

# 1 VEKTOREN UND MATRIZEN

## 1.1 Matrix-Manipulationen

### 1.1.1 Elementare Matrizen

Der Befehl `zeros` setzt alle Koeffizienten einer Matrix gleich Null. Der Befehl `zeros(n)` bildet eine  $n \times n$  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  $n \times n$ -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  $n \times m$  Elementen. Die Matrizen  $X$  und  $Y$  werden für das Plotten von Funktionen mit zwei Variablen 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 Variablen und für dreidimensionale Volumen Graphiken verwendet.

### 1.1.2 Informationen über die Dimension

Der Befehl `size(A)` informiert über die Dimension der Matrix  $A$ . Die erste Zahl des zweizeiligen Ausgabevektors steht für die Anzahl 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`.

### 1.1.3 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.

## 1.2 Lineare Algebra

### 1.2.1 Grundoperationen

Die linke Division  $A \setminus B$  von  $A \setminus B$  entspricht der Multiplikation der Inversen der regulären Matrix (regulär:  $\det(A) \neq 0$ )  $A$  mit der Matrix  $B$ , d.h.  $\text{inv}(A) * B$ . Dasselbe gilt für  $A \setminus B$  und  $B * \text{inv}(A)$ .  $A \setminus b$  ist die Lösung des linearen Gleichungssystems  $Ax=b$ , falls  $A \in \mathbb{R}^{n \times n}$  und  $\text{Rang}(A)=n$ . 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.

$$e = \sum_{i=1}^n \left( b_i - \sum_{j=1}^m a_{i,j} x_j \right)^2 = \|b - Ax\|_2^2 \quad (1.1)$$

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 singular ist, erscheint im Command Window eine Fehlermeldung. Die Definition der inversen Matrix von  $A$  lautet

$$X = A^{-1} = \frac{1}{\det(A)} [(-1)^{i+j} \det(A_{ji})] \quad (1.2)$$

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  $\sigma$  von  $A$  lassen sich mit der Euklidischen Norm berechnen. Sie sind die positiven Quadratwurzeln der grössten Eigenwerte der Hermiteischen Matrix  $A^H A$ .

$$\sigma_i(A) = \sqrt{\lambda_i(A^H A)} \quad (1.3)$$

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 singulären 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 Spalten 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  $\text{sum}(\text{abs}(a) \cdot 2)^{(1/2)}$

$$\|a\|_2 = \sqrt{\sum_k |a_k|^2} \quad (1.4)$$

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

$$\|a\|_1 = \sum_k |a_k| \quad (1.5)$$

Der Befehl `norm(a, inf)` ermittelt die  $\infty$ -Norm von  $a$ . Sie steht für das betragsmässig grösste Zeilenelement von  $a$ ,  $\max(\text{abs}(a))$ .

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

Der Befehl `norm(A,1)` berechnet die betragsmässig grösste Summe der absoluten Beträge von jedem Spaltenelement

von  $A$ ,  $\max(\text{sum}(\text{abs}(A)))$ .

Der Befehl `norm(A, inf)` ermittelt die betragsmässig grösste Summe der absoluten Beträge von jedem Zeilenelement von  $A$ ,  $\max(\text{sum}(\text{abs}(A')))$ .

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 Spalten, gilt für die Determinante von  $A$

$$\det(A) = \sum_{k=1}^n (-1)^{j+k} a_{kj} \det(A_{kj}) \quad (1.6)$$

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

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

## 1.2.2 Eigenwerte und Singulärwerte

Die Eigenwerte  $\lambda$  berechnet man durch den Befehl `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  $\sigma$  der Matrix  $A$  die positiven Quadratwurzeln der grössten Eigenwerte der Hermiteischen 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.

## 2 GRAPHIK

### 2.1 Zweidimensionale Graphik

#### 2.1.1 Elementare zweidimensionale Graphik

Der Befehl `plot` öffnet ein Graphikfenster namens `figure` mit einer Nummer, in das eine Graphik eingebettet 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 muss 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  $n \times m$ -Matrix aus reellen oder komplexen Elementen. Auch Daten werden ja in Matrizen abgelegt. Für ein zweidimensionales Bild benötigt MATLAB also mindestens zwei Spaltenvektoren 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 resultierenden 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 Spaltenvektor  $y$  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  $n$  Spaltenwerte.

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  $m$  Spalten der Matrix  $A$  Element von  $\mathbb{R}^{n \times m}$  gegen die gemeinsame unabhängige Variable  $x$  aufgetragen. Der Vektor  $x$  muss die Dimension  $n$  haben:  $x$  ist Element von  $\mathbb{R}^{n \times 1}$ .

Beim Befehl `plot(A,B)` bilden je eine Spalte 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  $n \times m$  Bilder in ein Graphikfenster eingebettet werden.  $p$  ist der Laufindex der  $n \times m$  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 auch 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 Or-

dinat 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 Koordinatenursprung 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  $y_1$  in Funktion von  $x_1$  und die rechte  $y$ -Achse auf  $y_2$  in Funktion von  $x_2$ .

#### 2.1.2 Masstab

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 die ursprünglichen Werte ein. Mit dem Befehl `axis equal` erhalten alle Achsen die gleiche Skalierung. Mit dem Befehl `axis ij` wechselt das Vorzeichen 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` sind 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 Win-

dow 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 figurenamemit 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 Llinien, 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 die 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.

### 2.1.3 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 Legende 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 (" $\pi$ ") und tiefgestellte " $\pi$ ", kursiv  $\pi$ " geschrieben oder griechische Zeichen enthalten.

$$A_1 e^{-\alpha t} \sin \beta t \quad \text{'\it A_{1}e^{\{-\alpha\itt\}}sin\beta\itt'}$$

(2.1)

Der Befehl `xlabel('text')` schreibt die Zeile textunter 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 ge-

schriebene oder griechische Zeichen enthalten.

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

### 2.1.4 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]<filename>` speichert die aktuelle Graphik im Format des speziell gewählten Druckertreibers und der zusätzlichen Option im aktiven Directory unter filename. 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 gedruckt. 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.



## 2.2 Dreidimensionale Graphik

### 2.2.1 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 Ansammlung von möglichen Liniendefinitionen auf.

Bei `mesh(Z)` entsprechen die Werte der Matrix  $Z$  Elementen 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 in 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  $C$  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 aktiven `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` schattiert 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 jedoch zusätzlich mit schwarzen Linien versehen.

### 2.2.2 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 Graphik 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 die view der aktuellen Graphik in der Variable  $T$  als  $4 \times 4$ -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  $4 \times 4$ -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]` ein 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.

### 2.2.3 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 horizontale 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.

## 2.3 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  $\text{lim}=[x_{\min} \ x_{\max}]$ . Mit  $\text{lim}=[x_{\min} \ x_{\max} \ y_{\min} \ y_{\max}]$  werden zusätzliche Schranken gesetzt. Mit dem Befehl `fplot('f', lim, tol)` mit  $\text{tol}<1$  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, 'LineStyle')` 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, oder den Betrag 1, d.h. das dazugehörige Stück von  $x$  wird aus dem Kuchen herausgezogen.

Mit dem Befehl `stem(y)` zeichnet eine Verteilung der Da-

ten 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 einem 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...

### 3 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 `elseif`, führt eine Gruppe von Befehlen dann aus, wenn eine vorgegebene logische Bedingung erfüllt ist.
- b) `for` durchläuft eine Schleife mit Befehlen für eine feste Anzahl von Wiederholungen.
- c) `while` repetiert solange eine von einer logischen Bedingung abhängige Schleife, 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.

#### 3.1 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 3.1: Übung M2b) Teil 3

```
1 if x>0 & x<1
2     disp('x ist groesser 0 aber kleiner 1.')
3     xneu=x+1;
4     x=xneu;
5 elseif x>=1 & x<10
6     disp('x liegt zwischen 1 und 10.')
7     xneu=x-10;
8     x=xneu;
9 elseif x>10
10    disp('x ist groesser als 10.')
11 elseif x==0
12    disp('x ist 0.')
13 else
14    disp('x ist negativ.')
15 end
```

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

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

```
end
```

Index bezeichnet die Variable in der `for`-Schleife. Wird sie gestartet, so wird dem Index der Startwert zugewiesen. Bei jedem Durchlauf werden die Anweisungen in der `for`-Schleife 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 Schleife vorzeitig ausgestiegen werden.

Listing 3.2: For-Schleife

```
1 for t=1:-0.2:0.2
2     y(round(10*t/2))=exp(1/t);
3 end
4 %Angaben in Command Window: bspfor;y, dann
   werden Zahlen generiert
```

Listing 3.3: For-Schleife

```
1 n=1;m=3;
2 for i=1:n
3     for j=1:m
4         A(i,j)=i+j
5     end
6 end
7 %Angaben inc Command Window: bspfor2;A, dann
   werden Zahlen generiert
```

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

```
while Ausdruck
    Anweisungen
end
```

Solange die logische Bedingung wahr (`>0`) ist, wird die Schleife 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 Schleife ausgestiegen werden.

Listing 3.4: While

```
1 while n<=10
2     y(n)=1+n-n^2+0.1*n^3;
3 end
4 %Incd Command Window: n=0;bspwhile; w=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 3.5: Switch

```
1 x=1;
2 switch x
```

```
3 case 1
4     disp('x hat den Betrag 1')
5 case {2,3,4}
6     disp('x hat den Betrag 2, 3 oder 4')
7 otherwise
8     disp('x hat weder den Betrag 1, 2, 3
        noch 4')
9 end
10 %In Command Window bspswitch
```

Listing 3.6: Switch

```
1 Antwort=input('Moechten Sie einen Test machen?
    ja/nein ','s');
2 disp(' ')
3 switch Antwort;
4     case 'ja'
5         disp('wie vorbildlich!')
6     case 'nein'
7         disp('wie schade!')
8     otherwise
9         disp('Ein "jein" kenne ich nicht!')
10 end
11 %In Command Window: bspswitch2 => Inputangaben
```

## 3.2 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 umreißen. 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, Schleifen, Kalkulationen, Wertzuweisungen, Kommentare und Leerzeilen beinhalten oder andere Funktionen aufrufen.

Mit dem Befehl `return` kann eine Funktion vorzeitig ver-

lassen werden, z.B. wenn eine vorgegebene Abbruchbedingung in der Funktion eingetreten ist.

Listing 3.7: botta

```
1 function [V,A0,AM] = botta(r,s1,s2)
2 %Botta Schief abgeschnittenen Zylinder berechnen
3 % Die Funktion botta(r,s1,s2) berechnet das
4   Volumen V,
5   % die Oberflaeche A0 und die Mantelflaeche AM
6   eines schief
7   % abgeschnittenen Zylinders.
8   % r ist der Radius, s1 die laengste Mantellinie
9   % und s2 die kuerzeste.
10
11 V=pi*r^2*(s1+s2)/2;
12 A0=pi*r*(s1+s2+r*sqrt(r^2+(s1+s2)^2/4));
13 AM=pi*r*(s1+s2);
14
15 %In Command Window: help botta
16 %In Command Window: [V,A0,AM]=botta(4,5,2)
17   angeben
```

Der Befehl `nargin` bestimmt die Anzahl der Eingabeargumente einer Funktion. In einer Funktion eruiert der Befehl `nargin`, wieviele Argumente die 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 3.8: nargin

```
1 function y = bspnargin(a,b)
2   if nargin<1
3     error('Keine Eingänge')
4   elseif nargin==1
5     y=a^2-4;
6   elseif nargin==2
7     y=a^2+4*b^2-4;
8   end
9 end
```

```
10 %In Command Window: bspnargin(3) => 5, bspnargin
11 (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 3.9: nargout

```
1 function [y,z] = bspnargout(a,b)
2   if nargin<1
3     error('Keine Eingänge')
4   elseif nargin==1
5     y=a^2-4;
6   elseif nargin==2
7     y=a^2+4*b^2-4;
8     if nargout==2
9       z=a^2+b-16;
10    end
11  end
12 end
13 %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 sie dort ebenfalls für global erklärt wurde.

Listing 3.10: nargout

```
1 function y = bspglobal(a)
2   global b
3   if nargin<1
4     error('Keine Eingänge')
5   elseif nargin==1
6     y=b^2+a^2-4;
7     b=b+1;
8   end
9 end
10 %In Command Window: global b; b; b=3; bspglobal(b); b
```

### 3.3 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, und 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-Files 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 ausge-

führt 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.

## 4 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

### 4.1 Getting started in Matlab

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

### 4.2 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:  $\Rightarrow$  Try this example  $\Rightarrow$  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

### 4.3 Was ist eigentlich Simulink?

Simulink basiert auf Matlab und ist eine zusätzliche Umgebung. Während Matlab ein text-basiertes 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**.

### 4.4 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 beinhaltet Auslegungen von z.B. einem PID-Regler.
5. **Matlab** kennen Sie bereits: dies ist ein leistungstarker textbasierter Editor.
6. **Simulink** ist dagegen grafisch orientiert.

## 4.5 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 Physikimplementiert... und auch diese Rechner werden in Simulink programmiert. Dies ist SState of the art für die Automobilindustrie, Flugzeugindustrie und Traction.
5. **Testen und watchen** Die Software wird auf der Target-hardware und im Feld getestet.

## 5 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 Einstieg 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.

### 5.1 M2a) übung zu Kapitel 3

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

⇒ `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.

⇒ `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.

⇒ `logspace(xstart, xend)` erzeugt einen Vektor. Der Vektor besteht somit aus 50 logarithmischen verteilten Punkten zwischen  $10^{xstart}$  und  $10^{xend}$ .

⇒ `logspace(xstart, xend, n)` erzeugt `n` Punkte zwischen  $10^{xstart}$  und  $10^{xend}$

⇒ `logspace(xstart, pi, n)` erzeugt `n` Punkte zwischen  $10^{xstart}$  und  $\pi$ .

- If `n` is 1, `logspace` returns  $10^{xend}$ .
- 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`).

⇒ `a=(0:1:19)` oder `a=[0:1:19]`

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

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

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

⇒ `A=[a;b]`

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

⇒ `A(1,20)=0`

⇒ `A(2,20)=0`

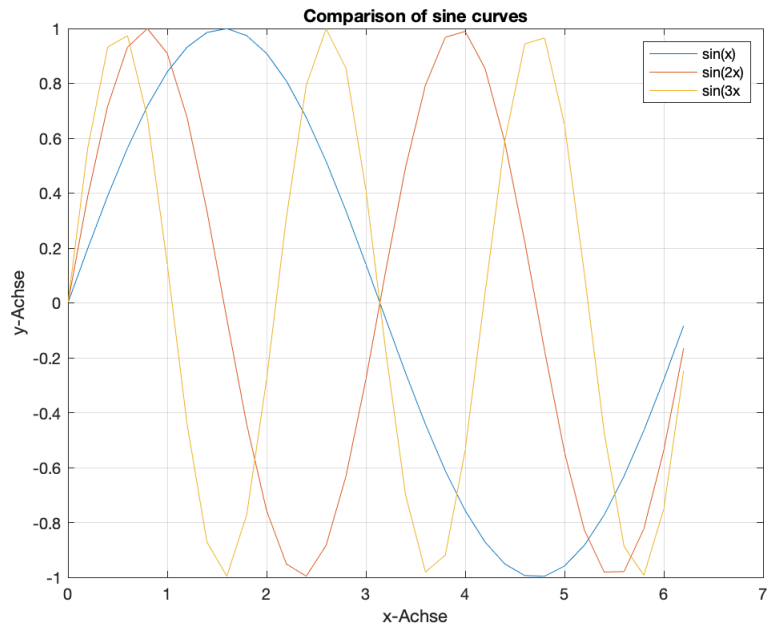
### 5.2 M2b-c) übung zu Kapitel 4 und 5

- 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 5.1: übung M2b) Teil 1

```
1 | x=(0:0.2:2*pi);
2 | plot(x,sin(x))
3 | grid off, hold on
4 | plot(x,sin(2*x))
5 | plot(x,sin(3*x))
6 | legend('sin(x)', 'sin(2x)', 'sin(3x)')
7 | title('Comparison of sine curves')
8 | xlabel('x-Achse')
9 | ylabel('y-Achse')
10 | grid on
```





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

Listing 5.2: Übung M2b) Teil 2

```

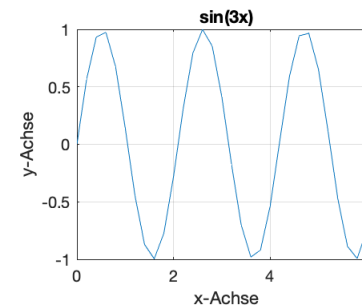
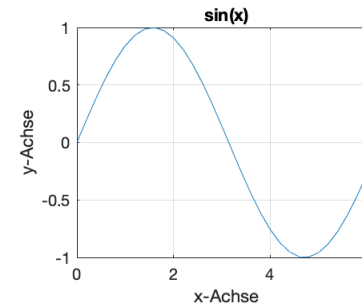
1 subplot(2,2,1)
2 x=(0:0.2:2*pi);
3 plot(x,sin(x))
4 title('sin(x)')
5 xlabel('x-Achse')
6 ylabel('y-Achse')
7 grid on
8
9 subplot(2,2,2)
10 plot(x,sin(2*x))
11 title('sin(2x)')
12 xlabel('x-Achse')
13 ylabel('y-Achse')
14 grid on
15
16 subplot(2,2,3)

```

```

17 plot(x,sin(3*x))
18 title('sin(3x)')
19 xlabel('x-Achse')
20 ylabel('y-Achse')
21 grid on

```



c) Stellen Sie  $e^x$  und  $\ln(x)$  in einer neuen Figur dar.

Listing 5.3: Übung M2b) Teil 3

```

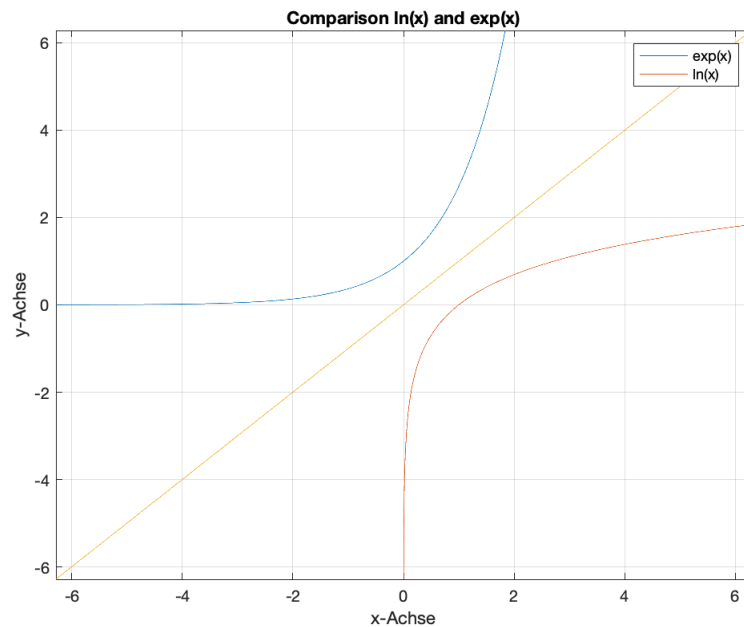
1 % Definition zweier Intervalle, eine fuer die Exponentialfunktion und
2   andere fuer die Logarithmusfunktion.
3 x=(-2*pi:0.002:2*pi);
4 y=(0.00001:0.002:2*pi);
5 % Plotten der Exponentialfunktion
6 plot(x,exp(x))
7 hold on

```

```

8  % Plotten der Logarithmusfunktion
9  plot(y,log(y))
10 % Plotten der ersten Winkelhalbierende
11 plot(x,x)
12 % Gleiche Achsenverteilung
13 axis([-2*pi 2*pi -2*pi 2*pi])
14
15 grid on
16 % Legende
17 legend('exp(x)', 'ln(x)')
18 xlabel('x-Achse')
19 ylabel('y-Achse')
20 title('Comparison ln(x) and exp(x)')

```



## 6 M3 ÜBUNG: VEKTOREN UND ARRAYS, FUNKTIONEN

### Drehbuch

1. Fragen zum Buch.
2. Lernsequenz: 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, dass 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.

### 6.1 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:

$$sx(t) = |\vec{v}| \cdot \cos(\alpha) \cdot t \quad (6.1)$$

$$sy(t) = |\vec{v}| \cdot \sin(\alpha) \cdot t - \frac{g}{2} \cdot t^2 \quad (6.2)$$

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

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

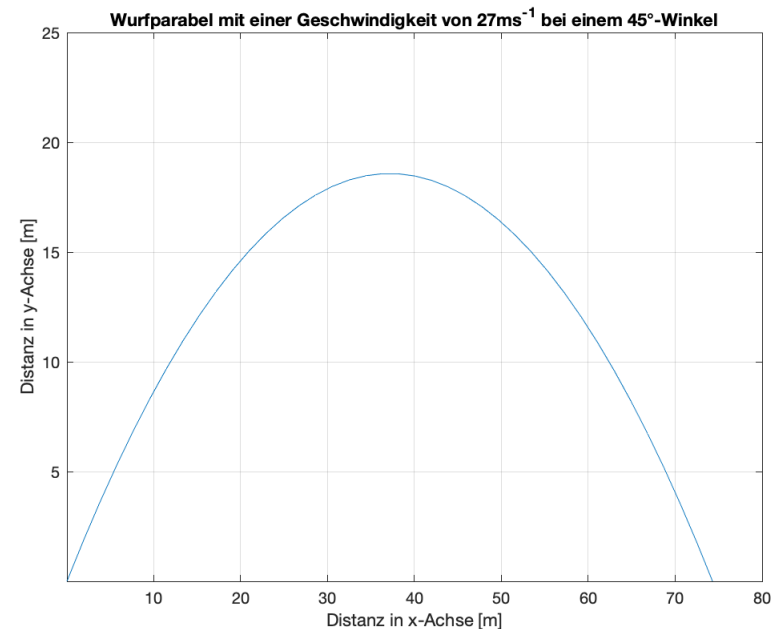
$|\vec{v}|$ : Betrag der Wurfgeschwindigkeit [m/s].

$\alpha$ : Abschusswinkel [grad, rad].

$t$ : Zeit [s].

$g = 9.81$ : Gravitationskonstante [m/s<sup>2</sup>]

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 6.1: Wurfparabel für 60 Grad

```
1 clear variables, close all
2
3 % Zeit zwischen 0 und 40 Sekunden
4 t = (0:0.1:40);
5
6 % Gravitation
7 g = 9.81;
8
9 % Abschusswinkel
10 alpha = 45;
```

```

11 angle = alpha .* (pi/180);
12
13 % Definition der Wurfgeschwindigkeit
14 v = 27;
15
16 % Anfangszustand der Masse
17 sx0 = 0;
18 sy0 = 0;
19
20 % Komponenten der Beschleunigung
21 ax = 0;
22 ay = -g;
23
24 % Komponenten der Geschwindigkeit
25 vx = v*cos(angle);
26 vy = v*sin(angle)-g.*t;
27
28 % Komponente der Bewegungszustand
29 sx=v*cos(angle).*t+sx0;
30 sy=v*sin(angle).*t-g/2*t.^2+sy0;
31
32 % Wurfparabel
33 figure;
34 plot(sx,sy); grid on
35 xlabel('Distanz in x-Achse [m]');
36 ylabel('Distanz in y-Achse [m]');
37 title(['Wurfparabel mit einer Geschwindigkeit von ',num2str(v),'ms-1 bei
    einem ',num2str(alpha),'-Winkel']);
38 axis([0.01 80 0.01 25]);

```

## 6.2 M3a) übung zu Funktionen

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

Listing 6.2: übung M3a)

```

1 function y = linspace(d1, d2, n)
2 %Linspace Linearly spaced vector.
3 %   Linspace(X1, X2) generates a row vector of 100 linearly
4 %   equally spaced points between X1 and X2.
5 %
6 %   Linspace(X1, X2, N) generates N points between X1 and X2.
7 %   For N = 1, Linspace returns X2.
8 %

```

```

9 %   Class support for inputs X1,X2:
10 %       float: double, single
11 %
12 %   See also LOGSPACE, COLON.
13
14 %   Copyright 1984-2016 The MathWorks, Inc.
15
16 if nargin == 2
17     n = 100;
18 else
19     n = floor(double(n));
20 end
21 if ~isscalar(d1) || ~isscalar(d2) || ~isscalar(n)
22     error(message('MATLAB:linspace:scalarInputs'));
23 end
24 n1 = n-1;
25 c = (d2 - d1).*(n1-1); %check intermediate value for appropriate treatment
26 if isinf(c)
27     if isinf(d2 - d1) %opposite signs overflow
28         y = d1 + (d2./n1).*(0:n1) - (d1./n1).*(0:n1);
29     else
30         y = d1 + (0:n1).*((d2 - d1)./n1);
31     end
32 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

- Wie lang ist die Länge eines Standardvektors, wenn **keine Länge** definiert wird?  
⇒ Generierung eines Vektors mit 100 linear gleichabständigen Punkten.
- Mit `help linspace` erscheint die Hilfe dieser Funktion im Command Window
  - Wo befindet sich dieser Text in der Datei?  
⇒ Der angezeigte Text befindet sich im Command Window.
  - Ab welcher Zeile wird der Hilfetext nicht mehr ausgegeben? Wie wird dies abgegrenzt?  
⇒ Eine Zeile vor der Zeile des Copyrights wird nichts mehr angezeigt. Durch `edit help` kann man dies ändern.

## 6.3 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 6.3: 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
```

## 6.4 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 Funktion 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:

- Ein pdf von Ihrer Funktion dual(a,b)
- Das Resultat der Publish Funktion (auch als pdf). Der Code des Testfiles soll nicht eingebunden werden