MATLAB Befehlsübersicht/Einführung

M. Thaler, ZHAW

Inhaltsverzeichnis

1 Allgemeines

- MATLAB

  MATLAB ist ein Interpreter (wie z.B. BASIC) Eingegebene Anweisungen werden unverzögert ausgeführt und ihr Ergebnis angezeigt (Echo). Wird eine Anweisung mit einem Semikolon (Strichpunkt) abgeschlossen, so wird das Echo unterdrückt.

- Datentypen

  MATLAB kennt nur komplexe Matrizen als Objekte Ein Skalar entspricht dabei einer 1x1-Matrix.

- Online Hilfe

  MATLAB enthält eine online-Hilfe ≫ `help topic` öffnet Hilfe zum Thema `topic`.

- Gross- / Kleinschreibung

  MATLAB ist per Default case-sensitive.
  ≫ `casesen off` schaltet case-sensitivity aus
  ≫ `casesen on` schaltet case-sensitivity wieder ein.

- Matlab beenden

  Beenden einer MATLAB-Sitzung mit`Quit` aus dem File-Menu oder mit Eingabe von ≫ `quit`.

- Kommentare

  Kommentare beginnen mit % und sind mit dem Zeileende abgeschlossen.
  ```
  ≫ % Folgende Zeile enthält einen Matlab-Befehl und einen Kommentar
  ≫ Z=zeros(10, 4) % erzeuge Matrix Z mit 10 Zeilen und 4 Spalten
  ```

- Initialisierung

  Allgemeine Befehle (sollten zu Beginn von jedem Programm stehen)
  ≫ `clear all` löscht sämtliche Variablen
  ≫ `clc` löscht das MATLAB Command Window

- Pi

  Die Konstante π ist vordefiniert und hat den Symbolischen Namen `pi`.

- Komplexe Zahlen

  Die Imaginärzahl i, I oder j ist als i = sqrt(-1) vordefiniert, eine komplexe Zahl wird wie folgt geschrieben (mit oder ohne *): ≫ `a = 3 + 5i` ≫ `b = 4 + 2*i` Achtung: Eine Neudefinition von i, z.B. i = 2, überschreibt die ursprüngliche Bedeutung.

- Ausgabe unterdrücken

  Befehle (Zeilen), die mit einem `;` abgeschlossen werden, werden zwar ausgeführt, das Resualtat aber nicht auf dem Bildschirm ausgegeben.

- Fortsetzungszeichen

  Ein Anweisung kann auf der nächsten Zeile fortgesetzt werden, indem am Ende der Zeile drei Punkte als Fortsetzungszeichen eingegeben werden:
  ```
  ≫ A=[0 1 0 1 0 2 ...
  1 2 3]
  ≫ A = 0 1 0 1 0 2 1 2 3
  ```

2 Matrizen

2.1 Eingabe

- auf mehreren Zeilen

```
≫ M = [ 0 1 0
        1 0 2
        1 2 3 ]
```

- auf einer Zeile (; Zeilentrennzeichen)

```
≫ M = [0 1 0; 1 0 2; 1 2 3]
```

2.2 Vektoren

`≫ z = [1 2 3 4]`	Zeilenvektor (1x4-Matrix)
`≫ s = [2 2 2]'`	Spaltenvektor (Hochkomma = konjugiert-transponiert, 3x1-Matrix)

2.3 Zugriff auf Matrixelemente

`≫ M(i,j)`	das ij-te Element der Matrix M (i-te Zeile, j-te Spalte)

2.4 Zugriff auf Matrixelemente

`≫ A.*B`	elementweise Multiplikation $a_{ij} \cdot b_{ij}$
`≫ A./B`	elementweise Division a_{ij}/b_{ij}
`≫ A.^x`	elementweise Exponentierung a_{ij}^x (x: Skalar)

2.5 Grundoperationen auf Matrizen

`≫ A+B`	Matrix-Summe
`≫ A-B`	Matrix-Differenz
`≫ A*B`	Matrix-Produkt (bei produktkompatiblen Matrixdimensionen)
`≫ A/B`	Matrix-*Division*, entspricht $A \cdot B^{-1}$
`≫ A\B`	Matrix-*Linksdivision*, entspricht $A^{-1} * B$ (löst das Gleichungssystem $A \cdot x = B$)
`≫ A'`	konjugierte Transposition, bei reellen Koeffizienten: Transposition
`≫ A.'`	Transposition (ohne Konjugation)

2.6 Spezielle Matrizen

`≫ Z=zeros(m,n)`	definiert eine m×n-Matrix mit lauter Nullen
`≫ O=ones(m,n)`	definiert eine m×n-Matrix mit lauter Einsen
`≫ I=eye(n)`	definiert eine (quadratische) m×n-Einheitsmatrix
`≫ I=eye(m,n)`	definiert eine m×n-Einheitsmatrix (m Zeilen, n Spalten)
`≫ R=rand(m,n)`	definiert eine m×n-Matrix mit gleichverteilten Zufallszahlen zwischen 0 und 1 als Elemente
Beispiel:	
`≫ R=randn(3,2)`	definiert eine 3×2-Matrix mit normalverteilten Zufallszahlen
`≫ I=zeros(size(R))`	definiert eine Nullmatrix mit der gleichen Dimension wie R

2.7 Determinaten und inverse Matrizen

≫ `det(A)`	Determinante der quadratischen Matrix A
≫ `inv(A)`	Inversion einer quadratischen Matrix
≫ `A\eye(size(A))`	Berechnung der Inversen der quadratischen Matrix `A`

Verwenden Sie aus numerischen Grüden zur Berechnung von $A^{-1} \cdot B$ die Anweisung $A \setminus B$ und nicht $inv(A) \cdot B$

2.8 Der Doppelpunkt-Operator

2.8.1 Erzeugen von Vektoren

≫ `x=5:12`	$\Longleftrightarrow$	`x=[5 6 7 ... 11 12]`
≫ `x=0:0.1:1`	$\Longleftrightarrow$	`x=[0.0 0.1 0.2 ... 0.9 1.0]`
≫ `x=1:-0.1:0`	$\Longleftrightarrow$	`x=[1.0 0.9 0.8 ... 0.1 0.0]`

2.8.2 Auswählen von Zeilen oder Spalten in Matrizen

≫ `A(:,j)`	die j-te Spalte der Matrix A
≫ `A(j,:)`	die j-te Zeile der Matrix A
≫ `A(:,j:k)`	Untermatrix von A mit den Spalten A(:,j), A(:,j+1), .., A(:,k)
≫ `A(:)`	alle Elemente der Matrix A als ein Spaltenvektor angeordnet
≫ `x=x(:)`	macht aus jedem x einen Spaltenvektor
≫ `y=y(:).'`	macht aus jedem y einen Zeilenvektor
≫ `A=x(:,ones(1,n))`	erzeugt Matrix A aus n Kopien des Spaltenvektors x
≫ `A=y(ones(m,1),:)`	erzeugt Matrix A aus m Kopien des Zeilenvektors y

2.9 Elementare mathematische Funktionen

Elementare mathematische Funktionen mit einem Argument, das ein Skalar, ein Vektor oder eine Matrix sein kann. Die Funktionen werden elementweise angewendet mit Ausnahme der Matrix-Exponent Funktion `expm`.

`acos, asin, atan`	inverse trigonometrische Funktionen
`atan2`	4-Quadranten arcus-tangens, atan2: zwei Argumente
`cos, sin, tan`	trigonometrische Funktionen
`exp`	Exponentialfunktion
`log`	natürlicher Logarithmus
`log10`	Logarithmus zur Basis 10
`real, imag`	Real- resp. Imaginärteil des komplexen Arguments
`sqrt`	quadratische Wurzel
`expm`	Matrix-Exponent: e^A

3 Graphische Funktionen

≫ `plot(x,y)`	zeichnet die Werte des Vektors y (Ordinate) gegen diejenige des Vektors x (Abszisse), lineare Interpolation zwischen den Punkten
≫ `plot(x,y,'o')`	zeichnet Kreise (klein o) in den Punkten, keine Interpolation
≫ `plot(x1,y1,x2,y2)`	zeichnet zwei Funktionen in derselben Graphik
≫ `semilogx(x,y)`	Abszisse mit logarithmischem (zur Basis 10) Massstab
≫ `semilogy(x,y)`	Ordinate mit logarithmischem Massstab
≫ `loglog(x,y)`	beide Kordinaten mit logarithmischem Massstab
≫ `grid`	zeichnet ein Gitternetz in die bestehende Graphik
≫ `axis([xs,xe,ys,ye])`	legt die Skalierung der x- und der y-Achse fest (wird nach Plot-Befehl aufgerufen)
≫ `axis('auto')`	automatische Skalierung
≫ `v=axis;`	Skalengrenzen des aktuellen Plots in Vektor v
≫ `axis(axis)`	Einfrieren der Skalenwerte
≫ `axis('equal')`	Einheiten für beide Achsen gleich (Kreis erscheint als Kreis und nicht als Ellipse)
≫ `subplot(2,1,1)`	Plot obere Hälfte des Felds (2 Zeilen, 1 Spalte, 1. Graph)
≫ `subplot(2,1,2)`	untere Hälfte (2 Zeilen, 1 Spalte, 2. Graph)
≫ `subplot(1,2,1)`	linke Seite (1 Zeile, 2 Spalten, 1. Graph)
≫ `subplot(2,2,3)`	links unten (2 Zeilen, 2 Spalten, 3. Graph)
≫ `subplot(1,1,1)`	ganzes Feld (1 Zeile, 1 Spalte, 1. Graph)
≫ `hold on`	einfrieren der aktuellen Graphik, um zusätzliche Graphen in das Bild eintragen zu können
≫ `hold off`	zurücksetzen
≫ `title('Text')`	Titel für Graphik
≫ `xlabel('Text')`	Beschriftung x-Achse (Abszisse)
≫ `ylabel('Text')`	Beschriftung y-Achse (Ordinate)
≫ `set(gcf,'name','Text')`	Titel des Plot-Fensters
≫ `figure(#)`	setzt das Plot-Fenster mit Nummer # als aktiv oder erzeugt ein neues Fenster mit dieser Nummer
≫ `clf`	Löschen der Graphik (clear figure)
≫ `close all`	Schliessen aller Figurenfenster

Beispiele:

≫ `x=0:0.01:pi/2.0;`	$0 < x < \pi/2$, Schrittweite: 0.01
≫ `plot(x,tan(x))`	Darstellung: $tan(x)$
≫ `axis([0 pi/2 0 10])`	x-Achse: 0 bis $\pi/2$, y-Achse: 0 bis 10
≫ `title('tangens')`	Titel der Graphik
≫ `xlabel('freq')`	Beschriftung x-Achse
≫ `ylabel('tan(x)')`	Beschriftung x-Achse

4 Kontrollstrukturen

4.1 Die for-Schleife

```
for variable = <expression>,
    <statements>
end
```

- Definition einer (dekrementierten) Variablen in einer for-Schleife

```
for s=1.0:-0.1:0.0,
   ...
end
```

- Verschachtelte for-Schleife (zur Definition der Hilbert-Matrix H)

```
H=zeros(n,n);
  for i=1:n,
    for j=1:n,
      H(i,j)=1/(i+j-1);
    end
  end
```

Anmerkung: die Vordefinition (preallocation) der Matrix H mit der Funktion `zeros` erhöht die Effizienz der Programmausführung, da sonst die Dimension von `H` in jedem Schritt der verschachtelten for- Schleife neu festgelegt werden muss.

- Die folgenden Anweisungsfolgen sind identisch (im zweiten Fall müssen jedoch `j`, `m` und `n` nicht definiert werden

```
[m,n]=size(A);
for j=1:n,
  v=A(:,j);
    ...
end
```

```
for v=A,
    ...
end
```

- Verwenden Sie immer die Eigenschaft von MATLAB, Funktionen von Vektoren und Matrizen zu verarbeiten

```
t=0.0:0.01:10.0;
y=sin(t);
```

anstelle von

```
j=0;
for t=0.0:0.01:10.0,
  j=j+1;
  y(j)=sin(t);
end
```

4.2 Die while-Schleife

Die while-Schleife ist wie folgt definiert:

```
while <expression>,
    <statements>
end
```

mit <expression> := <expression> <relational operator> <expression>
und <relational operator>:

<	kleiner als
>	grösser als
<=	kleiner-gleich
>=	grösser-gleich
==	gleich (Identität)
~=	nicht gleich

Beispiel zur Bestimmung der Rechnergenauigkeit

```
eps = 1;
while (1 + eps) > 1,
    eps = eps / 2;
end
eps = eps * 2;
```

4.3 Bedingte Anweisungen

```
if <expression>,
    <staements>
end
```

```
if <expression>,
    <statements>
else
    <statements>
end
```

```
if <expression>,
        <statements>
elseif <expression>,
        <statements>
else
        <statements>
end
```

4.4 Case oder Switch Anweisung

```
switch <switch_expr>
   case <case_expr>,
        <statement>, ..., <statement>
   case {<case_expr1>, <case_expr2>, <case_expr3>,...}
        <statement>, ..., <statement>
   ...
   otherwise,
        <statement>, ..., <statement>
end
```

mit

<switch_expr>	ein Ausdruck (auch Variable), der eine Menge von Werten annehmen kann
<case_expr>	einer der Werte, die zur Menge der Werte von <switch_expr> gehören

4.5 Beispiel zu Kontrollstrukturen

```
n=6;
a=zeros(n,n);
for i=1:n,
    for j=1:n,
        if i==j,
            a(i,j)=4;
         elseif abs(i-j)==1,
            a(i,j)=2;
         else
            a(i,j)=1;
         end
    end
end
```

4.6 Funktionen

Funktionen können mit und ohne Parameter und mit und ohne Rückgabewert definiert werden, die allgemeinste Definition hat folgende Form (Angaben in geschweiften Klammern sind optional):

```
function {returnValue  = } <name> {(<parameter>, ...<parameter>)}
    <statement>;
    <statement>;
end
```

Beispiel:

```
function nextState = update(currentState, inVal)
    switch currentState
        case 0,
            if inVal > 0
                nextState = 1;
            else
                nextState = 0;
            end
        case 1,
            if inVal > 5
                nextState = 1;
            else
                nextState = 0;
            end
        otherwise,
                nextState = currentState
end
```

4.7 M-Files

M-Files sind Skripts, die eine Folge von MATLAB-Befehlen ausführen können. Beachten Sie, dass ein M-File entweder nur Befehlszeilen oder Funktionen enthalten kann. Beim Ausführen eines M-Files mit Funktionen, wird zu Beginn immer die erste Funktion ausgeführt.

5 Systeme

5.1 Systeme 2.Ordnung

LTI-SISO Systeme: LTI: linear time invariant, SISO: single input single output
Übertragungsfunktion T(s):

$$T(s) = \frac{1}{s^2 + \frac{\omega_0}{q} + \omega_0^2} = \frac{1}{s^2 + 2\xi\omega_0 + \omega_0^2} \qquad \text{mit } s = j\omega$$

≫ `[num, den] = ord2(w0, d)`	LTI 2. Ordnung mit d $= \xi = \frac{1}{2q}$ und w0 $= \omega_0$
≫ `[A, B, C, D,] = ord2(w0, d)`	LTI 2. Ordnung in Zustandsraumdarstellung

5.2 LTI Systeme beliebiger Ordnung

Übertragungsfunktion T(s):

$$T(s) = \frac{b_m s^m + \cdots + b_1 s + b_0}{a_n s^n + \ldots + a_1 s + a_0} \qquad \text{mit } n \geq m$$

≫ num = $[b_m, b_{m-1} \ldots b_1, b_0]$	
≫ den = $[a_n, a_{n-1} \ldots a_1, a_0]$	
≫ `[A, B, C, D] = tf2ss(num, den)`	Übetragungsfunktion → Zustandsraum
≫ `[num, den] = ss2tf(A,B,C,D,ik)`	Zustandsraum → Übertragungsfunktion mit `ik = k-tes` Eingangssignal (wenn nur 1 Eingangssignal: `ik = 1`)
≫ `sys = tf(num, den)`	erzeugt lti-System aus Übertragungsfunktion
≫ `sys = ss(A,B,C,D)`	erzeugt lti-System aus Zustandsraumdarstellung

5.3 Stoss- und Schrittantwort

≫ `impulse(sys)`	zeichnet Stossantwort
≫ `impulse(sys,tfinal)`	zeichnet Stossantwort bis zum Zeitpunkt t_{final}
≫ `[y,t] = impulse(sys)`	Stossantwort als Vektor und dazugehörigem Zeitvektor t
≫ `impulse(A,B,C,D,ik)`	zeichnet alle Stossantworten für i_k-ten Eingang
≫ `impulse(A,B,C,D,ik,t)`	zeichnet Stossantworten für i_k-ten Eingang für vorgegebene Zeitpunkte
≫ `step(sys)`	zeichnet Schrittantwort
≫ `step(sys,t)`	zeichnet Schrittantwort für vorgegebene Zeitpunkte (in Vektor t)
≫ `[y,t] = step(sys,t)`	Schrittantwort als Vektor und dazugehörigem Zeitvektor t
≫ `step(A,B,C,D,ik)`	zeichnet alle Stossantworten für i_k-ten Eingang
≫ `step(A,B,C,D,ik,t)`	zeichnet alle Stossantworten für i_k-ten Eingang für vorgegebene Zeitpunkte

5.4 Pole und Nullstellen

Stabilitätsuntersuchungen, Pole von $G_0(s) = \frac{num}{den}$:

≫ `roots(den)`	Nullstellen des Nennerpolynoms
≫ `eig(A)`	Eigenwerte der Systemmatrix
≫ `pzmap(sys)`	graphische Darstellung der Pole und Nullstellen eines LTI-Systems

Stabilitätsuntersuchungen, Pole von:

$$T(s) = \frac{\cdots}{1 + G_0(s)} = \frac{\cdots}{1 + \frac{num}{den}} = \frac{\cdots den}{den + num}$$

≫ `roots(num + den)`	Pole von $T(s)$ resp. $num + den$

5.5 Bodediagramm

≫ `bode(sys)`	zeichnet Bodediagramm, Frequenzbereich automatisch bestimmt
≫ `bode(num, den)`	dito
≫ `om = logspace(a,b)`	erzeugt im log-Massstab 50 gleichmässig verteilte Stützstellen zwischen 10^a und 10^b
≫ `om = logspace(a,b,N)`	dito, aber mit N Stützstellen
≫ `bode(sys, om)`	zeichnet Bodediagramm mit vorgegebenem Frequenzbereich
Beispiel:	
≫ `[ampl,phase,om] = bode(sys)`	Frequenz- und Phasengang (in Grad) speichern
≫ `[ampl,phase,om] = bode(A,B,C,D,ik,om)`	dito aus Zustandsdarstellung
≫ `magdB = 20*log10(ampl)`	Amplitudengang in dB
≫ `subplot(2,1,1)`	oberes Plotfenster
≫ `semilogx(om/2/pi,magdB)`	Amplitudengang in Funktion der Frequenz
≫ `subplot(2,1,2)`	unteres Plotfenster
≫ `semilogx(om/2/pi,phase)`	Phasengang in Funktion der Frequenz

5.6 Ortskurven und Nyquistdiagramme

≫ `nyquist(sys)`	erzeugt Nyquistdiagramm, Frequenzbereich und Graphik automatisch bestimmt
≫ `om = linspace(a,b)`	erzeugt im lin-Massstab 100 gleichmässig verteilte Stützstellen zwischen a und b (a, b ¡ 0 erlaubt)
≫ `om = linspace(a,b,N)`	dito, aber mit N Stützstellen
≫ `nyquist(sys, om)`	erzeugt Nyquistdiagramm mit vorgegebenem Frequenzbereich
Beispiel:	
≫ `[re,im,om ] = nyquist(sys, om)`	Speichern des Frequenzganges
≫ `[re,im,om ] = nyquist(A,B,C,D,ik,om)`	dito, aber aus Zustandsdarstellung
≫ `plot(re, im)`	Ortskurve zeichen
≫ `axis('equal')`	ohne Masstabsverzerrung
≫ `hold on`	Zeichnung einfrieren
≫ `plot(-1, 0, 'o')`	Nyquist-Punkt einfügen
≫ `hold off`	Zeichnung freigeben

5.7 Betrag und Argument komplexer Zahlen

≫ `mag = abs(F)`	Beträge der komplexen Elemente von F
≫ `phase = angle(F)`	Argimente (Winkel) der komplexen Elemente von F in Rad
≫ `phase = unwrap(angle(F))`	elimieren der 2π-Phasensprünge zwischen aufeinanderfolgenden Elementen
≫ `phase = 180 * phase / pi`	Umrechnung von Rad → Grad

6 Signale

6.1 Abtastung und Rekonstruktion

≫ `xr = decimate(x,r)`	reduziert die Abtastrate des (äquidistant) abgetasteten Signals x um den Faktor r, vor der Datenreduktion (Dezimation) wird das Signal tiefpassgefiltert
≫ `xi = interp(x,r)`	erhöht die Abtastrate des abgetasteten Signals x um den Faktor r, kann verwendet werden um ein abgetastetes Signal zu rekonstruieren; Umkehrfunktion von `decimate`
≫ `xi = spline(t,x,ti)`	interpoliert zwischen den Werten des abgetasteten Signals x zu den vorgegebenen Zeitpunkten t_i mit einem kubischen Interpolationsspline, kann verwendet werden um ein abgetastetes Signal zu rekonstruieren, das Signal x muss dabei nicht äquidistant abgetastet sein, Vorsicht beim Verwenden bei verrauschten Signalen

6.2 Signal- und Leistungsspektrum

≫ `fft(x,n)`	berechnet das diskrete Fourier-Spektrum von x basierend auf dem FFT Algorithmus, n muss ein Vielfaches von 2 sein, z.B. $2^8 = 256$
≫ `spectrum(x)`	zeichnet das diskrete Leistungsspektrum von x mit normierter Frequenz: 2 entspricht der Abtastfrequenz, 1 der Nyquist-Frequenz bzw. der halben Abtastfrequenz
≫ `P = spectrum(x)`	berechnet das Leistungsspektrum von x in Funktion der Frequenz
≫ `specplot(P, fs)`	zeichnet das geschätzte Leistungsspektrum von P in Funktion der Frequenz mit 95%-Vertrauensintervall

6.3 Zeitreihenanalye

≫ `mean(x)`	berechnet eine Schätzung für den Erwartungswert der Zufallsgrösse x, entspricht dem linearen Mittelwert der Realisierungen der Stichprobe $\frac{sum(x)}{n}$, wobei der Umfang der Stichprobe `n = length(x)` ist
≫ `var(x)`	berechnet eine erwartungstreue und konsistente Schätzung für die Varianz (Quadrat der Streuung) der Zufallsgrösse x (die einzelnen Elemente von x müssen dabei voneinander unabhängig sein), entspricht $\frac{sum(x^2)-sum(x)^2}{(n-1)}$
≫ `xcorr(x,'unbiased')`	berechnet eine Schätzung für die Autokorrelationsfunktion des diskreten stochastischen Signals x
≫ `xcorr(x,y,'unbiased')`	berechnet eine Schätzung für die Kreuzkorrelationsfunktion der diskreten stochastischen Signale x und y