A=[1 -8 9; 4 -1 5; 9 6 -9];
>> plot(A)

>> subplot(3,2,5)
>> x=[0 1 4];
>> plot(x,A)

>> subplot(3,2,6)
>> B=[7 6 0; 4 3 -8; -4 7 5];
>> plot(A,B)
```

Listing 3.3: Figure mit logarithmischer x-Achse

```
>> x=[0:1:100];
>> semilogx(x,x)
>> hold on
>> semilogx(x,exp(x))
```

Listing 3.4: Figure mit doppeltlogarithmischer Achsen

```
>> x=[0:0.1:100];
>> loglog(1./(1+x.^2))
>> axis([0 100 0.01 10])
```

Listing 3.5: Figure mit Polarkoordinaten

```
>> omega=[0:0.01:pi];
>> r=r=omega.*sin(2*omega);
>> phi=r.*cos(omega);
>> polar(phi,r)
```

Listing 3.6: Figure zweier unabhängigen y-Achsen

```
>> x1=[0:1:5];
>> x2=[0:0.01:5];
>> y1=exp(x1);
>> y2=log(x2);
>> grid on
>> box on
>> axes('position', [0.55 0.5 0.4 0.8])
>> plotyy(x1,y1,x2,y2)
>> legend('exp(x)', 'log(x)')
>> title('Exponential- und logarithmusfunktionen')
>> xlabel('x-Achse')
>> ylabel('y-Achse')
>> gtext('Text mit der Maus einfuegen')
```

Listing 3.7: Figure im dreidimensionalen Raum

```
>> t=[0:0.1:2*pi];
>> plot3(cos(t),sin(t),sin(t));
>> hold on
>> plot3(cos(t),sin(t),sin(2*t),'r:')
>> plot3(cos(t),sin(t),sin(4*t),'k-.')
```

Listing 3.8: Figure im dreidimensionalen Netzoberflaeche

```
>> Z=[0 1 0 0 0; 2 2 6 1 0; 0 4 4 5 6; 1 8 4 7 10; 4 16 8 9 20];
>> x=[-pi./2:0.1:pi./2];
>> y=[-pi./2:0.1:pi./2];
>> [X,Y]=meshgrid(x,x);
>> subplot(2,1,1)

>> mesh(Z)
>> subplot(2,1,2)

>> Z=cos(Y).*cos(X);
>> mesh(x,y,z)
```

Listing 3.9: Figure mit Oberflaechen-Plot

```
>> x=[-pi./2:0.3:pi./2];
>> y=[-pi./2:0.3:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=[cos(Y).*sin(X)];
>> surf(x,y,z)
>> colormap(hsv)
```

Listing 3.10: Figure dreidimensional ausfüllen

```
>> x=[-pi./2:0.2:pi./2];
>> y=[-pi./2:0.2:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=cos(y).*cos(x);
>> Z=cos(Y).*cos(X);
>> subplot(2,2,1)
>> fill3(x,y,z,z)
>> subplot(2,2,2)
>> fill3(x,y,z,'b')
>> subplot(2,2,3)
>> fill3(X,Y,Z,[0.4 0.9 0.5])
>> subplot(2,2,4)
>> fill3(X,Y,Z,X)
```

Listing 3.11: Figure Blickwinkel

```
>> x=[-pi./2:0.3:pi./2];
>> y=[-pi./2:0.3:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=[cos(Y).*sin(X)];
>> subplot(2,2,1)
>> surf(x,y,z)
>> view(3)
>> subplot(2,2,2)
```

```
>> surf(x,y,z)
>> view(25,-10)
>> subplot(2,2,3)
>> surf(x,y,z)
>> view(0,0)
>> subplot(2,2,4)
>> surf(x,y,z)
>> view(-40,-25)
>> colormap(jet)
```

Listing 3.12: Figure Interaktives Rotieren

```
>> x=[-pi./2:0.3:pi./2];
>> y=[-pi./2:0.3:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=[cos(Y).*sin(X)];
>> surf(x,y,z)
>> colormap(jet)
>> rotate3d on
```

Listing 3.13: Figure beschriftung der z-Achse

```
>> x=[0:0.2:4];
>> y=[0:0.2:4];
>> [X,Y]=meshgrid(x,y);
>> surf(X,Y,exp(1./(1+X))+exp(1./(1+Y))+Y)
>> view(30,10)
>> xlabel('x-Achse')
>> ylabel('y-Achse')
>> zlabel('z-Achse: e^(1/(1+x))+e^(1/(1+y))+y')
>> colorbar
```

Listing 3.14: Figure mit einer Variable plotten

```
>> fplot('sin(4*x)*(1/(1+x^2))', [0 15],1e-4)
>> fplot('sign(sin(4*x))*(1/(1+x^2)), [0 15',1e-4)
```

Listing 3.15: Figure Histogramm erzeugen

```
>> x=[9 3 20 4 6 12 15 3 8 5 11 17 7 4 9 6 18 13 7 8];
>> y=[5 9 13];
>> subplot(3,1,1),hist(x)
>> subplot(3,1,2),hist(x,5)
>> subplot(3,1,3),hist(x,y)
```

Listing 3.16: Figure Kuchendiagramm erstellen

```
>> x1=[3.6 0.2 1.5 6.5 2.3 4.1 4.4 0.1];
>> x2=[4 2 9 8 6 3 5 1];
>> subplot(1,2,1)
>> pie(x1,[0 0 0 1 0 0 0 0])
>> subplot(1,2,2)
>> pie(x2,(x2==min(x2))+(x2==max(x2)))
```

Listing 3.17: Figure Daten plotten

```
>> x=[0:0.3:6];
>> subplot(2,1,1)
>> stem(x,sin(x))
>> grid on
>> subplot(2,1,2)
>> stem(x,sin(4.*x).*(1./(1+x.^2)), 'filled', 'r--')
>> grid on
```

Listing 3.18: Figure Oberfläche plotten

```
>> x=[0:0.1:pi];
>> y=[0:0.1:pi];
>> v=[-1:0.1:1];
>> [X,Y]=meshgrid(x,y);
>> Z=[8 0 1 1 6 8; 4 0 2 3 4 6; 2 9 5 6 2 4; 1 2 7 5 9 7; 0 1 2
    2 1 5; 4 2 1 1 2 3];
>> subplot(1,2,1)
>> contour(Z,20)
>> subplot(1,2,2)
>> contour(x,y,cos(X).*sin(Y),v)
>> colorbar
```

Kapitel 4

Programmieren

Dieses Kapitel beinhaltet unter anderem Anweisungen, mit denen der Kontrollfluss in einem M-File gesteuert werden kann. Die Ausführung von Befehlen kann damit z.B, von logischen Bedingungen abhängig gemacht werden. Eine logische Bedingung, die wahr ist, hat in Matlab den Wert 1, eine nicht wahre Bedingung den Wert 0.

Folgende vier Auswahl- und Wiederholungsanweisungen stehen in Matlab zur Verfügung:

a) `if`, zusammen mit `else` und `else if`, führt eine Gruppe von Befehlen dann aus, wenn eine vorgegebene logische Bedingung erfüllt ist.

b) `for` durchläuft eine Schlaufe mit Befehlen für eine feste Anzahl von Wiederholungen.

c) `while` repetiert solange eine von einer logischen Bedingung abhängige Schlaufe, bis die logische Bedingung nicht mehr erfüllt ist.

d) `switch`, zusammen mit `case` und `otherwise`, führt abhängig von einer Entscheidungsvariablen unterschiedliche Gruppen von Befehlen aus.

4.1 Bedingte Befehlsabfolge

Der Befehl `if` führt abhängig von einer logischen Bedingung eine Gruppe von Befehlen aus. Die allgemeine Formel einer `if`-Anweisung lautet

```
 if Ausdruck1
   Anweisungen
elseif Ausdruck2
   Anweisungen
else
   Anweisungen
end
```

Falls der zu `if` gehörende Ausdruck, der eine logische Bedingung beschreibt,

wahr ist, werden die darauf folgenden Befehle abgearbeitet. Dasselbe gilt für `elseif`. Ist keiner der Ausdrücke von `if` oder `elseif` wahr, werden die auf `else` folgenden Befehle ausgeführt.

Es können mehrere `elseif` innerhalb der `if`-Anweisung vorkommen, `elseif` und `else` müssen aber nicht zwingend vorkommen.

Wenn kein Ausdruck wahr ist, und kein `else` vorkommt, fährt MATLAB mit dem auf `end` folgenden Befehl fort. Für die logische Bedingung können die logischen Operatoren `==`, `<`, `>`, `<=`, `>=` oder `~=` verwendet werden.

Listing 4.1: Übung M2b) Teil 3

```
if x>0 & x<1
    disp('x ist groesser 0 aber kleiner 1.')
    xneu=x+1;
    x=xneu;
elseif x>=1 & x<10
    disp('x liegt zwischen 1 und 10.')
    xneu=x-10
    x=xneu;
elseif x>10
    disp('x ist groesser als 10.')
elseif x==0
    disp('x ist 0.')
else
    disp('x ist negativ.')
end
```

Der Befehl `for` ist eine Schlaufe mit einer festen Anzahl Durchläufe. Die allgemeine Form lautet

```
for Index=Start:Inkrement:Ende
  statements
end
```

Index bezeichnet die Variable in der for-Schlaufe. Wird siegestartet,so wird demIndex der Startwert zugewiesen. Bei jedem Durchlauf werden die Anweisungen in der for-Schlaufe ausgeführt, und der Index erhöht sich jeweils um den Wert des Inkrements bis der Endwert erreicht wird. Das Inkrement kann auch negativ sein. Das vordefinierte Inkrement hat den Wert 1. Mit dem Befehl `break` kann aus einer Schlaufe vorzeitig ausgestiegen werden.

Listing 4.2: For-Schleife

```
for t=1:-0.2:0.2
    y(round(10*t/2))=exp(1/t);
end
%Angeben in Command Window: bspfor;y, dann werden Zahlen
    generiert
```

Listing 4.3: For-Schleife

```
n=1;m=3;
```

```
for i=1:n
    for j=1:m
        A(i,j)=i+j
    end
end
%Angeben inc Command Window: bspfor2;A, dann werden Zahlen
    generiert
```

Der Befehl `while` durchläuft eine Schlaufe in Abhängigkeit einer logischen Bedingung. Die allgemeine Form einer while-Schlaufe lautet

```
while Ausdruck
  Anweisungen
end
```

Solange die logische Bedingung wahr (>0) ist, wird die Schlaufe durchlaufen, d.h. die Befehle werden ausgeführt. Für die logische Bedingung können die logischen Operatoren ==, <, >, <=, >=, ∼= verwendet werden. Mit dem Befehl `break` kann vorzeitig aus einer Schlaufe ausgestiegen werden.

Listing 4.4: While

```
while n<=10
    y(n)=1+n-n^2+0.1*n^3;
end
%Incd Command Window: n=0;bspwhile; x=0:10:plot(x,y)
```

Der Befehl `switch` ist abhängig vom Wert, den ein festgelegter Ausdruck annimmt. Dieser Befehl führt unterschiedliche Teile eines M-Files aus. Die allgemeine Form einer switch-Bedingung lautet

```
switch Ausdruck
  case Wert1
     Anweisungen
  case {Wert2, Wert3}
     Anweisungen
  ...
  otherwise
     Anweisungen
end
```

`switch` vergleicht den Wert von Ausdruck mit den Werten in den verschiedenen `case`. Das erste `case`, von dem einer der Werte mit dem Wert von Ausdruck übereinstimmt, wird ausgeführt. Ein `case` kann mehrere Werte haben. Es wird maximal ein `case` ausgeführt. Stimmt keiner der WErte mit dem Wert von Ausdruck überein, so werden die unter `otherwise` aufgelisteten Befehle abgearbeitet, falls `otherwise` existiert.

Listing 4.5: Switch

```
x=1;
switch x
    case 1
        disp('x hat den Betrag 1')
```

```
    case {2,3,4}
        disp('x hat den Betrag 2, 3 oder 4')
    otherwise
        disp('x hat weder den Betrag 1, 2, 3 noch 4')
end
%In Command Window bspwswitch
```

Listing 4.6: Switch

```
Antwort=input('Moechten Sie einen Test machen? ja/nein ','s');
disp(' ')
switch Antwort;
    case 'ja'
        disp('wie vorbildlich!')
    case 'nein'
        disp('wie schade!')
    otherwise
        disp('Ein "jein" kenne ich nicht!')
end
%In Command Window: bspswitch2 => Inputangaben
```

4.2 MATLAB Funktionen

Mit dem Befehl `function` wird eine neue Funktion in MATLAB definiert. Eine Funktion in MATLAB ist ein spezielles M-File, das durch den Befehl `function` in der ersten Zeile gekennzeichnet ist. Im Gegensatz zu Skript M-Files müssen bei Funktionen Variablen explizit übergeben und zurückgegeben werden. Skript M-Files arbeiten mit den Variablen aus dem MATLAB-Workspace. Funktionen dagegen haben ihren lokalen Workspace.

In einer Funktion verwendete Variablen sind im MATLAB-Workspace nicht definiert und umgekehrt. Eine Funktion besteht aus der Kopfzeile, dem help-Text und den Befehlszeilen.

```
function y = fname(x)
```

Die Kopfzeile weist ein M-File als MATLAB-Funktion aus und legt die Syntax der Funktion fest. Die hier verwendeten Variablennamen müssen nicht identisch sein mit den später beim Funktionsaufruf verwendeten Variablennamen.

Die Kopfzeile muss mit dem Befehl `function` beginnen. Nach einem Leerzeichen werden die Ausgabeargumente definiert. Auf das Gleichheitszeichen folgt der Name der MATLAB-Funktion. Sie trägt bis auf die File-Erweiterung `‘‘.m’’` (die weggelassen wird) denselben Namen wie das M-File, in dem sie abgespeichert wird. Direkt nach dem Namen werden in einem Klammerpaar die Eingabeargumente definiert.

Es ist von Vorteil, eine Funktion mit einem ausführlichen Kommentar zu versehen. Damit kann leichter nachvollzogen werden, welchen Zweck die Funktion hat.

Auf die Kopfzeile folgt direkt der help-Text der Funktion, der durch %-Zeichen am Zeilenanfang als Kommentar gekennzeichnet ist. Wird in MATLAB `help fname` eingegeben, so wird dieser help-Text wiedergegeben. Die erste Zeile des help-Texts sollte den Funktionsnamen enthalten und mit einigen charakteristischen Begriffen den Zweck der Funktion umreisen. Der Befehl `lookfor` referenziert diese erste Zeile bei der Stichwortsuche in MATLAB.

Es erscheinen nur diejenigen Kommentare als help-Text, die zwischen der Kopfzeile und der ersten Befehlszeile liegen.

Die Funktionen kann weiter Unterfunktionen, Schlaufen, Kalkulationen, Wertzuweisungen, Kommentare und Leerzeilen beinhalten oder andere Funktionen aufrufen.

Mit dem Befehl `return` kann eine Funktion vorzeitig verlassen werden, z.B. wenn eine vorgegebene Abbruchbedingung in der Funktion eingetreten ist.

Listing 4.7: botta

```
function [V,AO,AM] = botta(r,s1,s2)
%Botta Schief abgeschnittenen Zylinder berechnen
% Die Funktion botta(r,s1,s2) berechnet das Volumen V,
% die Oberflaeche AO und die Mantelflaeche AM eines schief
% abgeschnittenen Zylinders.
% r ist der Radius, s1 die laengste Mantellinie
% und s2 die kuerzeste.

V=pi*r^2*(s1+s2)/2;
AO=pi*r*(s1+s2+r+sqrt(r^2+(s1+s2)^2/4));
AM=pi*r*(s1+s2);

%In Command Window: help botta
%In Command Window: [V,AO,AM]=botta(4,5,2) angeben
```

Der Befehl `nargin` bestimmt die Anzahl der Eingabeargumente einer Funktion. In einer Funktion eruiert der Befehl `nargin`, wieviele Arguemntedie Funktion erhalten hat. Je nach Anzahl der Argumente können dann z.B. unter Verwendung von if-Anweisungen unterschiedliche Aufgaben zw. Berechnungen durchgeführt werden.

`nargin('fname')` gibt die Anzahl der definierten Eingabeargumente der Funktion “fname”.

Listing 4.8: nargin

```
function y = bspnargin(a,b)
    if nargin<1
        error('Keine Eingaenge')
    elseif nargin==1
        y=a^2-4;
    elseif nargin==2
        y=a^2+4*b^2-4;
    end
end
%In Command Window: bspnargin(3) => 5, bspnargin(3,2) => 21
```

Der Befehl `nargout` bestimmt die Anzahl Ausgabeargumente einer Funktion. In einer Funktion eruiert der Befehl `nargout`, wieviele Werte die Funktion auszugeben hat. Je nach Anzahl der verlangten Ausgabeargumente können dann z.B. mittels einer if-Anweisung unterschiedliche Befehle zur Ausführung gelangen.

`nargout('fname')` gibt die Anzahl der definierten Ausgabeargumente der Funktion "fname".

Listing 4.9: nargout

```
function [y,z] = bspnargout(a,b)
    if nargin<1
        error('Keine Eingaenge')
    elseif nargin==1
        y=a^2-4;
    elseif nargin==2
        y=a^2+4*b^2-4;
        if nargout==2
            z=a^2+b-16;
        end
    end
end
%In Command Window: [y,z]=bspnargout(3,2) => 21
```

Der Befehl `global` definiert eine globale Variable. Grundsätzlich sind ie in einer Funktion verwendeten Variablen nur lokal definiert. Aus einer anderen Funktion oder aus dem Workspace kann nicht darauf zugegriffen werden. Wie auch die Funktion selber nicht auf Variablen des workspace oder anderer Funktionen zugreifen kann.

Wenn aber in einer Funktion eine Variable als globale Variable definiert wird, ist sie für alle anderen Funktionen und für den workspace zugänglich, sofern soe dort ebenfalls für global erklärt wurde.

Listing 4.10: nargout

```
function y = bspglobal(a)
    global b
    if nargin<1
        error('Keine Eingaenge')
    elseif nargin==1
        y=b^2+a^2-4;
        b=b+1;
    end
end
%In Command Window: global b; b; b=3; bspglobal(b); b
```

4.3 Befehle auswerten und ausführen

Der Befehl `eval` führt einen in einem String enthaltenen MATLAB-Befehl aus. `eval('Befehl')` interpretiert den String als MATLAB-Befehl und behandelt ihn dementsprechend. Mit `eval('Befehl1', 'Befehl2')` besteht die Möglichkeit, Fehlermeldungen zu unterdrücken, indem in zwei Strings für eine Berechnung

zwei unterschiedliche Befehle eingegeben werden.

Falls einer der beiden Befehle einen Fehler erzeugt, wird ohne Fehlermeldung der andere ausgeführt. Im ersten String kann z.B. eine neue Berechnung ausprobiert werden, unnd im zweiten der alte Befehl eingegeben werden. Damit ist eine Ausgabe garantiert.

Der Befehl `feval` führt eine in einem String enthaltene MATLAB-Funktion aus. `feval('Funktion,x1,...,xn')` interpretiert den String als MATLAB-Funktion und behandelt ihn dementsprechend. `feval` berechnet die genannte Funktion, die gewöhnlich in einem separaten M-File definiert wird, an den Stellen `x1` bis `xn`.

Der Befehl `run` führt ein M-File aus, das nicht in einem Directory des aktuellen Pfads gespeichert ist. Normalerweise wird der Name eines M-Filies im Command Window eingetippt, um es auszuführen. Dies funktioniert jedoch nur, wenn das das M-File enthaltene Directory im aktuellen Pfad vermerkt ist.

Mit dem Befehl Der Befehl `run` kann nun ein M-File ausgeführt werden, das sich nicht im aktuellen Pfad befindet. Mit `run Dateinamen` wird das entsprechende M-File gestartet. Wird im Dateinamen der komplette Pfadname angegeben, so wechselt `run` vom momentan aktiven Directory zu dem Directory, in dem sich das betreffende M-File befindet, führt es aus, und wechselt wieder ins ursprünglich aktive Directory zurück.

Kapitel 5

M1 Übung: Matlab und seine Toolboxen

Drehbuch

- Administration Unterricht, Leistungsbewertung, Unterlagen, Homepage.
- Matlab und Toolboxen: eine Einführung.
- Erster Start von MATLAB, Fensterorganisation und Erklärungen.
- Gastreferenten aus der Praxis zeigen den Umgang mit Matlab

Lernziele Die Studierenden...

- haben Klarheit über die Unterrichtsorganisation und die Leistungsbeurteilung.
- lernen Matlab kennen mit den wichtigsten Fenstern.
- kennen die wichtigsten Toolboxen für ihr Studium mit einem Beispiel.
- und wissen, wie man überprüfen kann, welche installiert sind

5.1 Getting started in Matlab

1. Sie definieren eine Variable a mit dem Wert 5 $\Longrightarrow$ `a=5`
2. Sie definieren einen Vektor mit 100 Werten $\Longrightarrow$ `linspace(0,1)`
3. Sie löschen den Bildschirm $\Longrightarrow$ `clc`

5.2 Die Matlab Homepage

Matlab besitzt eine gute Homepage. Sie finden diese unter folgendem Link: Unter der Mathworks Community finden Sie viele Antworten auf Ihre Probleme.

Aufgabe: Suchen Sie in der Community Hilfe für den Befehl `rotate surface`.

Wählen Sie das erste Beispiel aus. Dieses Beispiel kann direkt im Webbrowser (ohne Benutzung von Matlab) ausgeführt werden.

Probieren Sie dies aus: ⇒ Try this example ⇒ Try it in your browser ändern Sie den Code auf `surf(peaks(100),'Parent',t)` ab und führen Sie diesen aus. Waren Sie erfolgreich? Nein

5.3 Was ist eigentlich Simulink?

Simulink basiert auf Matlab und ist eine zusätzliche Umgebung. Während Matlab ein textbasiertes Programm ist, ist Simulink grafisch orientiert. Man kann komplexe Gleichungen einfach grafisch darstellen.

1. Die Programmierung in Matlab erfolgt: mittels Text.
2. Die Programmierung in Simulink ist: grafisch aufgebaut.
3. Simulationen in Simulink: sind immer in Funktion der Zeit.
4. Komplexe differentialgleichungen können: einfach grafisch dargestellt werden.
5. Programmblöcke werden mittels: grafischer Linien miteinander verbunden.

5.4 Toolboxen in Matlab und Simulink

1. Eigentlich ist Matlab ein numerisches Programm. Die Symbolic Math Toolbox bildet eine Ausnahme. Diese Toolbox wurde vor einigen Jahren von einer externen Firma eingekauft und integriert.
2. Simscape erlaubt die Simulation von physikalischen Systemen.
3. Die Signal Processing Toolbox ist ein Zusatz in Matlab und erlaubt die Verarbeitung von z.B. Audiosignalen.
4. Die Control System Toolbox war eine der ersten Toolboxen in Matlab. Sie ist extrem mächtig. Sie wird häufig in der Regelungstechnik verwendet und beihaltet Auslegungen von z.B. einem PID-Regler.
5. Matlab kennen Sie bereits: dies ist ein leistungsstarker texbasierter Editor.
6. Simulink ist dagegen grafisch orientiert.

5.5 Model Based Design

Ein Design wird folgendermassen umgesetzt:

1. Aufbau des Modells: Aufbau des Modells (Physik) vor allem in Simulink z.T. auch in Matlab (embedded Functions) Regelungen werden direkt in Simulink gezeichnet

2. Simulation des Modells: Das System wird in Simulink simuliert. Für die Physik wird häufig Simscape verwendet (z.B. Umrichter, Motoren, Kondensatoren, ...)

3. Generation von C-Code Falls die Simulation erfolgreich ist, wird direkt aus dem Simulink Modell der C-Code compiliert und auf die Target-hardware heruntergeladen. Dies geht übrigens auch für kleine Prozessoren und DSPs.

4. Optional: Simulation des Systems mit einem RTS (real time simulator) Es gibt Leistungsrechner (dSpace). Mit denen kann die programmierte Hardware in Echtzeit getestet werden. Auf diesen Rechnern wird die "Physik" implementiert... und auch diese Rechner werden in Simulink programmiert. Dies ist "State of the art" für die Automobilindustrie, Flugzeugindustrie und Traction.

5. Testen und watchen Die Software wird auf der Target-hardware und im Feld getestet.

Kapitel 6

M2 Übung: Einführung in Matlab

Drehbuch

1. Kap. 1 - Kap. 5 vom Buch „Die nicht zu kurze Kurzeinführung in MATLAB“ selbstständig durcharbeiten.
2. Die ausgewiesenen Kurzübungen sind abzugeben.

Lernziele Die Studierenden...

1. lernen autodidaktisch den Einstig in Matlab.
2. wissen, wie man die Hilfe zu einer Funktion findet.
3. kennen das Command Window.
4. verstehen den Hauptunterschied zwischen einer normalen Variable und einer Matrix.
5. können einfache grafische Darstellungen erstellen.

6.1 M2a) übung zu Kapitel 3

(a) Studieren Sie die Befehle linspace und logspace. Wo sind die Unterschiede?

$\Longrightarrow$ `linspace(xstart, xend)` erzeugt einen Vektor zwischen `xstart` und `xend`, der in 99 gleihe Intervalle unterteilt wird. Der Vektor besteht somit aus 100 linear, gleichmässig verteilten Punkten.
$\Longrightarrow$ `linspace(xstart, xend, n)` erzeugt einen Vektor zwischen `xstart` und `xend`. Der Vektor besteht somit aus `n` linear, gleichmässig verteilten Punkten.

- If `n` is 1, linspace returns `xend`.
- If `n` is zero or negative, linspace returns an empty 1-by-0 matrix.
- If `n` is not an integer, linspace rounds down and returns `floor(n)` points.

$\Longrightarrow$ `logspace(xstart, xend)` erzeugt einen Vektor. Der Vektor besteht somit aus 50 logarithmischen verteilten Punkten zwischen $10^{\texttt{xstart}}$ und $10^{\texttt{xend}}$.
$\Longrightarrow$ `logspace(xstart, xend, n)` erzeugt `n` Punkte zwischen $10^{\texttt{xstart}}$ und $10^{\texttt{xend}}$
$\Longrightarrow$ `logspace(xstart, pi, n)` erzeugt `n` Punkte zwischen $10^{\texttt{xstart}}$ und `pi`.

- If `n` is 1, logspace returns $10^{\texttt{xend}}$.
- If `n` is zero or negative, logspace returns an empty row vector.
- If `n` is not an integer, logspace rounds `n` down and returns `floor(n)` points.

(b) Erstellen Sie einen Vektor a mit 20 Elementen von 0 bis 19 (ohne linspace).

$\Longrightarrow$ `a=(0:1:19)` oder `a=[0:1:19]`

(c) Erstellen Sie einen Vektor b mit 20 Elementen von 0 bis 2pi.

$\Longrightarrow$ `b=linspace(0,2*pi,20)` `%include 2*pi`

(d) Erstellen Sie eine 2x20 Matrix A mit den Vektoren a und b.

$\Longrightarrow$ `A=[a;b]`

(e) Setzen Sie die beiden Elemente in der letzten Spalte auf 0.

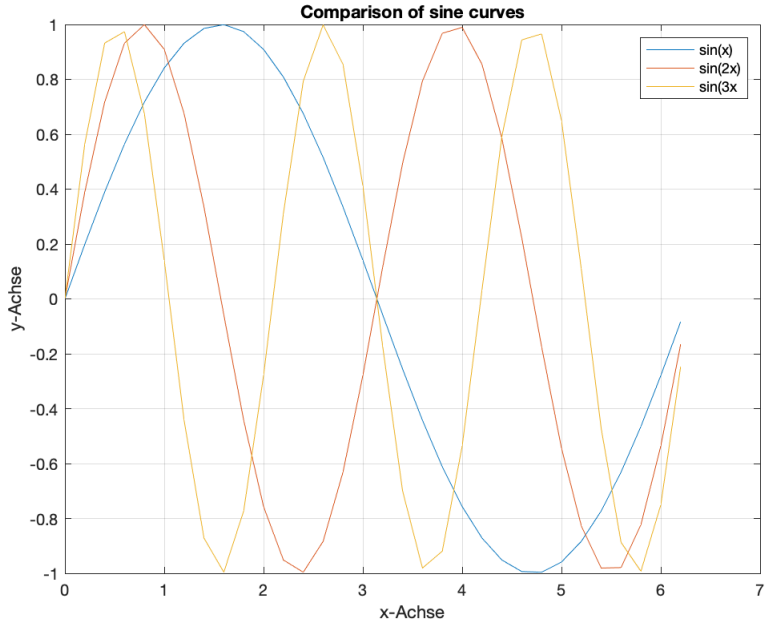
$\Longrightarrow$ `A(1,20)=0`
$\Longrightarrow$ `A(2,20)=0`

6.2 M2b-c) übung zu Kapitel 4 und 5

a) Stellen Sie `sin(x)`, `sin(2x)` und `sin(3x)` gemeinsam in einer Figur dar. Beschriften Sie diese Figur ausführlich (Achsen) und erstellen Sie eine Legende.

Listing 6.1: übung M2b) Teil 1

```
x=(0:0.2:2*pi);
plot(x,sin(x))
grid off, hold on
plot(x,sin(2*x))
plot(x,sin(3*x))
legend('sin(x)', 'sin(2x)', 'sin(3x')
title('Comparison of sine curves')
xlabel('x-Achse')
ylabel('y-Achse')
grid on
```



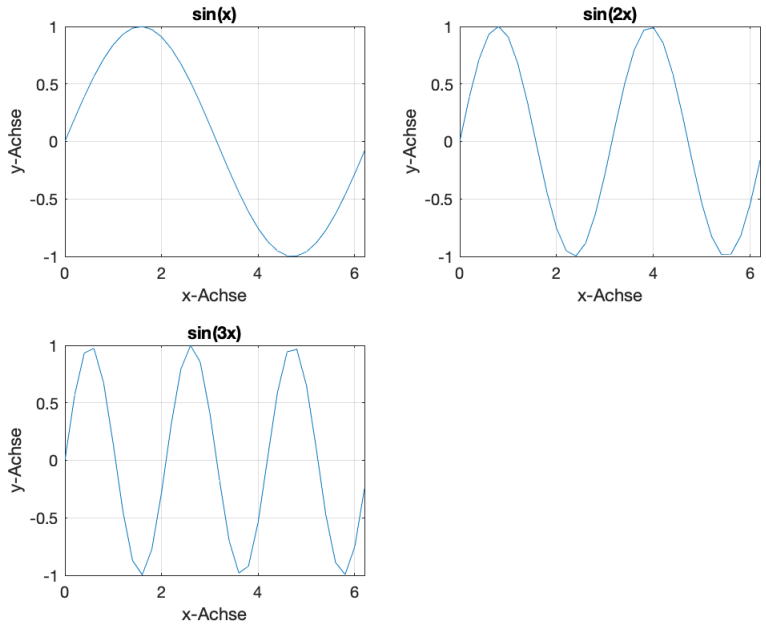
b) Stellen Sie diese Funktionen in einer neün Figure alle untereinander dar.

Listing 6.2: übung M2b) Teil 2

```
subplot(2,2,1)
x=(0:0.2:2*pi);
plot(x,sin(x))
title('sin(x)')
xlabel('x-Achse')
ylabel('y-Achse')
grid on

subplot(2,2,2)
plot(x,sin(2*x))
title('sin(2x)')
xlabel('x-Achse')
ylabel('y-Achse')
grid on

subplot(2,2,3)
plot(x,sin(3*x))
title('sin(3x)')
xlabel('x-Achse')
ylabel('y-Achse')
grid on
```



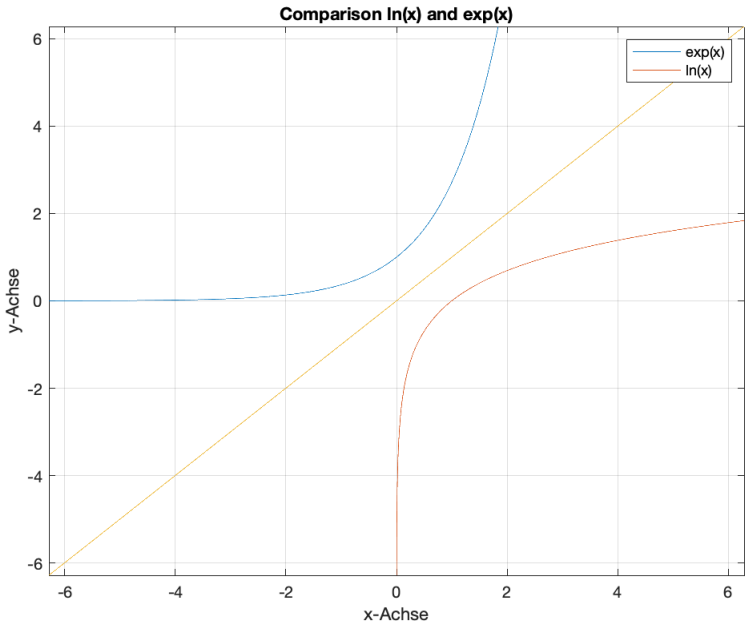
c) Stellen Sie e^x und `ln(x)` in einer neün Figur dar.

Listing 6.3: übung M2b) Teil 3

```
% Definition zweier Intervalle, eine fuer die
    Exponentialfunktion und andere fuer die Logarithmusfunktion.
x=(-2*pi:0.002:2*pi);
y=(0.00001:0.002:2*pi);
% Plotten der Exponentialfunktion
plot(x,exp(x))

hold on
% Plotten der Logarithmusfunktion
plot(y,log(y))
% Plotten der ersten Winkelhalbierende
plot(x,x)
% Gleiche Achsenverteilung
axis([-2*pi 2*pi -2*pi 2*pi])

grid on
% Legende
legend('exp(x)', 'ln(x)')
xlabel('x-Achse')
ylabel('y-Achse')
title('Comparison ln(x) and exp(x)')
```


Comparison ln(x) and exp(x)
exp(x)
ln(x)
y-Achse
x-Achse
6
4
2
0
-2
-4
-6
-6
-4
-2
0
2
4
6

Kapitel 7

M3 Übung: Vektoren und Arrays, Funktionen

Drehbuch

1. Fragen zum Buch.
2. Lernseqünz: Berechnung mittels Vektorisierung (geleitet).
3. übung 1: Wurfparabel (geleitet).
4. Kap. 6 (Funktionen und Fehlerhandling) vom Buch „Die nicht zu kurze Kurzeinführung in MATLAB“ selbstständig durcharbeiten.

Lernziele Die Studierenden...

1. festigen den Umgang mit Matlab.
2. erkennen, das viele Schleifen in Matlab nicht ausgeführt werden müssen.
3. verstehen die Mächtigkeit von Vektoroperationen.
4. und ebenso Darstellungsmöglichkeiten von ganzen Matrizen.
5. wissen, was ein m-File ist und können eine Funktion erstellen.

7.1 Wurfparabel

Die Wurfparabel stellt den Verlauf eines Balles dar, der mit einer gewissen Anfangsgeschwindigkeit und einem gewissen Abschusswinkel geworfen wird. Der Weg, den dieser Ball beschreibt, kann folgendermassen ausgedrückt werden:

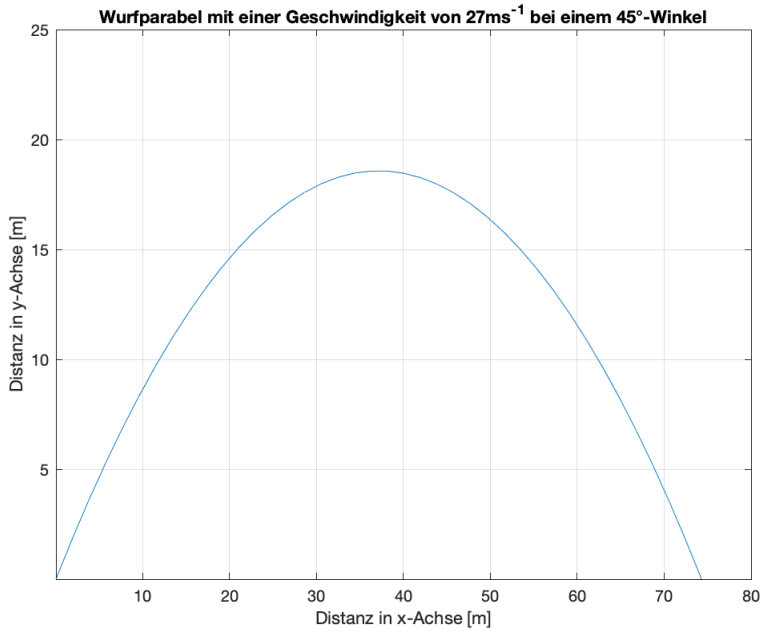
$$sx\left(t\right) = \left|\overrightarrow{v}\right| \cdot \cos\left(\alpha\right) \cdot t \tag{7.1}$$

$$sy\left(t\right) = \left|\overrightarrow{v}\right| \cdot \sin\left(\alpha\right) \cdot t - \frac{g}{2} \cdot t^2 \tag{7.2}$$

$sx(t)$: x-Komponente der Wurfbahn [m].
$sy(t)$: y-Komponente der Wurfbahn [m].

$\left|\overrightarrow{v}\right|$: Betrag der Wurfgeschwindigkeit [m/s].
α: Abschusswinkel [grad, rad].
t: Zeit [s].
$g = 9.81$: Gravitationskonstante [m/s2]

Programmieren Sie in MATLAB ein M-File, welches die Wurfparabel darstellt. Die Anzahl der dargestellten Parabeln soll im M-File verändert werden können. Bei welchem Abschusswinkel wird die Wurfweite maximal?



Listing 7.1: Wurfparabel für 60 Grad

```
clear variables, close all

% Zeit zwischen 0 und 40 sekunden
t = (0:0.1:40);

% Gravitation
g =  9.81;

% Abschusswinkel
alpha = 45;
angle = alpha .* (pi/180);

% Definition der Wurfgeschwindigkeit
v = 27;

% Anfangszustand der Masse
sx0 = 0;
sy0 = 0;

% Komponenten der Beschleunigung
ax = 0;
```

```
ay = -g;

% Komponenten der Geschwindigkeit
vx = v*cos(angle);
vy = v*sin(angle)-g.*t;

% Komponente der Bewegungszustand
sx=v*cos(angle).*t+sx0;
sy=v*sin(angle).*t-g/2*t.^2+sy0;

% Wurfparabel
figure;
plot(sx,sy); grid on
xlabel('Distanz in x-Achse [m]');
ylabel('Distanz in y-Achse [m]');
title(['Wurfparabel mit einer Geschwindigkeit von ',num2str(v),'
    ms^-^1 bei einem ',num2str(alpha),'-Winkel']);
axis([0.01 80 0.01 25]);
```

7.2 M3a) übung zu Funktionen

öffnen Sie die Funktion `linspace` mit dem Befehl `edit linspace` und analysieren Sie diese Funktion von Mathworks.

Listing 7.2: übung M3a)

```
function y = linspace(d1, d2, n)
%LINSPACE Linearly spaced vector.
%   LINSPACE(X1, X2) generates a row vector of 100 linearly
%   equally spaced points between X1 and X2.
%
%   LINSPACE(X1, X2, N) generates N points between X1 and X2.
%   For N = 1, LINSPACE returns X2.
%
%   Class support for inputs X1,X2:
%      float: double, single
%
%   See also LOGSPACE, COLON.

%   Copyright 1984-2016 The MathWorks, Inc.

if nargin == 2
    n = 100;
else
    n = floor(double(n));
end
if ~isscalar(d1) || ~isscalar(d2) || ~isscalar(n)
    error(message('MATLAB:linspace:scalarInputs'));
end
n1 = n-1;
c = (d2 - d1).*(n1-1); %check intermediate value for appropriate
     treatment
if isinf(c)
    if isinf(d2 - d1) %opposite signs overflow
        y = d1 + (d2./n1).*(0:n1) - (d1./n1).*(0:n1);
    else
        y = d1 + (0:n1).*((d2 - d1)./n1);
    end
else
```

```
33      y = d1 + (0:n1).*(d2 - d1)./n1;
34  end
35  if ~isempty(y)
36      if d1 == d2
37          y(:) = d1;
38      else
39          y(1) = d1;
40          y(end) = d2;
41      end
42  end
```

Beantworten Sie folgende Fragen

a) Wie lang ist die Länge eines Standardvektors, wenn keine Länge definiert wird?
$\Longrightarrow$ Generierung eines Vektors mit 100 linear gleichabständigen Punkten.

b) Mit `help linspace` erscheint die Hilfe dieser Funktion im Command Window

- Wo befindet sich dieser Text in der Datei?
  $\Longrightarrow$ Der angezeigte Text befindet sich im Command Window.
- Ab welcher Zeile wird der Hilfetext nicht mehr ausgegeben? Wie wird dies abgegrenzt?
  $\Longrightarrow$ Eine Zeile vor der Zeile des Copyrights wird nichts mehr angezeigt. Durch `edit help` kann man dies ändern.

7.3 M3b) übung zu Kapitel 6

Schreiben Sie eine Funktion welche die zwei Zahlen `a` und `b` addiert (`add`), subtrahiert (`sub`), multipliziert (`mult`), dividiert (`div`) und potenziert (`pow`). Testen Sie diese Funktion gründlich.

Listing 7.3: übung M3a)

```
1   % Diese Funktion berechnet 5 verschiedene mathematische
        Operationen
2   % durch die Angabe zweier reellen Zahlen.
3   % Die Name der Funktion entspricht die Name der Datei
4
5   function [add, sub, mult, div, pow] = ubungm3b(a,b)
6       add = a+b;
7       sub = a-b;
8       mult = a*b;
9       div = a/b;
10      pow = a^b;
11  end
12  %In Command Window angeben: [add,sub,mult,div,pow] = ubungm3b
        (3,4)
13  %=> add=7, sub=-1, mult=12, div=0.7500, pow=81
```

7.4 M3b) übung publish-pdf zu Kapitel 6

Testen Sie Ihre Funktion nun selber, indem Sie die Testfunktion für die übung M3b) (oberhalb dieses Textblockes) herunterladen und mittels der Publish Funk-

tion ausführen.

Korrigieren Sie Ihre Funktion dual(a,b) so lange, bis keine Fehlermeldung mehr auftritt. Einzig bei der Division durch Null kann es möglich sein, dass Fehler auftreten werden.

Speichern auf Moodle folgende zwei Dateien ab:

$a)$ Ein pdf von Ihrer Funktion dual(a,b)

$b)$ Das Resultat der Publish Funktion (auch als pdf). Der Code des Testfiles soll nicht eingebunden werden