Some Useful MATLAB Commands

tf – converts system representation from transfer function to “system” format. This format provides an efficient way to store and manipulate the system model, but does not allow direct access to the system’s numerator and denominator or poles and zeros. The syntax to convert from transfer function to “system” format is:

\[ sys = tf(num, den); \]

tfdata – converts system representation from “system” format back to transfer function format. The syntax (for single-input, single-output systems) is:

\[ [num, den] = tfdata(sys, 'v'); \]

series – forms the series combination of 2 blocks to produce the equivalent transfer function. The output of block 1 is the input to block 2. The syntax for the function in transfer function and in “system” formats are:

\[ [num, den] = series(num1, den1, num2, den2); \]
\[ sys = series(sys1, sys2); \]

parallel – forms the parallel combination of 2 blocks to produce the equivalent transfer function. The outputs of block 1 and block 2 are added together. The same input is applied to both block 1 and block 2. The syntax for the function in transfer function and in “system” formats are:

\[ [num, den] = parallel(num1, den1, num2, den2); \]
\[ sys = parallel(sys1, sys2); \]

feedback – forms the closed-loop system that has block 1 in the forward path and block 2 in the feedback path. The default configuration is for negative feedback. In the syntax example shown below, the −1 for negative feedback is shown explicitly. For positive feedback, a +1 would be used in the function’s input argument list. For unity feedback, \( num2 = den2 = 1 \). The syntax for the function in transfer function and in “system” formats are:

\[ [num, den] = feedback(num1, den1, num2, den2, -1); \]
\[ sys = feedback(sys1, sys2, -1); \]

cloop – forms the closed-loop system when unity feedback is used. This function is obsolete, but is still available in most MATLAB versions. It can only be used when the block in the feedback path is unity, and it can only be used with the system model expressed in transfer function form, not in “system” format. The syntax for the function is:

\[ [num, den] = cloop(num1, den1, -1); \]
**rlocus** – computes and plots the root locus for a system, using an internally generated set of gain values. A user-specified vector of gain values can be included as an input argument if desired. Output arguments of closed-loop poles and the corresponding gain values can also be obtained. In this case, no plot is made. If the root locus plot for negative gain values is desired, a minus sign is placed in front of *num* or *sys*. The syntax for the function in transfer function and in “system” formats are:

```
rlocus(num, den);
rlocus(sys);
```

**rlocfind** – computes the gain and the corresponding closed-loop pole locations for a specified point on the root locus. A crosshair is made available to select a point on the root locus plot. The gain corresponding to that point is returned, as well as all the closed-loop poles for that gain. The syntax for the function in transfer function and in “system” formats are:

```
[K, poles] = rlocfind(num, den);
[K, poles] = rlocfind(sys);
```

Instead of graphically selecting a point, one or more desired closed-loop pole locations can be input to the function. The gains and closed-loop poles corresponding to those specified pole locations are returned. No error message is given if the specified pole locations are not actually on the root locus for the system. The syntax for the function in transfer function and in “system” formats are:

```
[K, poles] = rlocfind(num, den, P);
[K, poles] = rlocfind(sys, P);
```

**bode** – computes the magnitude (absolute value) and phase (degrees) of a system evaluated on the $j\omega$ axis. If no frequency vector is provided as an input argument, MATLAB selects a set of frequency values. If no output arguments are specified, the Bode plots are automatically made. If output arguments are specified, no plots are made. Specifying a frequency vector and output arguments, the syntax for the function in transfer function and in “system” formats are:

```
[mag, ph] = bode(num, den, w);
[mag, ph] = bode(sys, w);
```

If the transfer function form is used with a single-input, single-output (SISO) system, the magnitude and phase arrays are column vectors, with the number of rows being equal to the number of frequency values. If the “system” form is used, the arrays that are returned are 3-dimensional and cannot directly be plotted in MATLAB. In order to plot the magnitude and phase that are returned when
using \( [mag, ph] = \text{bode}(sys, w) \); the following commands can be performed:

\[
\text{mag}1(:, 1) = \text{mag}(1, 1,:); \\
\text{ph}1(:, 1) = \text{ph}(1, 1,:);
\]

**logspace** – generates a row array of elements that are spaced logarithmically, that is, the ratio of two consecutive elements is equal throughout the array. This is useful for generating a frequency vector for use with the bode function. The first two input arguments are the powers of 10 corresponding to the lowest and highest frequencies in the array. The third input argument is the total number of elements in the array. This number should be selected based on the number of frequency values that want in each decade of frequency. If the frequency vector is to contain values from \( \omega_{\text{low}} = 10^{N_1} \text{ rad/sec} \) to \( \omega_{\text{high}} = 10^{N_2} \text{ rad/sec} \) with \( N_3 \) values per decade of frequency, the syntax for generating the frequency vector is:

\[
w = \text{logspace}(N_1, N_2, 1 + N_3 \times (N_2 - N_1));
\]

For example, to generate frequencies from 0.001 \( \text{r/s} \) to 10 \( \text{r/s} \) with 100 points per decade (the value that I use), the command would be

\[
w = \text{logspace}(-3, 1, 401);
\]

**semilogx** – used for making Bode plots when the magnitude and phase arrays are returned from the bode function. A linear scale is used for the vertical axis and a logarithmic scale is used for the horizontal axis. To plot the magnitude in decibels and the phase in degree on the same plot, the following instruction can be used:

\[
\text{semilogx}(w, 20 \times \log_{10}(\text{mag}), w, \text{ph}), \text{grid}
\]

**freqresp** – similar to bode function, but can only be used with the system model in “system” format. The frequency response is returned in one 3-dimensional array, as a set of complex numbers. For plotting, the array must be converted as shown in the bode function. The syntax and plotting commands are:

\[
H = \text{freqresp}(sys, w);
\]

\[
H1(:, 1) = H(1, 1,:);
\]

\[
\text{semilogx}(w, 20 \times \log_{10}(\text{abs}(H1))), \text{grid}
\]

\[
\text{semilogx}(w, \text{unwrap}(\text{angle}(H1)) \times 180/\pi), \text{grid}
\]

**margin** – computes the following measures of relative stability based on the frequency response of a system: gain margin (\( \text{Gm} \)), phase margin (\( \text{Pm} \)), phase crossover frequency (\( \text{Wcg} \)), gain crossover frequency (\( \text{Wcp} \)). These measures provide information about how much perturbation to a system can be tolerated without becoming unstable and are often used as design specifications. If no frequency vector is included as an input argument, the values are computed analytically from the transfer function. If a frequency vector is included, the values are obtained by searching the magnitude and phase arrays at those frequencies. If no output arguments are specified, Bode plots are made with the
gain and phase margins shown on the plots. The syntax for the function in
transfer function and in “system” formats are:

\[
[Gm, Pm, Wcg, Wcp] = \text{margin}(\text{num}, \text{den});
\]

\[
[Gm, Pm, Wcg, Wcp] = \text{margin}(\text{sys});
\]

\[
[Gm, Pm, Wcg, Wcp] = \text{margin}(\text{mag}, \text{ph}, w);
\]

**nyquist** – computes the real and imaginary components of the frequency re-
sponse of a system (rather than magnitude and phase). A frequency vector is
an optional input argument. If no output arguments are specified, a plot of the
imaginary part vs. the real part of the frequency response is plotted, and the
\(s = -1 + j0\) point is indicated on the plot. If output arguments are specified,
the real and imaginary parts are returned, and no plot is made. The syntax for
the function in transfer function and in “system” formats are:

\[
[\text{re}, \text{im}] = \text{nyquist}(\text{num}, \text{den}, w);
\]

\[
[\text{re}, \text{im}] = \text{nyquist}(\text{sys}, w);
\]

**step** – computes the unit step response for a system. A time vector is an optional
input argument. The vector of values of the step response is an optional output
argument. If no output argument is specified, a plot of the response is made.
The syntax for the function in transfer function and in “system” formats are:

\[
\text{c} = \text{step}(\text{num}, \text{den}, t);
\]

\[
\text{c} = \text{step}(\text{sys}, t);
\]

Making use of the relationship between the step and ramp functions, an easy
way to get the ramp response is the following:

\[
\text{c}_{\text{ramp}} = \text{step}(\text{num}, [\text{den} \ 0], t);
\]

\[
\text{sys1} = \text{tf}(\text{num}, [\text{den} \ 0]);
\]

\[
\text{c}_{\text{ramp}} = \text{step}(\text{sys1}, t);
\]

**lsim** – computes the response of a system to a user-specified input vector. A
vector of equally-spaced time points is an input argument, as is the vector of
input values. The system output is an optional output argument. If no output
argument is specified, a plot of the output signal is made. The syntax for the
function in transfer function and in “system” formats are:

\[
\text{c} = \text{lsim}(\text{num}, \text{den}, u, t);
\]

\[
\text{c} = \text{lsim}(\text{sys}, u, t);
\]

**other possibly useful functions** – roots, pole, tzero, damp, angle, abs, poly,
polyval, pzmap, find, unwrap, linspace, conv, size, length, real, imag, sum, prod,
eval.