EEE 480 Lab (3/4/99)

MATLAB 101

Note: This command list was written for MATLAB version 5. Most of these commands are valid in version 4, but not all. Please check the help for each command before using them.

MATLAB Commands to Know and Love

The most important command is the ***help*** command. Type 'help' alone to see a list of topics. Then type '**help** *<command>*' to see help for that command. For example, to find out about the 'step' command, type '**help step**' *<enter>*.

• **general commands:**	clear, who, whos, clc, diary, save, load, exit, sprintf
• **plotting/graph commands:**	figure, plot, subplot, semilogx, zoom, grid, title, linspace, hold, gtext, legend
• **system construction commands:**	tf, feedback, series, parallel, conv
• **system analysis commands:**	step, rlocus, rlocfind, bode, nyquist, max, min

MATLAB Hints and Tips

- Press the up-arrow key to go through the command history.
- Type some letters followed by the up arrow to find previous commands
- When running a bunch of commands, use a script (see below).
- To print your command history, select the text with the mouse, then select |Edit|Copy|. Paste this into your favorite word-processor to edit and print.

Writing scripts (.m files)

Click on |**file| new**|**M-file|** or type 'edit' in the command window to create a new file. **Note:** in version 4 of MATLAB, this uses the built-in Windows notepad editor. Version 5 has a new MATLAB editor (that's a lot nicer).

A script is essentially a list of MATLAB command you would execute in the command window in order.

Defining functions

A function is a special script (.m) file. Here is the format:

```
function [output_var_1, output_var_2, …] = function_name(input_var_1, input_var_2, … )
%
% Function information (comments)
%

…
function body
…
```

For example, let's define a function that returns the maximum amplitude of a system's step response:

```
function max_amp = max_step_amp( num, den )
%
% This function returns the maximum amplitude of a step response for the system:
%       H(s) = num / den
%
sys= tf(num, den);            % define the system as a transfer function
t= linspace( 0,20,200 );      % We will only look up to time=20s
[y, t]= step( sys );          % Compute the step response
max_amp= max(y);              % Find (and return) the maximum amplitude
```

Automatically labeling your MATLAB plots

The script below shows how to create and label your plots on the fly. The key is the **sprintf** (**s**tring **print** **f**unction) command.

```
% label_demo
%       - Demonstrates labeling plots on-the-fly

figure                                          % create a new figure
t= linspace(0,15,200);
for zeta= 0.0:0.2:1.2
        sys= tf( [1], [1 2*zeta, 1] );
        step( sys, t );
        hold on;                                % overlay all plots
        str= sprintf( 'zeta= %3.1g', zeta );    % create the text label 'on-the-fly'
        gtext( str );                           % let user click to position text
end
grid on                                         % set up grid and title
title( '2nd order system step response for varying \zeta' );
```

Forming systems in MATLAB

The **sys** (system) structure in MATLAB v5 is very powerful, and it allows you to form complicated systems by joining together simpler systems. Below is an example script of how we can compute the closed-loop transfer function for a system from the plant and controller transfer functions. The system we are looking at looks like this:

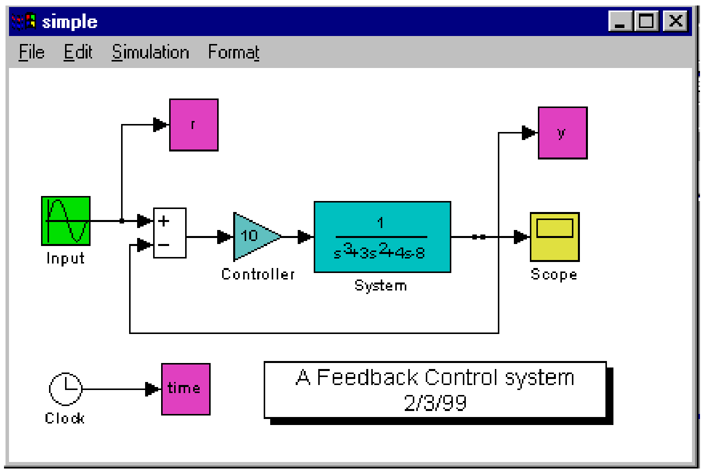


Figure 1-Feedback system example

```
%
% sys_demo
%   - Demonstrates creating feedback systems from individual blocks
%

Pnum= 1;                                  % The plant is P = 1 / (s^2+4s+8)(s-1)
                                          % Note that it is UNSTABLE!
Pden= conv( [1 2+j*2], [1 2-j*2] );       % conv is handy for multiplying two polynomials
Pden= conv( Pden, [1 -1] );
Psys= tf( Pnum, Pden );                   % The tf command specifies transfer function systems

figure;
rlocus( Psys );                           % Show the root-locus of this system

Ksys= tf( 10, 1 );                        % We can stabilize this system with a gain

OLsys= series( Ksys, Psys );              % The Open-Loop system is K-P in series
```

```
CLsys= feedback( OLsys, tf(1,1) );       % Use the feedback command to compute the
                                         % Closed-loop transfer function
figure;
step( CLsys );                           % See the step response of the controlled system
```

Using ‘freqs’ to Make Bode Plots

This will give you some insight into how the bode command works.

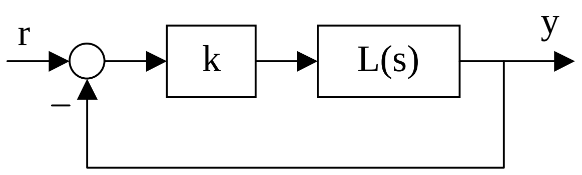
```
%
% freqs_demo
%   - freqs-command demonstration
%
num= 1;                                  % system is L(s)= 1 / (s+1)
den= [1 1];
w= logspace( -1, 1, 400 );               % frequency range to plot
h= freqs( num, den, w );                 % find frequency response
mag= abs(h);                             % Calculate magnitude
mag_db= 20*log10(mag);
phase= unwrap(angle(h));                 % Calculate phase
phase_deg= (180/pi)*phase;

figure                                   % Draw the plots
subplot( 2, 1, 1 );                      % Magnitude plot
semilogx( w, mag_db );
grid on
title( ’Bode plot of 1/(s+1) using freqs’ );
ylabel( ’Magnitude (dB)’ );

subplot( 2, 1, 2 );                      % Phase plot
semilogx( w, phase_deg );
grid on
xlabel( ’Frequency (rad/sec)’ );
ylabel( ’Phase (deg)’ );
```

Using the rlocus and rlocfind Commands

The ***rlocus*** (root-locus) command works with the negative-feedback structure shown below:



Remember: the ***rlocus*** command is given the **open-loop** transfer function, L(s), **NOT** the closed-loop transfer function. This is a common error. Don’t let it happen to you.

The example below demonstrates how to use this command. We will also use the ***conv*** (convolution) command to multiply together two polynomials. ***rlocfind*** is then used to let you pick your closed-loop pole locations.

```
%
% rloc_demo
%   - Root-Locus Demonstration
%
num= conv( [1 3], [1 4] );             % open-loop system is
den= conv( [1 0], [1 -1] );            %   L(s)= (s+3)(s+4) / s(s-1)

sys= tf( num, den );
figure
rlocus( sys );                         % Draw the root-locus
```

```
hold on
k= 10;                               % Feedback gain
rlocus( sys, k );                    % Plot the positions of the
                                     % Closed-loop poles

[k poles]= rlocfind( sys );          % Let's you pick the location of your closed-loop poles
                                     % Then displays the results
fprintf( 'Your selected closed loop poles are:' );
poles
fprintf( 'The necessary gain to achieve this is k=%g\n', k );
```

Step Response Analysis Macro

We will now present a macro that will automatically generate the Figure below:

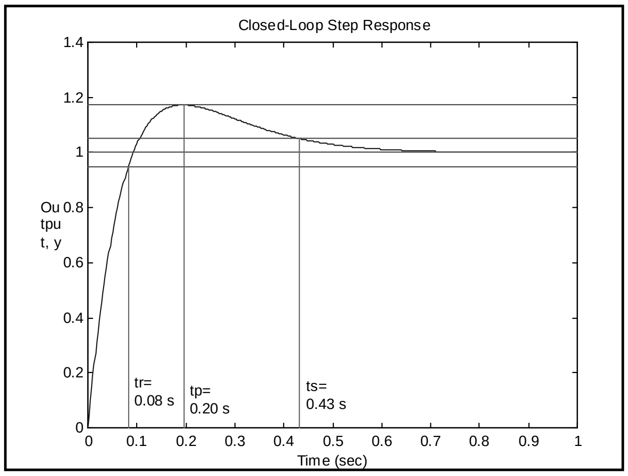


Figure 2-Step Response Plot

This macro will compute the step response, calculate the rise-time, peak-time, and settling time, and draw these lines in. All the user will have to do is to select the positions of the labels for tr, tp, and ts.

Here is the script. It looks a little longer than the previous ones, but we're just doing the same thing a few times.

```
%
% step_demo
%   - Demonstrates how to automatically analyze
%     and label a step-response plot.
%

%%% Plant definition
num= [1 5];                          % (open-loop) plant is
den= conv( [1 0], [1 -1] );          %   L(s)= (s+3) / s(s-1)
Psys= tf( num, den );                % Form plant system (block)

%%% Controller and closed-loop system
k= 20;                               % Just a proportional controller
Ksys= tf( k, 1 );                    % Form controller system (block)
OLsys= series( Ksys, Psys );         % Compute open-loop system
CLsys= feedback( OLsys, tf(1,1) );   % Compute closed-loop system
t= linspace( 0, 1, 400 );            % Set up the time vector
[y t]= step( CLsys, t );             % Compute step response
dcg= dcgain( CLsys );                % Compute the d.c. gain of the system
yp= max( y );                        % Find peak value of y (output)

figure                               % Plot step response
plot( t, y );
```

```
hold on
title( 'Closed-Loop Step Response' );    % Good practice to ALWAYS label our plots
xlabel( 'Time (sec)' );
ylabel( 'Output, y' );

%%% Draw horizontal Lines
plot( [0 1], [1.05 1.05]*dcg, 'r:' );    % Horiz. line at 1.05
plot( [0 1], [1 1]*dcg, 'r:' );          % Horiz. line at 1.0
plot( [0 1], [0.95 0.95]*dcg, 'r:' );    % Horiz. line at 0.95
plot( [0 1], [yp yp], 'r--' );           % Horiz. line at peak value

%%% Peak time
max_idx= min(find(y==yp));               % Use find command to find index where yp occurs
tp= t(max_idx);                          % tp is the time at this index in the time vector
plot( [tp tp], [0 yp], 'm--' );          % Draw vertical line at tp
str= sprintf( 'tp=\n%3.2f s', tp );      % Let user label tp
gtext( str );

%%% Rise time
yr= 0.95;                                % Look for 5% rise-time
rise_idx= min(find(y>=yr));              % Search for the index
tr= t(rise_idx);                         % to obtain the corresponding rise-time
plot( [tr tr], [0 yr], 'm--' );          % Draw vertical line at tr
str= sprintf( 'tr=\n%3.2f s', tr );      % Let user label tr
gtext( str );

%%% Settling time
settle_idx= max(find(y>=1.05|y<=0.95));  % Look for 5% settling time (both directions)
ys= y(settle_idx);                       % Search for the index
ts= t(settle_idx);                       % to obtain the corresponding settling-time
plot( [ts ts], [0 ys], 'm--' );          % Draw vertical line at ts
str= sprintf( 'ts=\n%3.2f s', ts );      % Let user label ts
gtext( str );

hold off                                 % And we're done!
```

SIMULINK Survival Guide

Useful SIMULINK Blocks (used in EEE480)

• **Sources:**	constant, signal generator, step, sine wave, clock
• **Sinks:**	scope, to workspace
• **Linear:**	gain, sum, integrator, transfer function
• **Connections:**	mux, demux

Basic Hints and Tips

- Drag a block from the SIMULINK library to your block diagram window and double click on it to modify its parameters. For example, a step input has three parameters (usually, the **step-time** should be **0**).
- Drag a block with the right-mouse button to make a copy of it
- When you have spare time, check out the SIMULINK demos to see some of the neat things you can do with it.

Getting rid of choppy graphs

– Click on |Simulation|Parameters| in the SIMULINK menu (type ‘simulink’ from the Command window to call it up).
– Set the **Max step size** to **0.1** (or whatever is appropriate in your case)
– Set the **Relative Tolerance** and **Absolute Tolerance** both to **1e-6**

Printing plots

By now, you are probably wondering why there isn’t a ‘print’ button on the scope. That’s a good question. Here’s how we recommend you do your plots:

– Use ‘*Send to Workspace*’ blocks to send output signals to the workspace
– Use a *clock* (in *sources*) to obtain the time array (See Figure 1 below)
– Run the simulation
– Go back to the command window and execute `plot(time,signal)` – e.g. `plot(time,y)` - to display your plot
– Print as normal

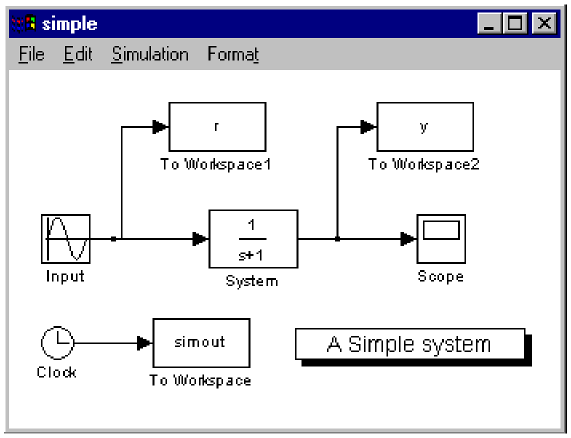


Figure 3-Printing Plots from SIMULINK

Advanced SIMULINK Hints and Tips

- Welcome to the Super-VGA age! Use colors to organize a system: e.g. make input blocks green, disturbance blocks red, plant blocks cyan, and controller blocks light blue. To change the color, select a block and then |Format|Background Color|.

- Add titles to your SIMULINK diagrams by double-clicking the left-mouse-button on an empty spot and typing in some text. You should label your diagrams with a descriptive title, your name, and the date you created it.

- For more complicated systems, you may use *subsystems* to organize your simulation. To use these, select a group of blocks you wish to place in a subsystem, and select |Edit|Create Subsystem|.