A Collection of MATLAB commands for Control Engineering

March 23, 2013

Here are some Matlab commands relevant to the control of single-input-single-output (SISO) systems.

1 SISO Control

Note: all commands with Ts as an input are for discrete-time systems. Ts is the sampling time here.

1. the s and z elements: s = tf('s'); z = tf('z',Ts); With these you can do, e.g., G = 1/(s+1), G = z^-1/(1-z^-1) etc
2. Construction of transfer functions:
   (a) state space: sys = ss(A,B,C,D) and ss(A,B,C,D,Ts)
   (b) transfer function: sys = tf(num,den), tf(num,den,Ts) and tf(num,den,Ts,'variable','z^-1'); or G = 1/(s+1) when s is properly defined as discussed in [1].
   (c) construction from poles and zeros: sys = zpk(z,p,k)
3. Obtaining information of a system:
   (a) [A,B,C,D] = ssdata(sys) and [A,B,C,D,Ts] = ssdata(sys)
   (b) [num,den] = tfdata(sys,'v')
   (c) [z,p,k] = zpkdata(sys,'v')
4. Converting from one system representation to another:
   (a) [A,B,C,D] = tf2ss(num,den)
   (b) [num,den] = ss2tf(A,B,C,D)
   (c) [z,p,k] = ss2zp(A,B,C,D)
   (d) [z,p,k] = tf2zp(num,den)
   (e) and zp2tf, zp2ss
   (f) sys_tf = tf(sys_ss)
5. Connecting transfer functions:
   (a) series(sys1,sys2) gives sys1*sys2 and parallel(sys1,sys2) gives sys1+sys2
   (b) in many cases you can directly use sys1+sys2 and sys1*sys2
   (c) you can also use num = conv(num1,num2), den = conv(den1,den2), and then sys = tf(num,den) to get sys1*sys2
6. Building closed loops:
   (a) feedback(L,1,-1) gives $T = L/(1 + L)$
   (b) feedback(1,L,-1) gives $S = 1/(1 + L)$
   (c) feedback(P,C,-1) gives $P/(1 + PC)$
7. Simulating the response of a linear system:

(a) lsim(T,u,t) gives $y = Tu$

(b) impulse(T) gives the impulse response of T

(c) step(T) gives the step response of T

(d) you can also manually write a script to simulate the loop. For instance, for a discrete-time system $x(k+1) = Ax(k) + bu(k)$, you can write a 'for' loop:

```
for ii = 1:N
.
.
.
x_k1 = A*x_k + B*u_k;
y_k = C*x_k + D*u_k;
.
.
.
x_k = x_k1;
end
```

8. System analysis:

(a) pole(sys) and zero(sys) give the poles and zeros of a SISO transfer function. Warning: zeros(2,1) gives [0,0]^T;

(b) pole(sys) gives the same result as [A,B,C,D] = ssdata(sys); eig(A)

(c) pzplot(sys) and pzmap(sys) give the pole-zero map of the system

(d) bode and bodeplot give the bode plot. In many cases we prefer to use Hz as unit of the xaxis, you can do this from at least two ways

i. >>figure;p=*bodeplot*(H);
>>setoptions(p,'frequnits','Hz');

ii. change the default bodeoption

```
bode_opt = bodeoptions;
bode_opt.FreqUnits = 'Hz';

figure, bodeplot(sys,bode_opt);
```

iii. use *set(cstprefs.tbsprefs,'FrequencyUnits','Hz')* when you start MATLAB. Then whenever you do bodeplot(sys), the frequency units will be in Hz. You can also write this to your *startup.m* file[1] so that it is loaded by default.

(e) different from bode and bodeplot, which give the magnitudes and phases of a transfer function, freqresp(sys,w) gives the complex value frequency response at the frequencies w

9. be careful with the transfer function construction

```
P = tf(1,[1 -3]);
C = tf(6,[0.2 1]);
% incorrect manner
Tb = P*C/(1+P*C); % results in a fourth-order system
figure, step (Tb,13)
% correct manner
Tc = feedback(P*C,1); % results in a second-order system
figure, step (Tc,13)
```

10. *Some more notes for bodeplot using bodeoption handles.*

```
P = bodeoptions;
P.PhaseVisible = 'off';
```

[1]Search startup.m to see where this file is located on your operation system.

```
P.FreqUnits = 'Hz';
P.YLim = {[-50,100];[-360,0]}; %{maglimits [-50,100]; phaselimits [-360,0]}
P.YLimMode = {'manual';'auto'}; %{manual maglimits mode; auto phaselimits mode}
figure;
h = bodeplot(Pn_d,P);
```