Useful MATLAB Functions

This document/tutorial was compiled by Dr. Joanna C. Hinks as a resource for UB's MAE 340 class. It illustrates how to use some of the most important functions for dynamic systems analysis.

Additional resources:

MATLAB's built-in help files contain many examples and tutorials that illustrate how to use various functions and accomplish simple tasks. In the main taskbar of the MATLAB window, select Help → Product Help.

Many online tutorials, guides, and examples are available, including those at http://www.mathworks.com/matlabcentral/

Many Dynamic Systems textbooks also contain tutorials on basic MATLAB usage, often in an appendix. In particular, "System Dynamics" by Ogata and "Dynamic Modeling and Control of Engineering Systems" by Kulakowski, Gardner, and Shearer each contain such appendices.

While students are encouraged to seek out available resources, the code you submit should be your own.

bode

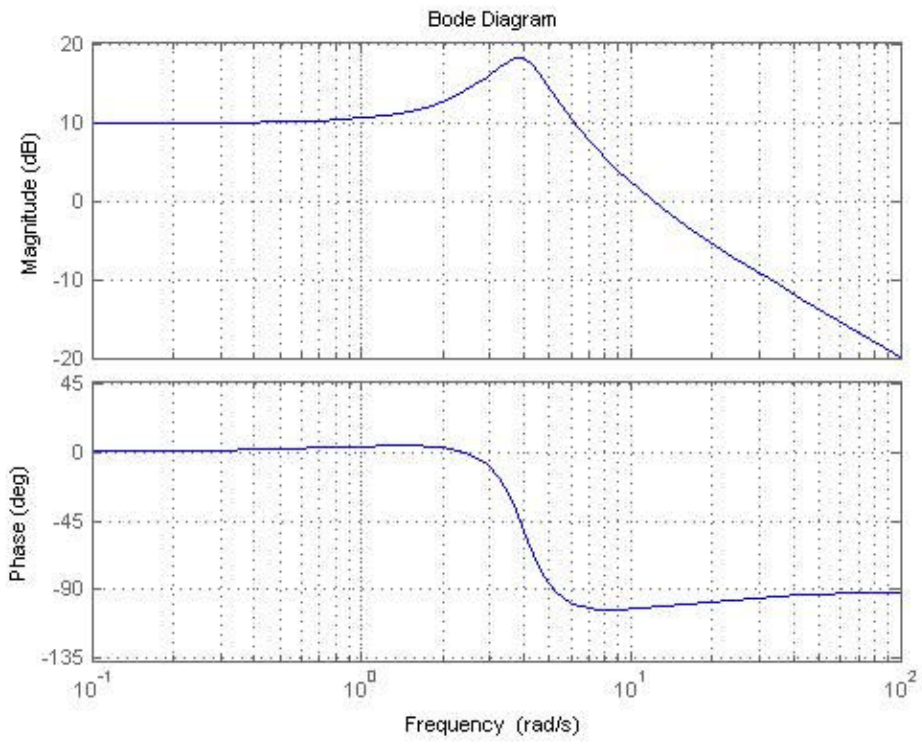
Generate a Bode plot for a given system in state space or transfer function form

Generate a bode plot for the system described by the transfer function

$$TF = 10\frac{s+5}{s^2+2s+16}$$

Also calculate the bandwidth of the given system (frequency range where the magnitude is within 3 dB of the maximum magnitude).

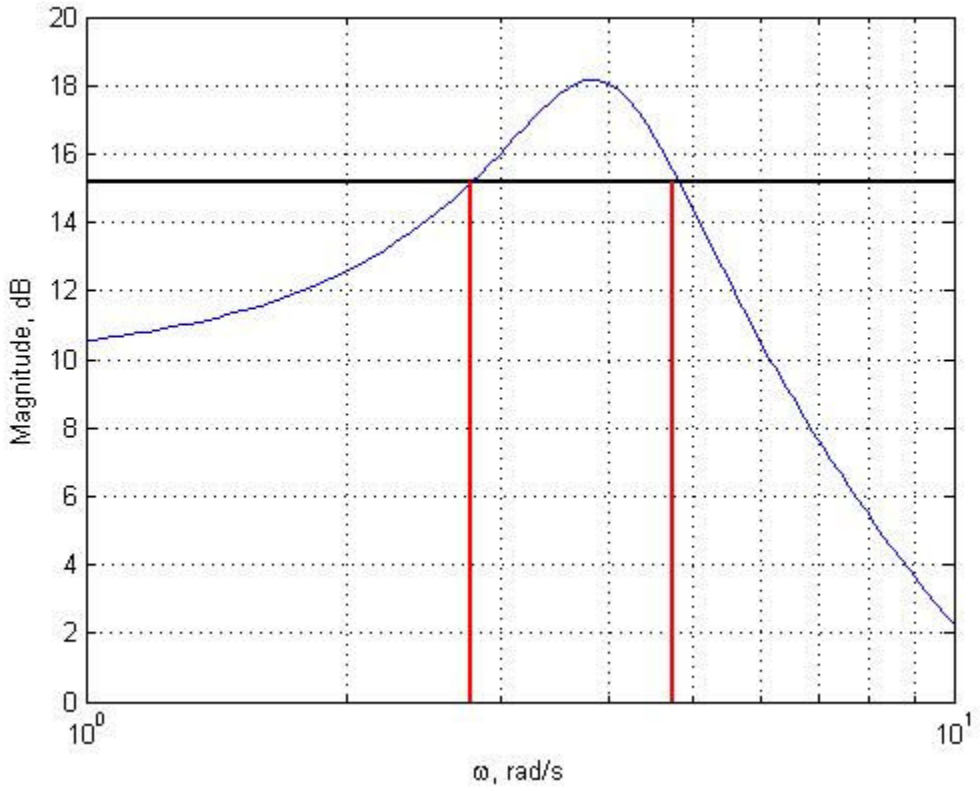
```
% Transfer function numerator and denominator:
num = 10*[1 5];     den = [1 2 16];

% One set of equivalent state space matrices (others possible):
A = [-2, -16; 1, 0]; B = [1; 0]; C = [10, 50]; D = 0;

% Specify the transfer function object:
sys_tf = tf(num,den);

% Specify the state space object:
sys_ss = ss(A,B,C,D);

% Plot the Bode plot in any of 4 equivalent ways:
bode(num,den);
bode(A,B,C,D);
bode(sys_tf);
bode(sys_ss);

% Display grid lines
grid on;
```



Now look at the magnitude part more closely instead of having MATLAB automatically generate the plot.

```
% Generate 100 logarithmically spaced frequency points between 10^0 and
% 10^1:
w = logspace(0,1,100);

% Compute the frequency response:
[mag,phase] = bode(sys,w);

% Convert outputs to 2D arrays instead of 3D
mag = squeeze(mag);
w = squeeze(w);

% Convert magnitude to dB:
magdB = 20*log10(mag);

% Find the maximum magnitude:
max_mag = max(magdB);

% Find the magnitude threshold that defines the bandwidth:
mag_thresh = max_mag - 3;

% Find approximate frequencies that define the bandwidth (not exact,
% because frequency points are not spaced closely enough)
wband = w(magdB >= mag_thresh);
wmin = min(wband); wmax = max(wband);
```

```
figure(2);
semilogx(w,magdB); grid on;
line([min(w), max(w)],[mag_thresh,
mag_thresh],'Color','k','LineWidth',2);
line([wmin wmin],[0, mag_thresh],'Color','r','LineWidth',2);
line([wmax wmax],[0, mag_thresh],'Color','r','LineWidth',2);
xlabel('\omega, rad/s');
ylabel('Magnitude, dB');
```



conv

Multiply two polynomials together.

Multiply the two polynomials:

$$a(s) = s^5 + 4s^2 + s + 3$$

and

$$b(s) = s^2 - 2s + 1$$

```
% Enter the two polynomials. Don't forget zero coefficients:
a = [1 0 0 4 1 3];
b = [1 -2 1];
% Calculate the product:
c = conv(a,b)
```

Result:

```
c =

     1    -2     1     4    -7     5    -5     3
```

So

$$a(s)b(s) = s^7 - 2s^6 + s^5 + 4s^4 - 7s^3 + 5s^2 - 5s + 3$$

deconv

Perform polynomial division.

Divide the polynomial:

$$c(s) = s^7 - 2s^6 + s^5 + 4s^4 - 7s^3 + 5s^2 - 5s + 3$$

by the polynomial

$$b(s) = s^2 - 2s + 1$$

```
% Enter the two polynomials. Don't forget zero coefficients, if
% applicable:
c = [1 -2 1 4 -7 5 -5 3];
b = [1 -2 1];
% Calculate the product:
a = deconv(c,b)
```

Result:

```
a =

     1     0     0     4     1     3
```

So

$$a(s) = s^5 + 4s^2 + s + 3$$

Note that `deconv` is the inverse of the `conv` function.

`for`

Repeat a set of calculations iteratively, often in a way that builds on previous results.

Find the roots of the polynomial

$$\lambda^3 + 6\lambda^2 + 2\lambda + K = 0$$

for all the integer values of K between -2 and 2. Also count the number of roots with non-negative (positive or zero) real parts within this set.

```
% Specify values of K from -2 to 2 in increments of 1
K = -2:1:2;

% Create a variable to store the roots. Fill it with zeros temporarily.
r = zeros(3,length(K));

% Counter for non-negative real parts (initialize at zero):
nonneg = 0;

for k = 1:1:length(K)
    % k goes from 1 to 5 (the length of K) in increments of 1.

    % Find the roots for the kth value of K
    rk = roots([1 6 2 K(k)]);

    % Store the roots in the kth colum of variable r:
    r(:,k) = rk;

    % Count the number of roots in rk with non-negative real parts
    % and add that number to the previous total:
    nonneg = nonneg + sum(real(rk)>=0);

end
```

Results:

```
r =
  -5.5771   -5.6119         0   -5.6788             -5.7111
  -0.8465   -0.6587   -5.6458   -0.1606 + 0.3877i   -0.1444 + 0.5739i
   0.4236    0.2705   -0.3542   -0.1606 - 0.3877i   -0.1444 - 0.5739i

nonneg =   3
```

The "`for`" loop repeats once for each value in the index vector `k`. For each value of `k`, it performs all of the calculations until it reaches the "`end`" statement, and then goes back to the beginning. Note how the variable `nonneg` may change in each iteration.

`if/else`

Calculate something or perform an action if some condition is met; otherwise do something else.

Display a warning in the MATLAB command window if the system with the following characteristic equation is unstable:

$$\lambda^4 + 3\lambda^3 + 5\lambda^2 + \lambda 3 + 2 = 0$$

```
% Specify a polynomial
x = [1 3 5 3 2];

% Calculate its roots:
r = roots(x); % This will be a 4 x 1 vector

% Test for instability and display result:
if any(real(r)) >= 0
    disp('The system is unstable')
else
    disp('All roots are stable')
end
```

Output:

```
The system is unstable
```

The statement after "`if`" must be a logical expression such as "`x < 0`" or "`y==3`". If the logical expression is true, the "`if`" action is taken. If it is false, the "`else`" action is taken. `if/else` loops can be nested to test multiple conditions.

initial

Calculate the state space dynamic system response for a given initial state vector.

Given the state-space model described by:

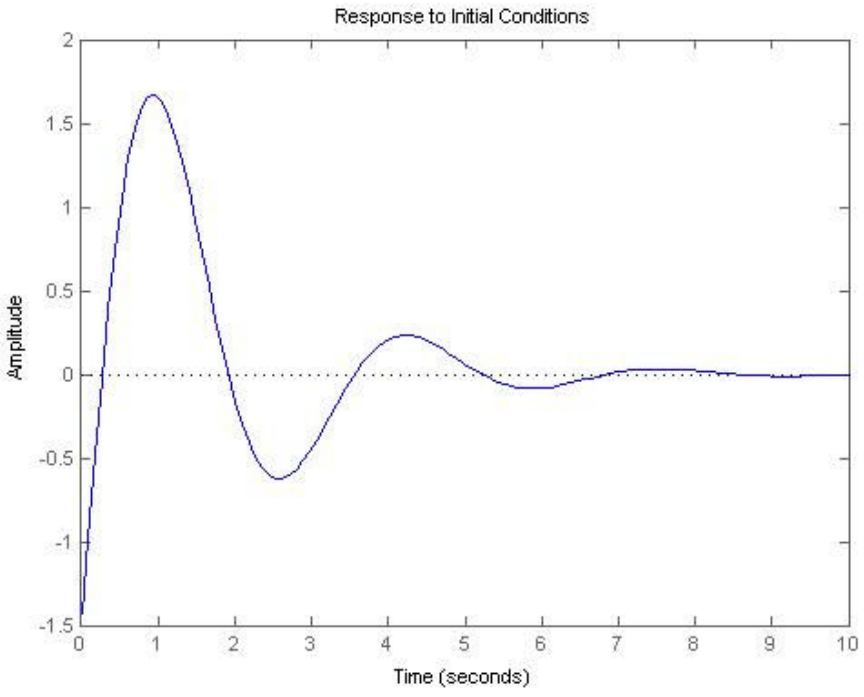
$$\frac{d}{dt}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -4 & -1.2 \end{bmatrix}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 3 & 0 \end{bmatrix}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} + [0]u(t)$$

Plot the system output ($y(t)$) assuming no inputs ($u(t) = 0$) and the initial states:

$$z_1(0) = -0.5, \quad z_2(0) = 2$$

```
% State space matrices
A = [0, 1; -4, -1.2]; B = [2; 3]; C = [3, 0]; D = 0;

% Create state space system object:
sys = ss(A,B,C,D);

% Vector of initial states:
z1_0 = -.5;  z2_0 = 2;
z0vec = [z1_0; z2_0];

% Calculate and plot free response (homogeneous solution):
initial(sys, z0vec);
% Equivalent function call using state space matrices:
initial(A,B,C,D,z0vec);
```



If the values of the states are of interest as well as the output, one can call initial with explicit outputs:

```
[y,t,z] = initial(sys,z0vec);
```

The function outputs are `y` (system output history), `t` (time vector corresponding to `y` and `z`) and `z` (state histories). **There are NO plots generated automatically in this case, but one can create them from the function ouputs.**

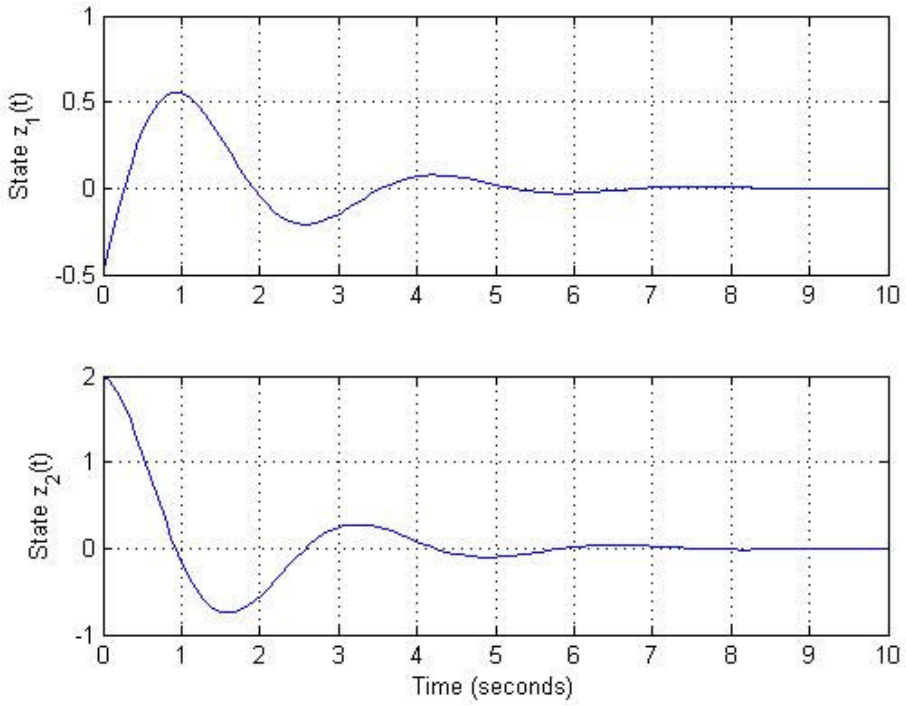
The following block of code reproduces the plot of `y(t)` generated by `initial` without outputs (as shown on previous page).

```
figure(1);
plot(t,y);
line([0 t(end)],[0 0],'LineStyle',':','Color','k');
xlabel('Time (seconds)');
ylabel('Amplitude');
title('Response to Initial Conditions');
```

To plot the states instead, try something like this:

```
figure(2);
subplot(2,1,1);
plot(t,z(:,1)); grid on;
ylabel('State z_1(t)');
subplot(2,1,2);
plot(t,z(:,2)); grid on;
ylabel('State z_2(t)');
xlabel('Time (seconds)');
```

The resulting set of plots looks like:



lsim

Simulate the time response of a dynamic system to specified inputs.

The function lsim is the most general and powerful tool for simulating dynamic systems. It can replicate the behavior of initial or step, as well as many other types of responses. lsim computes and simulates both the homogeneous and particular solutions (free and forced response).

Simulate the first 15 seconds of the response of the state space system:

$$\frac{d}{dt}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -3 & -0.5 \end{bmatrix}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} + \begin{bmatrix} 3 \\ 1 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} + [0]u(t)$$

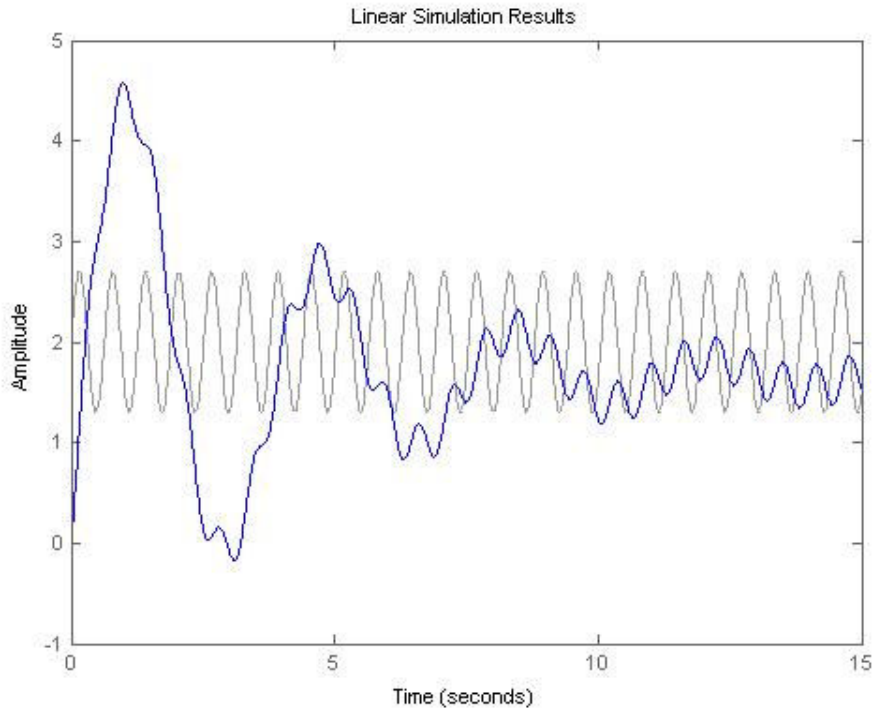
to the input

$$u(t) = 2 + 0.7\sin(10t)$$

with the initial conditions

$$z_1(0) = 0, \quad z_2(0) = -1$$

```
% State space matrices
A = [0, 1; -3, -.5]; B = [3; 1]; C = [1, 0]; D = 0;

% Create state space system object:
sys_ss = ss(A,B,C,D);

% Find equivalent transfer function:
[num,den] = ss2tf(A,B,C,D);
sys_tf = tf(num,den);

% Initial conditions
z1_0 = 0;
z2_0 = -1;
z0 = [z1_0; z2_0];

% Time vector
t = (0:0.01:15)';

% Input
u = 2*ones(length(t),1) + 0.7*sin(10*t);

% Simulate and plot response:
lsim(sys_ss,u,t,z0);
lsim(A,B,C,D,u,t,z0);
lsim(sys_tf,u,t);
lsim(num,den,u,t);
```



The gray curve is the input $u(t)$, and the blue curve is the output $y(t)$. **Note that if `lsim` is called for a system in transfer function form, initial conditions of zero are assumed (so the result will not, in general, be the same).** Note also the two different frequencies of oscillation in the output curve: the slower frequency comes from the system natural characteristic equation, and the faster frequency comes from the input frequency.

As with initial and step, the function can return histories of the output and states.

```
[y,t,z] = lsim(sys_ss,u,t,z0);
```

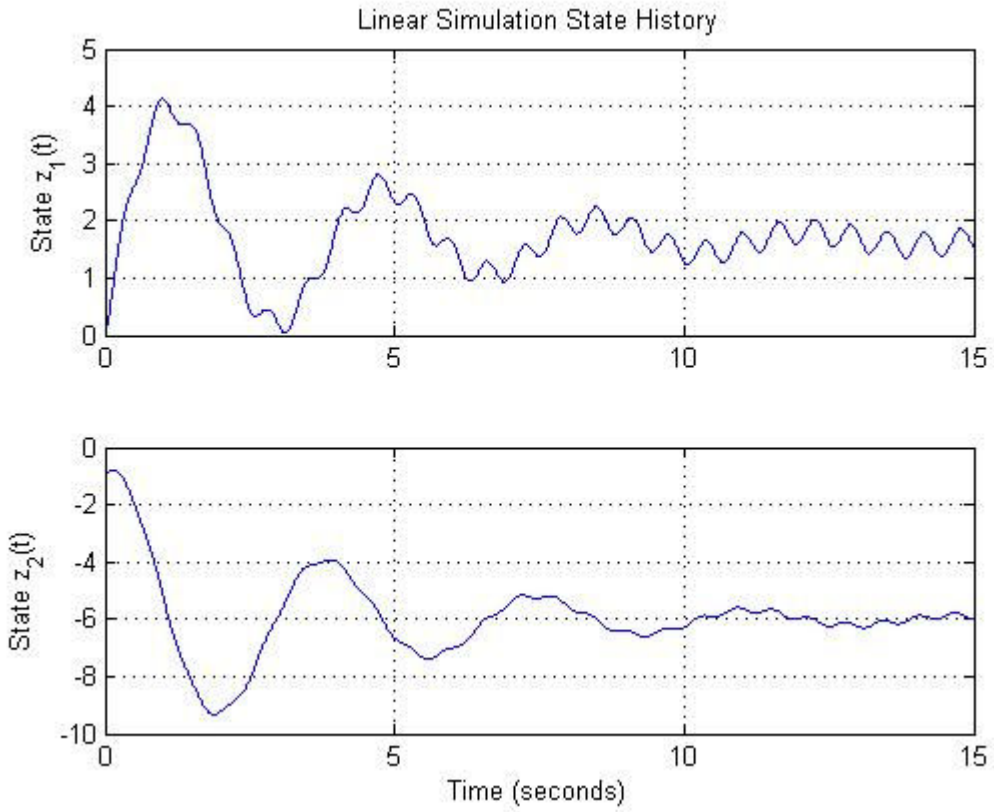
When called in this way, **lsim does NOT automatically generate a plot**. The input and output can be plotted as above using something like:

```
figure(1);
plot(t,u,'Color',[.6 .6 .6]); hold on;
plot(t,y);
xlim([0 t(end)]);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('Linear Simulation Results');
legend('Input','Output');
```

To plot the states, try something like:

```
figure(2);
subplot(2,1,1);
plot(t,z(:,1)); grid on;
xlim([0 t(end)]);
ylabel('State z_1(t)');
title('Linear Simulation State History');
```

```
subplot(2,1,2);
plot(t,z(:,2)); grid on;
xlim([0 t(end)]);
ylabel('State z_2(t)');
xlabel('Time (seconds)');
```



plot

Plot one or more functions or solutions and control what the plot looks like.

Plotting is one of the most useful things that MATLAB does. There are a huge array of options and special features that are possible with MATLAB plots, but the basic function is relatively simple.
Plot the two system responses:

$$x_1(t) = 5e^{-2t}\sin\left(10t + \frac{\pi}{3}\right)$$
$$x_2(t) = 4t - t^2$$

for 5 seconds, starting at $t = 0$ seconds.
First, specify the times for which you want to plot and the functions to plot:

```
t = 0:0.01:5; % Row vector of times from 0 to 8 in steps of 0.01.
% The step size must be small enough to make the plot look smooth.

% Two different functions (solutions) to plot:
x1 = 5*exp(-2*t).*sin(10*t + pi/3);
x2 = 4*t-t.^2;
```

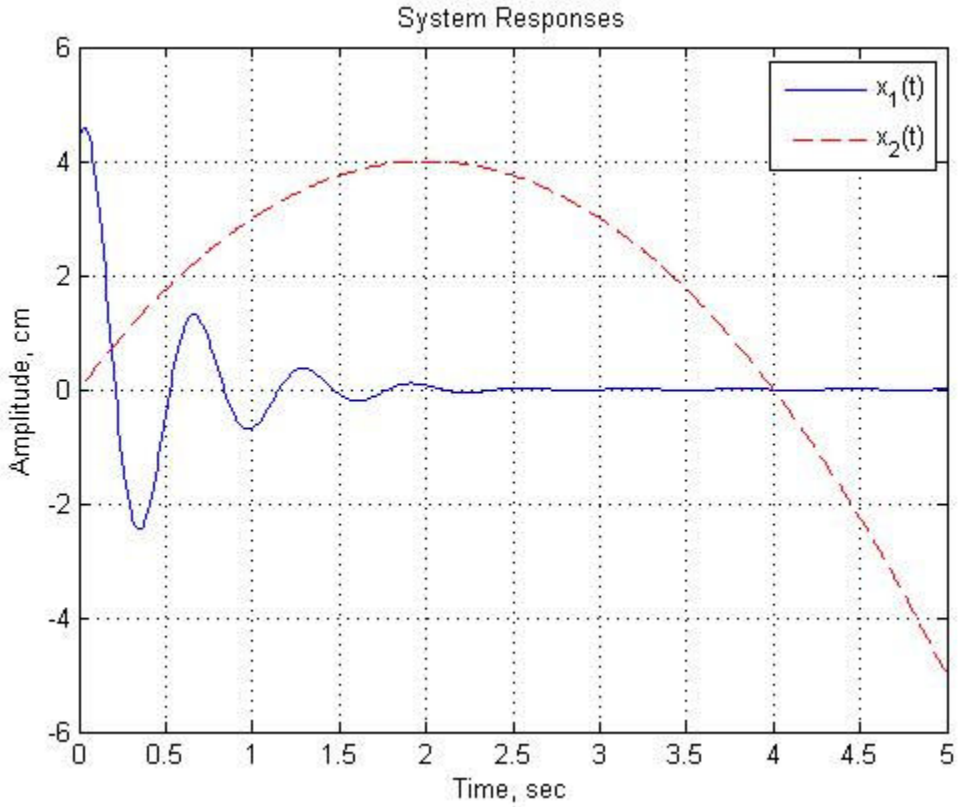
Note that `x1` and `x2` are defined in terms of `t`, so each of them is ALSO a row vector the same length as `t`. When a function like "`exp()`" or "`sin()`" is called with a vector input, the output is a vector, one output element for each input element.

Note also the use of "`.*`" and "`.^`" for multiplication and exponents. This is because you can't multiply two row vectors together or take the exponent of a row vector. Adding the "`.`" before the operator means that the calculations are element-wise. MATLAB multiplies the first elements of each vector together and stores the answer in the first element of the output vector, and then multiplies the second elements together and so on.

Next, create the basic plot and use some formatting commands as desired:

```
plot(t,x1,'b-',t,x2,'r--'); % Plot both functions in the same figure
% The optional inputs 'b-' and 'r--' cause the first function to be a
% solid blue line and the second to be a dashed red line.

% All of the commands below are not strictly necessary, but they
% control the way the plot looks (zoom level, information about units,
% etc.)
hold on; % If I call plot again, this command causes Matlab to add to
% the current plot instead of overwriting it.
grid on; % Add grid lines
xlabel('Time, sec'); % Add axis labels and a title
ylabel('Amplitude, cm');
title('System Responses');
xlim([0 5]); ylim([-6 6]);% Specify the x and y axis limits
% Add a legend (useful if there is more than one function plotted):
legend('x_1(t)','x_2(t)','Location','Northeast');
% The option 'Location' places the legend in the upper right corner
% (for the value 'Northeast')
```

The resulting plot looks like:



Many of the other examples in this guide also use the plot function in different ways to show the results of various commands.

poly

Calculate a polynomial with a given set of roots, or determine the characteristic polynomial associated with a square matrix.

Use the `poly` command to multiply terms of the form $(\lambda + \sigma)$ or $(\lambda + \sigma \pm i\omega)$ together and obtain the resulting polynomial. Suppose there are 5 roots: $1, -5, -8$, and $-3 \pm i4$.

```
% Specify all the roots of the polynomial in any order. Complex roots
% must come in complex conjugate pairs to guarantee real-valued
% polynomial coefficients.
r1 = 1;
r2 = -5;
r3 = -3 + 1i*4;
r4 = -3 - 1i*4;
r5 = -8;

% Store roots in a vector
rootvec = [r1; r2; r3; r4; r5];

% Calculate polynomial coefficients
coeffs = poly(rootvec);
```

Output:

```
coeffs =

           1       18      124      422      435    -1000
```

Note: used in this way, `poly` is the inverse of `roots`: `rootvec = roots(poly(rootvec))` and `coeffs = poly(roots(coeffs))`.

The function `poly` can also be used to calculate the characteristic equation using the "A" matrix from the state-space representation.

```
% Calculate the state space "A" matrix
A = [0 1 0 0 0;
     0 0 1 0 0;
     0 0 0 1 0;
     0 0 0 0 1;
     1000 -435 -422 -124 -18];

% Compute the coefficients of the corresponding characteristic
equation:
char_eq = poly(A);
```

The resulting polynomial is identical to the one above (because the first polynomial was used to construct the state space matrix).

polyval

Evaluate a polynomial at a specified value of the variable

Evaluate the polynomial

$$f(\lambda) = \lambda^6 - 3\lambda^4 + 22\lambda^3 + 50\lambda^2 + \lambda + 8$$

at $\lambda = -3$.

```
% Form a vector of the coefficients of the polynomial, including zeros
f_eq = [1, 0, -3, 22, 50, 1, 8];

% Evaluate:
polyval(f_eq,-3)

ans =

        347
```

This is equivalent to the calculation:

$$\begin{aligned} f(-3) &= (-3)^6 - 3(-3)^4 + 22(-3)^3 + 50(-3)^2 + (-3) + 8 \\ &= 729 - 3(81) + 22(-27) + 50(9) - 3 + 8 \\ &= 729 - 243 - 594 + 450 - 3 + 8 \\ &= 347 \end{aligned}$$

`rlocus`

Generate a root locus plot for a given dynamic system in transfer function or state space form.

Create a root locus plot for the characteristic equation given by

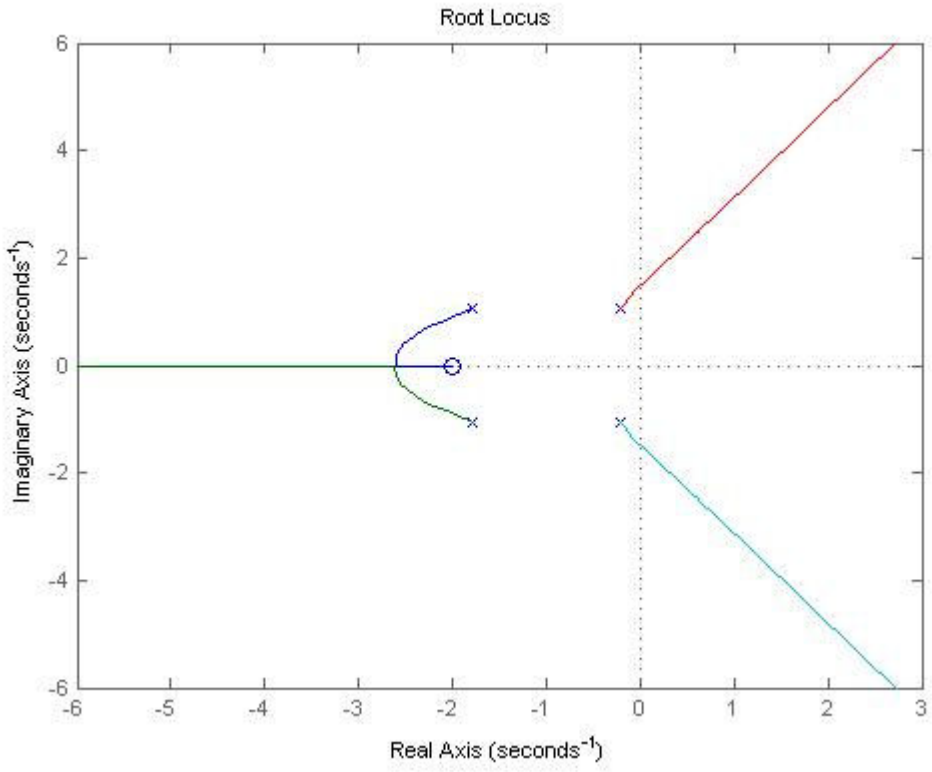
$$D(s) + K * N(s) = 0$$

where

$$D(s) = s^4 + 4s^3 + 7s^2 + 6s + 5$$

$$N(s) = s + 2$$

```
% Enter N and D coefficients:
N = [1 2];
D = [1 4 7 6 5];

% Plot the root locus
rlocus(num,den)
```



Note the poles, zeros, asymptotes, and break-in/break-out points. If you click on the plot in the MATLAB figure, it will display additional information about gain (K), damping, frequency, etc.

roots

Calculate the roots of a polynomial with given coefficients.

Calculate all the roots of the polynomial:

$$\lambda^8 + 22\lambda^6 + 354\lambda^5 - 501\lambda^3 + 88\lambda^2 + 74\lambda + 1330 = 0$$

```
% Enter the coefficients of a polynomial in a vector. Don't forget to
% include zeros if necessary:
coeffs = [1 0 22 354 0 -501 88 74 1330];

% Calculate the roots of this polynomial:
rootvec = roots(coeffs)
```

Output:

```
rootvec =

       3.0145 +     7.0809i
       3.0145 -     7.0809i
      -5.9318
      -1.6303
       1.196 +    0.63612i
       1.196 -    0.63612i
    -0.42953 +     1.0396i
    -0.42953 -     1.0396i
```

There are two negative real-valued roots, a complex-conjugate pair of roots with negative real parts, and two complex-conjugate pairs of roots with positive real parts.

Note: `roots` is the inverse of `poly`: `rootvec = roots(poly(rootvec))` and `coeffs = poly(roots(coeffs))`.

`ss`

Create a state space model representing a dynamic system

MATLAB can encapsulate information about a dynamic system in a special object called state space model. This model is often passed as an input to various functions instead of passing the state space matrices directly.

```
% Specify the state space matrices
A = [2 1;0 3]; B = [0; 1]; C = [-4 0]; D = 0;

% Create a MATLAB state space system model
sys = ss(A,B,C,D)
```

Output:

```
sys =

  a =
       x1  x2
   x1   2   1
   x2   0   3

  b =
       u1
   x1   0
   x2   1

  c =
       x1  x2
   y1  -4   0

  d =
       u1
   y1   0

Continuous-time state-space model.
```

The displayed information gives the 4 matrices, with columns and rows labeled according to which state (x), input (u) , or output (y) they correspond to. Multiple inputs and outputs are possible by changing the dimensions of the B, C, and D matrices.

ss2tf

Convert a dynamic system from state space form to transfer function form.

Find the 2x2 transfer function matrix equivalent to the following 3rd-order state space system with 2 inputs and 2 outputs:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & -6 & -1 \end{bmatrix},\ B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix},\ C = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 2 \end{bmatrix},\ D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
% Specify the state space system
A = [0 1 0; 0 0 1; -3 -6 -1]; B = [1 0; 0 1; 0 0];
C = [2 0 0; 0 1 2]; D = [0 0; 0 0];

% Compute the transfer function from the 1st input to both outputs
[num1,den1] = ss2tf(A,B,C,D,1)

% Compute the transfer function from the 2nd input to both outputs
% Note: den1 = den2
[num2,den2] = ss2tf(A,B,C,D,2)
```

Outputs:

```
num1 =
     0     2     2    12
     0     0    -6    -3

den1 =
     1     1     6     3

num2 =
     0     0     2     2
     0     1   -11    -6

den2 =
     1     1     6     3
```

The first row of `num1` is the numerator of the transfer function from input 1 to output 1, the second row of `num1` is the numerator of the transfer function from input 1 to output 2, and so on. The function `ss2tf` can handle many outputs but only one input at a time. The resulting transfer function matrix is:

$$TF = \begin{bmatrix} \dfrac{2s^2 + 2s + 12}{s^3 + s^2 + 6s + 3} & \dfrac{2s + 2}{s^3 + s^2 + 6s + 3} \\ \dfrac{-6s - 3}{s^3 + s^2 + 6s + 3} & \dfrac{s^2 - 11s - 6}{s^3 + s^2 + 6s + 3} \end{bmatrix}$$

step

Simulate the response of a dynamic system to a step function input.

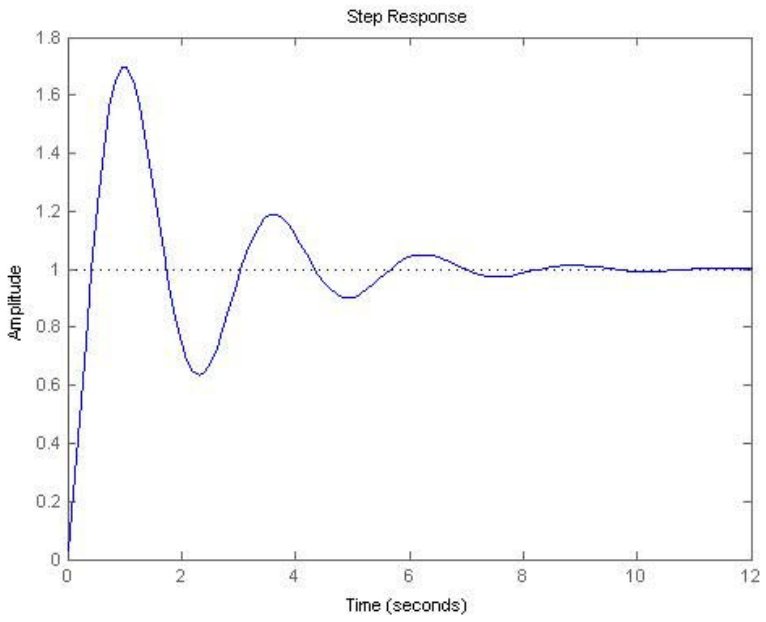
Simulate the response of the dynamic system in state-space form given by:

$$\frac{d}{dt}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -6 & -1 \end{bmatrix}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} + [0]u(t)$$

to a unit step input $u(t)$. Note that initial conditions of zero ($z_1(0) = z_2(0) = 0$) are typically assumed for step response problems, unless otherwise specified.

```
% State space matrices
A = [0, 1; -6, -1]; B = [2; 4]; C = [1, 0]; D = 0;

% Create state space system object:
sys_ss = ss(A,B,C,D);

% Convert to transfer function form, if desired:
[num,den] = ss2tf(A,B,C,D);
sys_tf = tf(num,den);

% Calculate and plot step response; any of these four expressions
% is equivalent:
step(A,B,C,D);
step(sys_ss);
step(num,den);
step(sys_tf);
```



Alternatively, if the system is in state space form (`A,B,C,D` or `sys_ss` above) one may call the `step` function in a way that returns the output history `y(t)`, a vector of corresponding times `t`, and the state history `z(t)`:

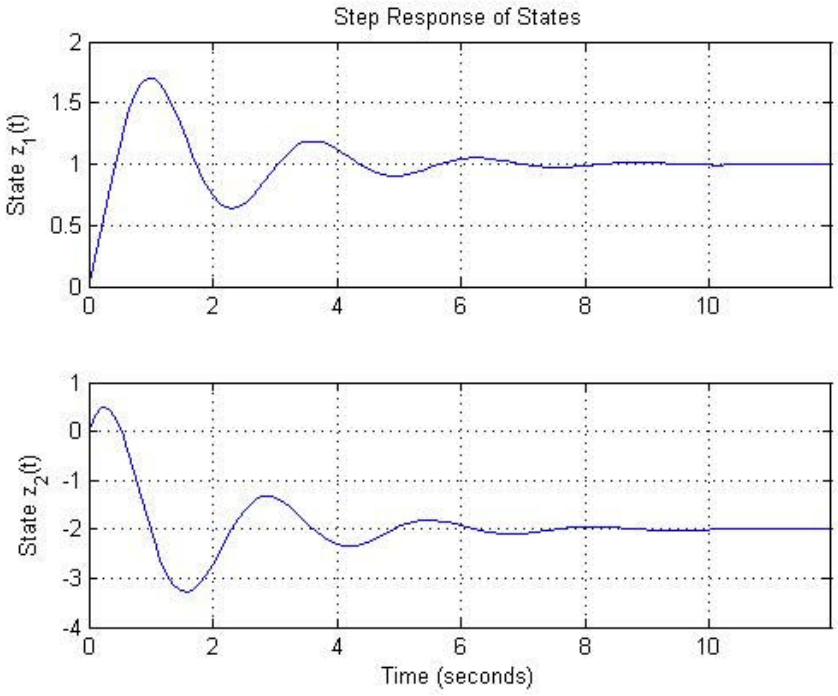
```
[y,t,z] = step(sys_ss);
```

In this case, **NO plot is automatically generated**, but one may reproduce the output history `y(t)` from the previous page using:

```
figure(1);
plot(t,y);
xlim([0 t(end)]);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('Step Response');
```

Alternatively, this call to the step function allows one to plot the state histories:

```
figure(2);
subplot(2,1,1);
plot(t,z(:,1)); grid on;
xlim([0 t(end)]);
ylabel('State z_1(t)');
title('Step Response of States');
subplot(2,1,2);
plot(t,z(:,2)); grid on;
xlim([0 t(end)]);
ylabel('State z_2(t)');
xlabel('Time (seconds)');
```



`tf`

Create a transfer function model representing a dynamic system.

MATLAB can encapsulate information about a dynamic system in a special object called transfer function model. This model is often passed as an input to various functions instead of passing the transfer function numerator and denominator vectors directly.

```
% Specify the transfer function numerator and denominator
num = [1 3 0]; den = [1 5 2 6];

% Create a MATLAB transfer function system model
sys = tf(num,den)
```

Output:

```
sys =

        s^2 + 3 s
  ---------------------
  s^3 + 5 s^2 + 2 s + 6

Continuous-time transfer function.
```

Multiple inputs and outputs (matrices of transfer functions) are possible, but they require `num` and `den` to be cell arrays.

tf2ss

Convert a dynamic system from transfer function form to state space form.

Suppose a system has the following transfer functions:

From the input $U(s)$ to the 1st output $Y_1(s)$:

$$\frac{3}{s^3 + s^2 + 5s + 10}$$

From the input $U(s)$ to the 2nd output $Y_2(s)$:

$$\frac{4(s^2 + 3s + 2)}{s^3 + s^2 + 5s + 10}$$

Find an equivalent state space representation (A,B,C, and D matrices).

```
% Enter the numerator coefficients
num1 = 3;
num2 = [4, 12, 8];
% Combine into numerator matrix (one output TF per row)
% using zero-padding if necessary.
num = [0 0 num1; num2];

% Enter the denominator coefficients (Note: TFs have same denominator)
den = [1, 1, 5, 10];

% Calculate one equivalent set of state space matrices (others exist)
[A,B,C,D] = tf2ss(num,den)
```

Output:

```
A =
    -1    -5   -10
     1     0     0
     0     1     0

B =
     1
     0
     0

C =
     0     0     3
     4    12     8

D =
     0
     0
```

The function can handle more than two outputs, but only one input per function call. (B will always be a column vector.)