Schnellsiedekurs Matlab

Im folgenden sollen die für Regelungstechnik I wichtigsten MATLAB-Befehle kurz zusammengestellt werden. Für eine genauere Darstellung sei hier auf [1, 2, 3, 4] und insbesondere auch auf die sehr ausführliche MATLAB Online Referenz, sowohl zu MATLAB selbst als auch speziell zur Control System Toolbox, verwiesen. Die genaue Syntax der Befehle erhält man leicht über die MATLAB Hilfe.

Der Name MATLAB steht für MATrix LABoratory. Die Herkunft und Stärke von MATLAB liegt also im numerischen Rechnen mit Matrizen. Zur Verwendung von MATLAB für regelungstechnische Zwecke ist die CONTROL SYSTEM TOOLBOX unerlässlich. Diese Einführung in MATLAB stellt zunächst den allgemeinen Gebrauch von MATLAB vor und geht dann auf die spezifischen Funktionen dieser Toolbox ein.

1 Allgemeine Matlab-Grundlagen

Vektoren

`a=[1 2 3]` `b=[4, 5, 6]`	Eingabe von Zeilenvektoren: Spalten trennen durch Leerzeichen oder Komma
`a=1:3`	Erzeugt den Vektor `[1 2 3]`
`c=1:.1:2`	Erzeugt den Vektor `[1 1.1 1.2 1.3 ... 2]`
`a*b`	Zeile*Zeile = nicht definiert
`a.*b`	Elementweise Multiplikation
`b'`	Transposition -> Spaltenvektor
`a*b'`	Zeile*Spalte = Skalarprodukt
`a'*b`	Spalte*Zeile = dyadisches Produkt
`a.*b'`	Elementweise Zeile*Spalte = geht nicht

Matrizen

`M=[1 2 3; 4 5 6]`	Eingabe einer Matrix. Zeilen trennen durch ";" oder Zeilenwechsel
`M*2`	Multiplikation mit Skalar, ebenso: /, +, -
`M(1,:)`	Erste Zeile von M
`M(:,2)`	Zweite Spalte von M
`M(2,2:3)`	2. und 3. Element aus 2. Zeile
`A=[1 2;3 4]`	Eingabe einer Matrix
`A=[1 2` `3 4]`	dto.
`eig(A)`	Eigenwerte von A
`[V,D]=eig(A)`	Matrix der Rechtseigenenvektoren und diag{EW} Beachte: hier werden zwei Variablen zurückgegeben.
`V^-1`	Inverse
`inv(V)`	dto.
`V^-1*A*V`	Trafo auf Diagonalform (= D, wenn Trafo möglich)
`V*D/V`	Ursprüngliche Form aus Diagonalform und Rechtseigenvektoren
`det(A)`	Determinante
`rank(A)`	Rang
`poly(A)`	Koeffizienten des char. Polynoms
`expm(A*2)`	Transitionsmatrix zum Zeitpunkt $t = 2$

Polynome

Polynome werden als Vektor Ihrer Koeffizienten dargestellt. Bsp: Das Polynom $p(x) = x^3 + 3x^2 + 3x + 1$ wird dargestellt als der Vektor `[1 3 3 1]`.

`roots([1 3 3 1])`	Wurzeln eines Polynoms bestimmen
`poly ([1 2 3])`	Erstellt das Polynom mit den angegebenen Wurzeln Beachte: hier ist das Argument ein Vektor und keine Matrix
`conv([1 3 3 1], [1 4])`	Multiplikation von zwei Polynomen
`deconv(...)`	Polynomdivision
`residue(...)`	Partialbruchzerlegung
`polyval(...)`	Auswertung des Polynoms
`polyadd(...)`	Addition von Polynomen

Komplexe Zahlen

`z=-3+4*j`	Eingabe einer komplexen Zahl. Alternative: `-3+4*i`
`real(z)`	Realteil
`imag(z)`	Imaginärteil
`abs(z)`	Betrag
`angle(z)`	Winkel (in rad)

Verschiedenes

`%`	"%" führt Kommentare an
`a=5;`	";" unterdrückt Bildschirmausgabe
`str='Hallo Welt!'`	Eingabe von Zeichenketten entspricht Vektor aus Buchstaben `['H' 'a' 'l' ...]`
`pi`	Zahl π
`a=1,b=2`	Zwei Befehle in einer Zeile eingeben: trennen durch "," (Ergebnisse werden angezeigt) oder durch ";" (Ergebnisanzeige unterdrückt).
`a=[1 2 ...` `3]`	Befehl geht in neuer Zeile weiter Äquivalent zur Eingabe `a=[1 2 3]`
`A=[1 2` `3 4]`	Mehrzeilige Eingabe in Kommandozeile: mit Shift-Return Äquivalent zur Eingabe `A=[1 2; 3 4]`

Grafikausgabe

`t=0:.1:7;`	
`y=sin(t);`	erzeugt Vektor mit Funktionswerten
`plot(t,y)`	Erzeugt Diagramm y über t
`title('Zeitplot')`	Titel
`xlabel('Zeit t')`	x-Achsenbeschriftung
`ylabel('Signal y')`	y-Achsenbeschriftung
`legend(...)`	Legende bei mehreren Kurven in einem Schaubild
`axis([2 5 -3 3])`	Achsenskalierung `xmin xmax ymin ymax`
`grid on/off`	Gitternetz an/aus

Systembefehle

`help befehl`	Zeigt hilfe zu `befehl` an. Besser geht es per Menü
`clear`	Löscht alle Variablen
`who`	Zeigt alle Variablen an
`save dateiname`	Sichert alle Variablen nach dateiname.mat
`load dateiname`	Lädt alle Variablen aus dateiname.mat

Skripte, Schleifen

Skripte bestehen aus Befehlsfolgen, die ohne Angabe von Argumenten aufgerufen werden. Der Aufruf geschieht entweder durch Eingabe des Namens in der Kommandozeile oder durch klicken auf das "run"-Icon in der Kopfzeile des MATLAB-Editorfensters.

Eine `for`-Schleife funktioniert in MATLAB wie in vielen Hochsprachen. Die Zählvariable durchläuft dabei alle Werte eines angegebenen Vektors. Es gibt in MATLAB auch `while`-Schleifen (siehe MATLAB-Hilfe).

Das folgende Beispiel für ein Skript verdeutlicht die Probleme, die bei numerischer Berechnung aufgrund von Rundungsfehlern auftreten können. Es wird durch Vorgabe der

Wurzeln ein Vektor mit den Koeffizienten des entsprechenden Polynoms konstruiert (Befehl `poly`). Anschliessend werden die Wurzeln aus den Koeffizienten berechnet (Befehl `roots`). Die so errechneten Wurzeln sollten wieder die gleichen sein wie die vorgegebenen. Im Beispiel unten werden zunächst die Wurzeln -1.1 und -1.2 vorgegeben. Die aus dem Polynom berechneten Wurzeln werden in der komplexen Ebene aufgetragen. Nach einer kurzen Pause wird eine weitere Wurzel dazu genommen usw. Was wird passieren?

```
% Achtung, Numerik!!!
for k=1:30                          % Vektor 1:30 gibt Werte für k vor.
    tmp=roots(poly(-1-(1:k)/10));
    plot(real(tmp), imag(tmp),'*')
    xlim([-6 0]);
    axis equal;
    grid on;
    pause(2);
end
```

Datei: `achtung.m`

Funktionen

Während ein Skript immer das gleiche Ergebnis produziert, kann eine Funktion mit einem Argument aufgerufen werden und die Ergebnisse von Berechnungen als Variablen zurückgeben.

Das folgende Beispiel addiert die beiden Argumente.

```
function y=aplusb(a,b)
y=a+b;
```

Achtung! Die Datei muss den gleichen Namen haben wie die Funktion. Dabei sind Umlaute und Sonderzeichen nicht gestattet. Hier müsste die Datei also `aplusb.m` heissen. Skripte und Funktionen unterscheiden sich nicht in der Dateinamenserweiterung ("`.m`"). Der Aufruf in MATLAB wäre z.B.

```
c=aplusb(3,7)
```

Eine Funktion kann auch mehrere Variablen zurückgeben:

```
function [summe,differenz]=plusundminus(a,b)
summe=a+b;
differenz=a-b;
```

Weitere Beispiele für Skripte und Funktionen folgen im nächsten Abschnitt über Simulation.

Numerische Lösung (*Simulation*) von Differentialgleichungen

Zur Simulation von Differentialgleichungen sind immer zwei Bestandteile nötig: ein Code, der die Differentialgleichung selbst enthält und ein Code, der die Simulation durchführt. Generell werden Simulationen im Zeitbereich (Frequenzbereich siehe CONTROL SYSTEM TOOLBOX) in MATLAB immer mit Hilfe der Zustandsraumdarstellung durchgeführt. Als Beispiel nehmen wir hier das aus der Vorlesung bekannte Modell eines kontinuierlichen Rührkesselreaktors. Die folgende Funktion liefert zu gegebener Zeit `t` und zu gegebenem Zustandsvektor (hier: `y`) die Ableitung `dy`. Sie beinhaltet also die rechte Seite des Differentialgleichungssystems $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, wobei $\mathbf{y}$ ein Vektor ist. Die Datei heisst `cstr.m` (eine vollständige Version gibt es auf den IST-Webseiten).

```
function dy=cstr(t,y)
% Simulationsgleichungen für den CSTR
Tc = 300;
Rho = 1000;
% (weitere Variablendefinitionen ausgelassen)
dy=[ -((ca*k)/exp(Ew/(R*T)))+((-ca+cap)*Vp)/Vr
     (-T+Tc)*(Vp/Vr+(A*U)/(cp*Vr*Rho))-(ca*hr*k)/(cp*exp(Ew/(R*T))*Vr*Rho)];
```

In MATLAB stehen mehrere verschiedene numerische Lösungsverfahren zur Verfügung. Die Auswahl des richtigen Simulationsverfahrens ist abhängig von der Differentialgleichung (→ Simulationstechnik). Der prinzipielle Aufruf ist jedoch bei allen Verfahren gleich, nur der dahinterstehende Algorithmus ist jeweils verschieden. Wenn die Differentialgleichung in oben dargestellter Weise vorliegt, kann die numerische Simulation mit folgendem Aufruf erfolgen:

```
T=1000;
tspan=0:T/1000:T;
y0=[5 335];
[t,y]=ode113(@cstr, tspan, y0);
figure(1)
plot(t,y(:,1))
figure(2)
plot(t,y(:,2))
```

Die nötigen Angaben beim Aufruf sind

`@cstr`	Zeiger auf die Funktion, die die Differentialgleichung enthält
`tspan`	Vektor, der den Zeitraum beschreibt, für den die Dgl. simuliert werden soll.
`y0`	Anfangsbedingung

Die von dem Löser zurückgegebenen Werte sind

`t`	Zeitvektor
`y`	Vektor mit den Lösungen für alle Zustände
`y(:,1)`	Zeitreihe für 1. Zustand (hier: Konzentration)

2 Befehle der Control System Toolbox

Darstellung von Systemen

`ss(A,B,C,D)`	Definition eines Systems in Zustandsform
`tf(Z,N)`	Definition eines Systems über Zähler und Nenner der Übertragungsfunktion
`zpk(z,p,k)`	Definition eines Systems über Pole, Nullstellen und Verstärkung
`ss2tf(sys)`	Umwandlung von Zustandsraum- in Übertragunsfunktions-Darstellung
`tf2ss(sys)`	Umwandlung von Übertragungsfunktions- in Zustandsraum-Darstellung
`minreal(G)`	Minimalrealisierung eines Systems (kürzen von Pol-/Nullstellenpaaren mit einstellbarer Toleranz)
`ssdata(sys)`	gibt Zustandsraum-Matrizen A, B, C, D eines Systems zurück
`tfdata(sys)`	gibt Nenner/Zähler eines Systems zurück

Zusammenschaltung von Systemen

`series(sys1,sys2)` bzw. `sys1*sys2`	Reihenschaltung
`parallel(sys1,sys2)` bzw. `sys1+sys2`	Parallelschaltung
`feedback(sys1,sys2)`	Rückführschaltung

System-Kenngrößen

`zero(sys)`	Nullstellen eines Systems
`pole(sys)`	Pole eines Systems
`damp(sys)`	Dämpfungskoeffizienten und Eigenfrequenzen
`dcgain(sys)`	statische Verstärkung
`margin(sys)`	Amplituden- / Phasenreserve

Systemantwort

`step(sys)`	Sprungantwort
`impulse(sys)`	Impulsantwort
`lsim(...)`	Systemantwort auf einen beliebigen Eingang
`initial(...)`	Systemantwort auf Anfangsbedingungen
`ltiview`	Interaktive Systemanalyse

Diagramme

`bode(sys)`	Bode-Diagramm
`margin(sys)`	Bode-Diagramm mit Angabe der Stabilitätsreserve
`nyquist(sys)`	Ortskurve (Nyquist-Diagramm)
`pzmap(sys)`	Pol-Nullstellen-Diagramm
`rlocus(sys)`	Wurzelortskurve ($\rightarrow$*RT2*)

Reglerentwurf im Zeitbereich

`acker(A,b,p)`	Zustandsrückführung mit Polvorgabe für SISO-Systeme `A` Systemmatrix, `b` Eingangsvektor, `p` Pole zurückgeliefert wird der Vektor `k`
`acker(A',c,p)`	Beobachterentwurf mit Polvorgabe für SISO-Systeme ZR: `A` Systemmatrix, c' Ausgangsvektor, `p` Pole zurückgeliefert wird der Vektor `l'`
`place(A,B,p)`	Polvorgabe für MIMO-Systeme, auch Beobachter
`lqr(A,B,Q,R)`	LQ-Reglerentwurf ($\rightarrow$*RT2*)
`kalman(sys,Q,R,N)`	Entwurf eines Kalman-Bucy-Filters (KBF, $\rightarrow$*RT2*)
`estim(sys,L)`	Beobachter aus System und Matrix `l` konstruieren
`reg(sys,K,L)`	Regler aus Polvorgabe und Beobachter
`lqgreg(est,k)`	Regler aus LQ-Regler und KBF

3 Alternativen zu Matlab

Als kostenlose Alternative zu Matlab seien die zwei Programmpakete

Octave `http://www.octave.org/`
Scilab `http://www-rocq.inria.fr/scilab`

erwähnt. Beide Programme stehen für eine Reihe von Betriebssystemen (unter anderem Linux und Windows) zur Verfügung.

Literatur

[1] Control Tutorials for Matlab. `http://www.engin.umich.edu/group/ctm/`.

[2] Tutorial for Control System Toolbox.
`http://www.techteach.no/publications/control_toolbox_matlab/cst4.htm`.

[3] The MathWorks, Inc. *Control System Toolbox User's Guide*, 4th edition, 1999.

[4] The MathWorks, Inc. *Using Matlab*, 4th edition, 1999.