SYSTEM DESIGN FOR UNCERTAINTY: Worked Examples

Franz S. Hover

Center for Ocean Engineering
Department of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts USA

Latest Revision: April 9, 2010

Contents

1 Linear Time Invariance

1. For each system below, determine if it is linear or non-linear, and determine if it is time-invariant or not time-invariant (adapted from Siebert 1986).

   (a) $y(t) = u(t+1)$
   The system is linear time-invariant; the output is just a time-advanced version of the input - it is noncausal!

   (b) $y(t) = 1/u(t)$
   Nonlinear, time-invariant. Replace $u(t)$ with $\alpha u(t)$ – this does not get us to $\alpha y(t)$, which would be required for linearity.

   (c) $3\ddot{y} + \dot{y}(t) - y(t) = u(t)$
   Linear time-invariant; an unstable second-order system. We have to assume $y(0) = 0$, and that we are talking about delays only if $u(t) = 0$ for $t <= 0$.

   (d) $y(t) = \sin(t)u(t)$
   Linear time-varying. The coefficient $\sin(t)$ is a function of time, so if a given input trajectory is played with different starting times, the outputs will be different - unless the initial times are off by a integer multiple of 2π.

   (e) $y(t) = u(t) + 2$
   Nonlinear time-invariant. Tiny input $u(t)$ still gives an output of about two, whereas large inputs will give an output of about the same; hence, the system as written does not capture scaling of $u(t)$.

   (f) $y(t) = \int_{-\infty}^{t} u(t_1)sin(t - t_1)dt_1$
   Linear time-invariant. Linearity is easy because the integral is a linear operator. Time-invariance is a little harder to show. Look at the right side first, with an advance in the input of τ, and where we set $t_2 = t_1 + \tau$:

   $$\int_{-\infty}^{t} u(t_1 + \tau)sin(t - t_1)dt_1 = \int_{-\infty}^{t+\tau} u(t_2)sin(t - (t_2 - \tau))dt_2$$

   Then advance the time in the expression for y:

   $$y(t + \tau) = \int_{-\infty}^{t+\tau} u(t_1)sin(t + \tau - t_1)dt_1$$

   These are the same and hence the system is time-invariant.

2. Determine whether the following is a time-invariant system or not; why? A large rocket in early flight is steered in pitch and yaw with control fins. As you know, the rocket assembly burns a *huge* amount of fuel during takeoff, and the rate of this burn is not affected by the steering. Because of the burn, however, the mass of the rocket is continually decreasing and its distribution is continually changing. Consider that the input is a perturbation to the fin angle, and the output is the pitch angle of the rocket. (As described, this system is linear.)
   The system is time-varying. The problem is an inverted pendulum, but the mass

at the top is changing; hence, a given deflection of the control surfaces will give a different dynamic response at different times during the flight. This attribute of the rocket requires a flight controller that also changes as an explicit function of time. Historically, the problem was one of the early successes in optimal control theory.

3. Determine whether the following is likely to be a linear system or not; why? A power electronics network contains resistors that heat up when the current through them is large. This heating causes them to increase their resistance. (As described, this system is time invariant.)
   The system is nonlinear, because running at different power levels will lead to different operating temperatures and different resistances. This assumes there is an adequate heat sink!

2 Convolution

The step function $s(t)$ is defined as zero when the argument is negative, and one when the argument is zero or positive:

$$s(t) = \begin{cases} 0 \text{ if } t < 0 \\ 1 \text{ if } t \geq 0 \end{cases}$$

For the LTI systems whose impulse responses are given below, use convolution to determine the system responses to step input, i.e., $u(t) = s(t)$.

1. $h(t) = 1$
   The impulse response is the step function itself - it turns on to one as soon as the impulse is applied, and this makes it a pure *integrator*. We get for the response to step input

$$\begin{aligned} y(t) &= \int_0^t s(\tau)s(t-\tau)d\tau \\ &= \int_0^t s(t-\tau)d\tau \text{ and the integrand is one because always } t \geq \tau \text{ so} \\ &= \int_0^t d\tau = t. \end{aligned}$$

   You recognize this as the integral of the input step.

2. $h(t) = \sin(t)$
   This impulse response is like that of an undamped second-order oscillator, having unity resonance frequency.

$$\begin{aligned} y(t) &= \int_0^t s(t-\tau)\sin(\tau)d\tau \\ &= \int_0^t \sin(\tau)d\tau \\ &= -\cos(\tau)|_0^t = 1 - \cos(t). \end{aligned}$$

3. $h(t) = 2\sin(t)e^{-t/4}$
   This is a typical underdamped response for a second-order system - a sinusoid multiplied by a decaying exponential. We make the substitution and find:

$$\begin{aligned} y(t) &= \int_0^t s(t-\tau)2\sin(\tau)e^{-\tau/4}d\tau \\ &= 2\int_0^t \sin(\tau)e^{-\tau/4}d\tau \\ &= \frac{32}{17}\left(1 - e^{-t/4}[\sin(t)/4 + \cos(t)]\right) \end{aligned}$$

3 Fourier Series

Compute the Fourier series coefficients A_0, A_n, and B_n for the following signals on the interval $t = [0, 2\pi]$:

1. $f(t) = 4\sin(t + \pi/3) + \cos(3t)$
   First, write this in a fully expanded form: $y(t) = 4\sin(t)\cos(\pi/3) + 4\cos(t)\sin(\pi/3) + \cos(3t)$. Then it is obvious that

$$\begin{aligned} A_0 &= 0 \text{ (the mean)} \\ A_1 &= 4\sin(\pi/3) \\ B_1 &= 4\cos(\pi/3) \\ A_3 &= 1, \end{aligned}$$

   and all other terms are zero, due to orthogonality.

2. $f(t) = \begin{cases} t, t < T/2 \\ t - T/2, t \geq T/2 \end{cases}$ (biased sawtooth)
   $A_0 = \pi/2$, the mean value of the function. Let's next do the A_n's:

$$\begin{aligned} A_n &= \frac{1}{\pi}\int_0^{2\pi} \cos(nt) f(t) dt \\ &= \frac{1}{\pi}\int_0^{\pi} \cos(nt) t dt + \frac{1}{\pi}\int_{\pi}^{2\pi} \cos(nt)(t - \pi) dt \\ &= \frac{1}{\pi}\int_0^{2\pi} \cos(nt) t dt - \int_{\pi}^{2\pi} \cos(nt) dt \\ &= \frac{1}{\pi}\left(\frac{\cos(nt)}{n^2} + \frac{t\sin(nt)}{n}\right)\Bigg|_0^{2\pi} - 0 \\ &= 0 \end{aligned}$$

   This makes sense intuitively because the cosines are symmetric functions around zero (even), whereas $f(t)$ is not. The signal's information is carried in the sine terms:

$$\begin{aligned} B_n &= \frac{1}{\pi}\int_0^{2\pi} \sin(nt) f(t) dt \\ &= \frac{1}{\pi}\int_0^{\pi} \sin(nt) t dt + \frac{1}{\pi}\int_{\pi}^{2\pi} \sin(nt)(t - \pi) dt \\ &= \frac{1}{\pi}\int_0^{2\pi} \sin(nt) t dt - \int_{\pi}^{2\pi} \sin(nt) dt \end{aligned}$$

   Now the second integral is $-2/n$ for n odd, and zero otherwise. Let's call it $q(n)$. Then continuing we see

$$\begin{aligned} B_n &= \frac{1}{\pi}\left(\frac{\sin(nt)}{n^2} - \frac{t\cos(nt)}{n}\right)\Bigg|_0^{2\pi} - q(n) \\ &= -2/n - q(n). \end{aligned}$$

   Hence $B_n = -2/n$ for even n, and zero otherwise. Try it out by making a plot!

4 Bretschneider Spectrum Definition

The formula for the Bretschneider (one-sided) ocean wave spectrum is

$$S(\omega) = \frac{5}{16}\frac{\omega_m^4}{\omega^5}H_{1/3}^2 e^{-5\omega_m^4/4\omega^4}$$

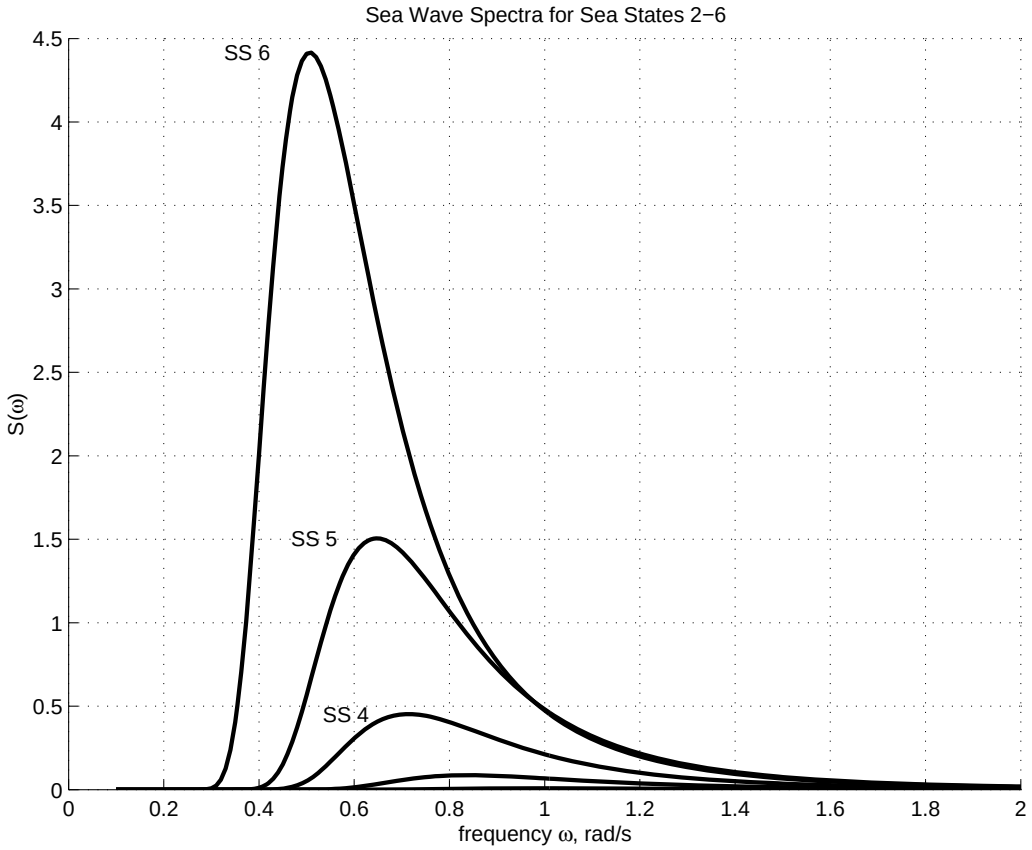
where ω is frequency in radians per second, ω_m is the modal (most likely) frequency of any given wave, and $H_{1/3}$ is the significant wave height. Make a single figure that shows the Bretschneider spectrum (S as a function of ω) for these cases:

SeaState	$2\pi/\omega_m$, sec	$H_{1/3}$, m
2	6.3	0.3
3	7.5	0.9
4	8.8	1.9
5	9.7	3.3
6	12.4	5.0

Here is the MATLAB code I used and the resulting figure:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2.017 Homework 1.  Bretschnieder Spectrum.
% FSH MIT Mechanical Engineering

clear all;
figure(1);clf;hold off; hold on; % note:  hold is on so we can overlay
                                 % figures

% show the data from the table in the problem
SSvec = [2 3 4 5 6] ; % sea states
wmvec = 2*pi*ones(size(SSvec)) ./ [6.3 7.5 8.8 9.7 12.4] ;
                                              % modal frequencies
Hsigvec = [0.3 0.9 1.9 3.3 5.0] ; % significant wave heights

% vector of frequencies for the spectrum calculation
wvec = [.1:.01:2];

% step through the different seastates
for i = 1:length(wmvec),

   wm = wmvec(i) ;
   Hsig = Hsigvec(i) ;
   SS = SSvec(i) ;
```

```
   for j = 1:length(wvec),
      w = wvec(j) ;
      S(j) = 5/16 * wm^4 / w^5 * Hsig^2 * exp(-5 * wm^4 / 4 / w^4) ;
   end;

   % check that we got the right formula!
   disp(sprintf(...
      'Square Root of Integral of Area of S:  %g;  4*Hsig:  %g', ...
      sqrt(sum(S)*mean(diff(wvec))), 1/4*Hsig));

   % add on the curves and some labels
   plot(wvec,S,'LineWidth',2);
   if SS > 3,
      text(wm-.18,max(S),sprintf('SS %g', SS));
   end;
end;

% finish off the plot
grid;
title('Sea Wave Spectra for Sea States 2-6');
xlabel('frequency \omega, rad/s');
ylabel('S(\omega)');

print -deps hw1_bret.eps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```


Sea Wave Spectra for Sea States 2−6
SS 6
SS 5
SS 4
S(ω)
frequency ω, rad/s
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0.2
0.4
0.6
0.8
1.2
1.4
1.6
1.8

5 LTI Machine?

If you put a sequence of five numbers into a certain machine, it responds with a five-number sequence; this exchange constitutes one experiment. For each possible machine characteristic below, give enough sets of input-output sequences (that is, "experiments") that would demonstrate it:

1. Linear, time-invariant
   A solution: [1,1,1,1,1] → [3,2,2,1,0] and [0,2,2,2,2] → [0,6,4,4,2].

2. Linear, time-varying
   A solution: [1,1,0,0,0] → [0,3,2,1,0] and [0,2,2,0,0] → [0,6,4,2,0].

3. Nonlinear, time-invariant
   A solution: [1,2,3,0,0] → [5,4,3,0,0] and [0,2,4,6,0] → [0,4,3,2,0]. Note this is a static mapping; a dynamic system would take more trials to determine that it was both nonlinear and time-invariant.

4. Nonlinear, time-varying
   A solution: [2,3,4,0,0] → [1,1,0,1,1] and [0,4,6,8,0] → [0,0,3,0,3]. A time-varying system cannot be a static map, so we have to have dynamics. However, similarly to the above case, there is a limit to what can be deduced with only two trials.

6 Convolution of Sine and Unit Step

The sine function $q(t)$ has a zero value before zero time, and then is a unit sine wave afterwards:

$$q(t) = \begin{cases} 0 \text{ if } t < 0 \\ \sin(t) \text{ if } t \geq 0 \end{cases}$$

For the LTI systems whose impulse responses $h(t)$ are given below, use convolution to determine the system responses to a sine function input, i.e., $u(t) = q(t)$.

1. $h(t) = 1$
   Solution:
   $$y(t) = \int_0^t h(t-\tau)q(\tau)d\tau = \int_0^t \sin(\tau)d\tau = -\cos(\tau)|_0^t = 1 - \cos t.$$

2. $h(t) = \sin(\alpha t)$, where α is a fixed positive number.
   Solution:
   $$\begin{aligned}
   y(t) &= \int_0^t h(\tau)q(t-\tau)d\tau \\
   &= \int_0^t \sin(\alpha\tau)\sin(t-\tau)d\tau \\
   &= \int_0^t \left[\sin(\alpha\tau)\sin t\cos\tau - \sin(\alpha\tau)\cos t\sin\tau\right]d\tau \qquad \text{(from a trig. identity)} \\
   &= \sin t\int_0^t \sin(\alpha\tau)\cos\tau d\tau - \cos t\int_0^t \sin(\alpha\tau)\sin\tau d\tau \\
   &= \frac{1}{2}\sin t\int_0^t \left[\sin((\alpha+1)\tau) + \sin((\alpha-1)\tau)\right]d\tau - \\
   &\quad \frac{1}{2}\cos t\int_0^t \left[\cos((\alpha-1)\tau) - \cos((\alpha+1)\tau)\right]d\tau \qquad \text{(two more trig. identities)} \\
   &= \frac{1}{2}\sin t\left[-\frac{1}{\alpha+1}\cos((\alpha+1)t) - \frac{1}{\alpha-1}\cos((\alpha-1)t) + \frac{1}{\alpha+1} + \frac{1}{\alpha-1}\right] - \\
   &\quad \frac{1}{2}\cos t\left[\frac{1}{\alpha-1}\sin((\alpha-1)t) - \frac{1}{\alpha+1}\sin((\alpha+1)t)\right] \\
   &= \frac{1}{2}\sin t\left[\frac{1}{\alpha+1} + \frac{1}{\alpha-1}\right] + \\
   &\quad \frac{1}{2}\sin t\left[-\frac{1}{\alpha+1}\cos((\alpha+1)t) - \frac{1}{\alpha-1}\cos((\alpha-1)t)\right] + \\
   &\quad \frac{1}{2}\cos t\left[-\frac{1}{\alpha-1}\sin((\alpha-1)t) + \frac{1}{\alpha+1}\sin((\alpha+1)t)\right] \\
   &= \frac{1}{2}\sin t\left[\frac{1}{\alpha+1} + \frac{1}{\alpha-1}\right] - \\
   &\quad \frac{1}{2}\frac{1}{\alpha-1}\sin(\alpha t) + \frac{1}{2}\frac{1}{\alpha+1}\sin(\alpha t) \qquad \text{(and two more identities)} \\
   &= \frac{1}{\alpha^2-1}\left[\alpha\sin t - \sin(\alpha t)\right].
   \end{aligned}$$

This result can be checked numerically, or with the LaPlace transform. It is important to note that this solution applies only when $\alpha \neq 1$ *- the factors of* $1/(\alpha - 1)$ *after the integrals are computed make no sense. Instead, for this case the* $\sin((\alpha - 1)t)$ *in the integral will be replaced with zero, and* $\cos((\alpha - 1)t)$ *will be replaced with one. Working things out along the same lines gives:*

$$y(t) = \frac{1}{2}(\sin t - t\cos t).$$

Clearly, this is an unbounded response, the usual result of forcing a system exactly at its resonant frequency.

7 Fourier Series Calculations

Compute the Fourier series coefficients A_0, A_n, and B_n for the following signals on the interval $T = [0, 2\pi]$:

1. $f(t) = 2\sin(t + \pi/4) + \cos(5t + \pi/3)$
   Solution: use trigonometric identities to rewrite this as $f(t) = 2\sin(t)\cos(\pi/4) + 2\cos(t)\sin(\pi/4) + \cos(5t)\cos(\pi/3) + \sin(5t)\sin(\pi/3)$. *Thus,* $A_0 = 0$, $A_1 = 2\sin(\pi/4) = \sqrt{2}$, $B_1 = 2\cos(\pi/4) = \sqrt{2}$, $A_5 = \cos(\pi/3) = 1/2$, $B_5 = \sin(\pi/3) = \sqrt{3}/2$, *and all the other coefficients are zero.*

2. $$f(t) = \begin{cases} 1, t < T/2 \\ 0, t \geq T/2 \end{cases} \text{ (biased square wave)}$$

   Solution: A_0 *is the mean value of the signal, or* $A_0 = 1/2$. *Applying the formulas for the coefficients, we get*

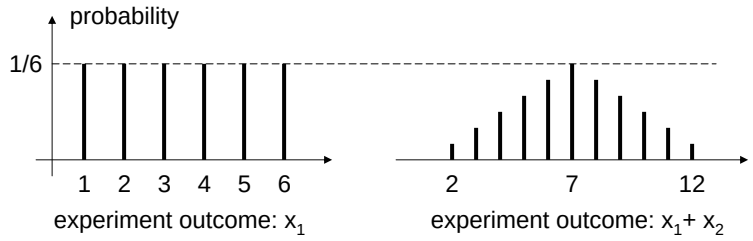
   $$\begin{aligned} A_n &= \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(nt)dt = \frac{1}{\pi}\int_0^{\pi} \cos(nt)dt = \frac{1}{n\pi}\sin(nt)|_0^{\pi} = 0 \\ B_n &= \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(nt)dt = \frac{1}{\pi}\int_0^{\pi} \sin(nt)dt = -\frac{1}{n\pi}\cos(nt)|_0^{\pi} = z(n), \end{aligned}$$

   where z *is zero if* n *is even, and* z *is* $2/n\pi$ *if* n *is odd. Try this out in MATLAB!*

8 Probability Primer with Dice

You are given two fair dice.

1. Make a plot of the possible outcomes of one toss of one die, versus the likelihood (probability) of that outcome. Clearly there are six possible outcomes, with equal likelihoods of occurring. We call this graph a probability mass function, or pmf.



2. Make a similar plot for a throw of both dice, where the outcome is the *sum* of the two values, i.e., the outcome of a toss giving [3,4] is seven. Your pmf should account for the fact that there are more ways to roll a three ([1+2],[2+1]) than to roll a two ([1,1]).

   Solution: As shown in the figure, the pmf for the two-throw case has a nice linear structure. The height of each spike follows the possible number of ways the additions can occur: 2 is achieved only by rolling [1,1], 3: [1,2] or [2,1], 4: [1,3] or [2,2] or [3,1], 5: [1,4] or [2,3] or [3,2] or [4,1], and so on.

3. For the case of throwing both dice, what is the most likely outcome, and what is its probability on any given toss?
   Solution: the most likely outcome is seven, and it occurs one-sixth of the time.

9 Autonomous Vehicle Mission Design, with a Simple Battery Model

An autonomous land robot carries its own energy in the form of a $E(t = 0) = 700Wh$ (Watt-hour) battery. The robot can stay stationary at a cost of $18W$ to run its processor and instrumentation, or it can move with the *additional* locomotion cost of $35U^3$ Watts, where U is the speed in m/s. The battery has a nonlinear discharge curve, that penalizes high loads: $-dE/dt = P + 0.005P^2$, where P is the total power load (taken in Watts), and dE/dt is also in Watts.

1. What is the longest *duration* mission that we can run, if the mission ends when $E = 0$?

   The longest duration mission is one where we don't move. We simply have the hotel load eating away at the battery, drawing $19.6W$*; the mission time is* $T = 35.7h$*. Formulas are in the attached MATLAB code.*

2. What is the longest *distance* mission that we can run? Give the distance, duration, and vehicle speed for this mission.

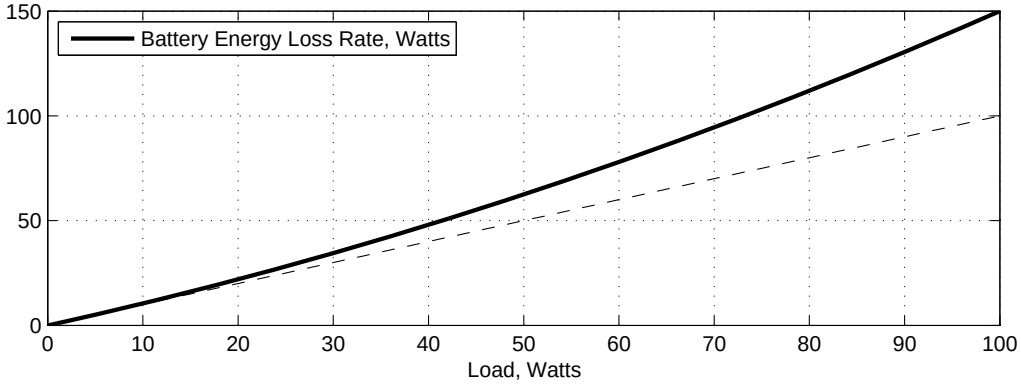
   The battery energy loss rate is a static function of load, so we have to just pick the best speed and run until the battery is dead. My code takes a numerical approach, sweeping through a number of possible speeds and picking out the case with greatest distance. You could also do it directly, by writing one equation for the distance as a function of the speed, and then finding the maximum using l'Hopital's rule (zero slope). Either way, we find that the greatest distance is about 52.5km*, over about* $24.3h$ *at a speed of* $0.60m/s$*.*

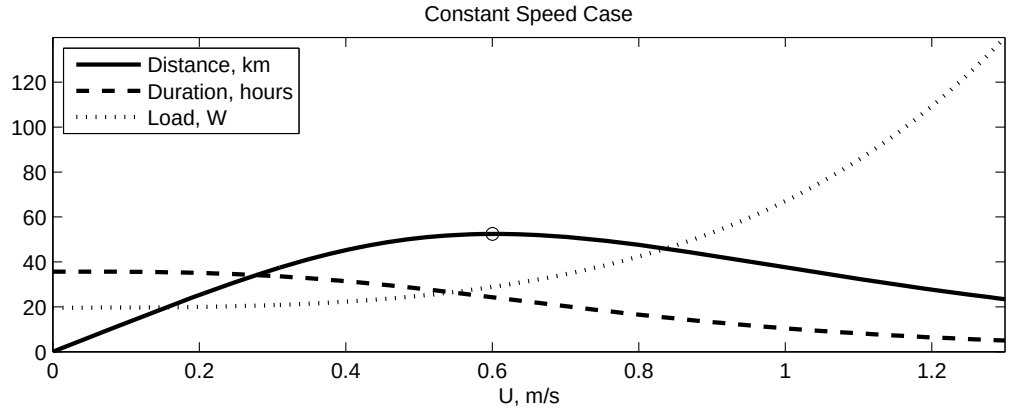
3. What is the longest *distance* mission, if we alternate resting periods and moving at a specific speed, with about half of the time spent in each state? Give the distance, duration, and the speed when moving.

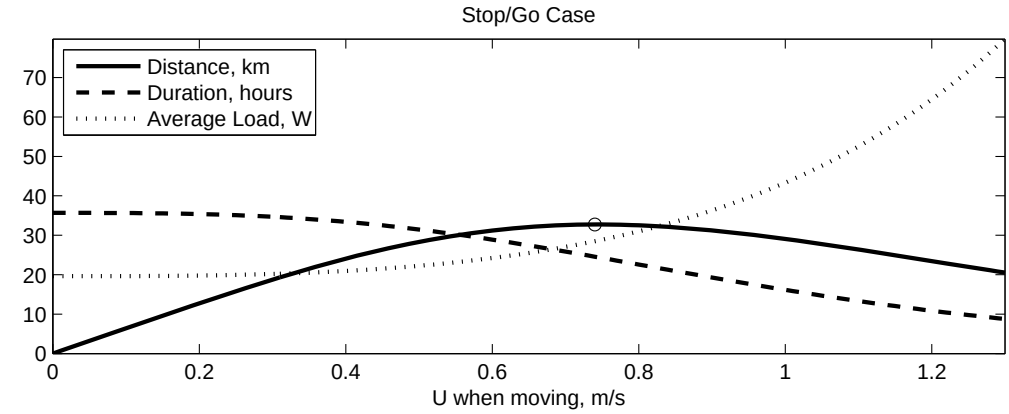
   The key here is that half of the time, we have only the hotel load to satisfy, and otherwise we have hotel plus locomotion. In all, the battery energy discharges at the average of the two rates. But the distance traveled is only the moving speed times half of the mission duration. I get a distance of $32.7km$*, over about* $24.6h$ *at a speed of* $0.74m/s$*.*

4. Comment on how you would use your findings in the design of effective mission plans.

   The longer the mission, the more important it is to understand the hotel load as distinct from locomotion. This is especially true for a vehicle that has a lot of power-hungry sensors or communication gear on board, but may not have much locomotion cost. In this example, the hotel load costs us $20km$ *of range when we operate in the stop-go mode.*


Battery Energy Loss Rate, Watts
150
100
50
0
0
10
20
30
40
50
60
70
80
90
100
Load, Watts


Constant Speed Case
Distance, km
Duration, hours
Load, W
120
100
80
60
40
20
0
0
0.2
0.4
0.6
0.8
1
1.2
U, m/s


Stop/Go Case
Distance, km
Duration, hours
Average Load, W
70
60
50
40
30
20
10
0
0
0.2
0.4
0.6
0.8
1
1.2
U when moving, m/s

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Autonomous Vehicle Energy with Battery Model
% MIT 2.017 FSH Sept 2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;

E0 = 700*3600 ;         % battery capacity, Joules
hotelLoad = 18 ;        % Watts
Uvec = 0:.01:1.3 ;      % a vector of speeds we'll look at
squareCoefficient = 0.005 ;
propLoadCoefficient = 35 ;

% Show the discharge curve of the battery %%%%%%%%%%%%%%%%%%%%%
P = 0:1:100 ;
Edot = P + squareCoefficient*P.^2 ;
figure(1);clf;hold off;subplot(211);
plot(P,Edot,'LineWidth',2);
hold on;
plot(P,P,'--k');
xlabel('Load, Watts');
legend('Battery Energy Loss Rate, Watts',2);
grid;

% Problem 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Edot = hotelLoad + squareCoefficient*hotelLoad^2 ;
duration1 = E0 / Edot ;
disp(sprintf('Max duration mission is %g hours, 0 km.', duration1/3600));

% Problems 2 and 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
i = 1 ;
for U = Uvec,
    propulsionLoad = propLoadCoefficient*U^3 ;
    totalLoad = hotelLoad + propulsionLoad ;

    % Problem 2
    Edot2(i) = totalLoad + squareCoefficient*totalLoad^2 ;
    duration2(i) = E0 / Edot2(i) ;
    distance2(i) = duration2(i) * U ;

    % Problem 3
    Edot3(i) = (hotelLoad + squareCoefficient*hotelLoad^2) / 2 + ...
               (totalLoad + squareCoefficient*totalLoad^2) / 2 ;
               % time-averaged value of Edot
    duration3(i) = E0 / Edot3(i) ;
```

```
    distance3(i) = duration3(i) * U / 2 ; % note division by two
    i = i + 1 ;
end;
figure(2);clf;hold off;
subplot(211);
plot(Uvec,distance2/1000,Uvec,duration2/3600,'--',Uvec,Edot2,':',...
    'LineWidth',2) ;
axis('tight');
xlabel('U, m/s');
title('Constant Speed Case');
legend('Distance, km', 'Duration, hours', 'Load, W',2);

[junk,ind] = sort(distance2) ;
ix = ind(end);
hold on;
plot(Uvec(ix),distance2(ix)/1000,'ro');
disp(sprintf('Max constant-speed distance mission is %g hours, %g km', ...
    duration2(ix)/3600, distance2(ix)/1000));

subplot(212);
plot(Uvec,distance3/1000,Uvec,duration3/3600,'--',Uvec,Edot3,':',...
    'LineWidth',2) ;
axis('tight');
xlabel('U when moving, m/s');
title('Stop/Go Case');
legend('Distance, km', 'Duration, hours','Average Load, W',2);

[junk,ind] = sort(distance3) ;
ix = ind(end);
hold on;
plot(Uvec(ix),distance3(ix)/1000,'ro');
disp(sprintf('Max stop/go distance mission is %g hours, %g km', ...
    duration3(ix)/3600, distance3(ix)/1000));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

10 Simulation of a System Driven by a Random Disturbance

1. Simulate the second-order system:

$$x'' + ax' + bx \quad = \quad d(t)$$

with $a = 0.4$ and $b = 2.25$.

Figure 1 below shows responses to the same disturbance $d(t)$ for all the three values of a. Note that because the phases are generated from random numbers, your figure might not look like this, although the size of the response should be similar. Also, the component periods repeat within sixty seconds, so $d(t)$ repeats itself in this graph.

2. From the graph, about what is the "significant height" of the motion?

The significant height of the response with $a = 0.4$ is around two units.

3. What is the effect of reducing or increasing the damping in this system, say $a = 0.2$ and then $a = 1.0$?

The response of the system is clearly strongly dependent on the damping ratio: the oscillations are much larger as the damping ratio gets smaller. This is no surprise since the undamped natural frequency is 1.5, near where the disturbance frequencies lie. This example shows a rather bad resonance condition between the physical system and the disturbance, which can be mitigated by increasing the damping.

With regard to design, however, it is worth pointing out that if the damping increases, then the system responds more slowly to initial conditions and to impulsive disturbances. In other words, if a boat or airplane is knocked over by a big wave or wind gust, damping will slow down the return to upright. Most people don't like that!

```
%------------------------------------------------------------------------
% Time-domain simulation of a second-order system with two-mode
% random disturbance.

clear all;
global a b omega A phi ;

% set the frequencies, amplitudes, and phase angles
omega = .5:.25:2.5 ;
A = [.2 .4 .6 .2 .1 .4 .3 .2 .1] ;
phi = 2*pi*rand(size(A)) ;

Tfinal = 60 ; % final time
```

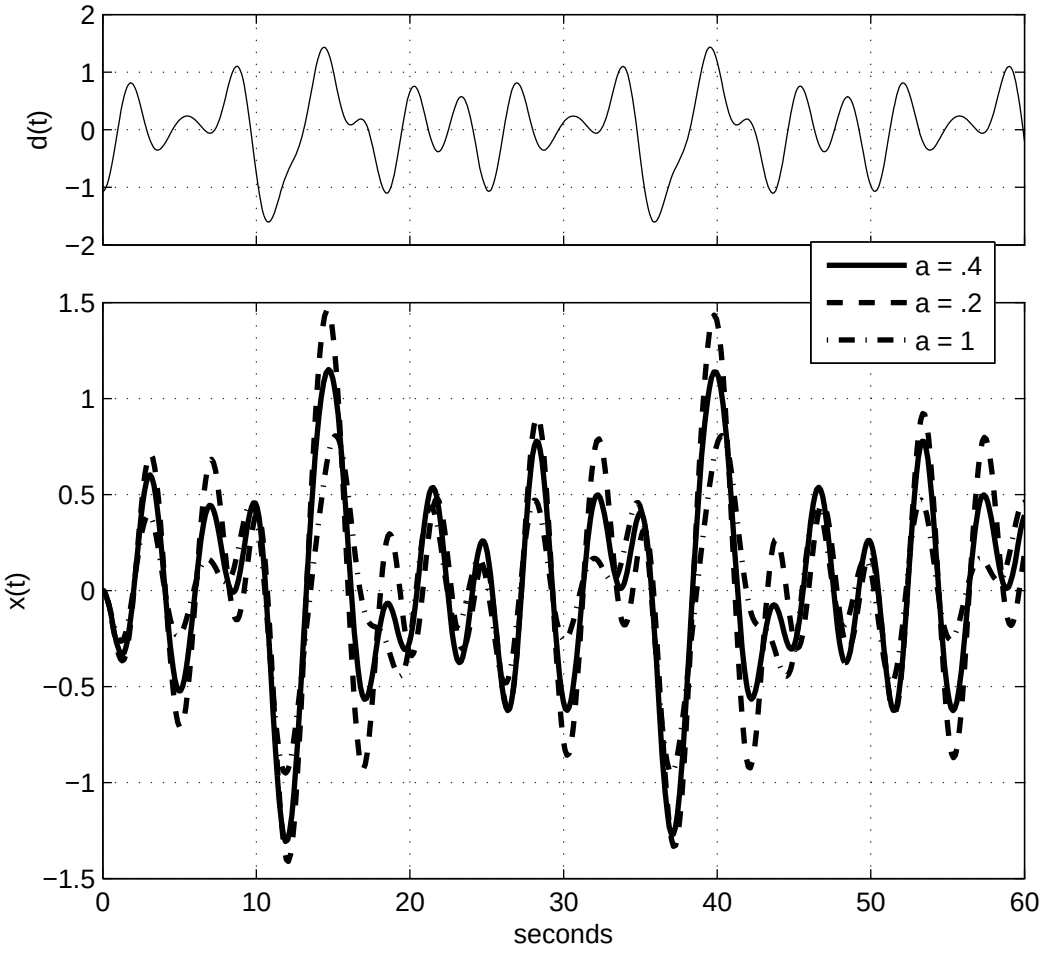


Figure 1: Time-domain simulation of vehicle subject to random disturbance.

```
b = 2.25 ;  % wn^2; spring term

% run the simulation for three values of a

a = .4 ;
[t1,y1] = ode45('td_veh_dxdt',[0 Tfinal],[0 0]) ;

a = .2 ;
[t2,y2] = ode45('td_veh_dxdt',[0 Tfinal],[0 0]) ;

a = 1 ;
[t3,y3] = ode45('td_veh_dxdt',[0 Tfinal],[0 0]) ;

% recreate d(t) here in the main program
```

```
tvec = 0:.1:Tfinal;
d = zeros(1,length(tvec));
for i = 1:length(omega),
    d = d + A(i) * sin(omega(i)*tvec + phi(i)) ;
end;

% plot the results

figure(1);clf;hold off;
subplot('Position',[.2 .75 .6 .2]);
plot(tvec,d);
ylabel('d(t)');
grid;
set(gca,'XTickLabel',[]);

subplot('Position',[.2 .2 .6 .5]);
plot(t1,y1(:,2),'-',t2,y2(:,2),'--',t3,y3(:,2),'-.','LineWidth',2);
legend('a = .4', 'a = .2', 'a = 1',3);
grid;
xlabel('seconds');
ylabel('x(t)');

disp('Move label if necessary, then hit a key to save.');
pause ;

print -deps td_veh.eps
%-----------------------------------------------------------------


%-----------------------------------------------------------------
% Time Domain simulation of second-order system - derivative
% function

function [xdot] = td_veh_dxdt(t,x) ;

global a b A omega phi ;

% construct the disturbance - it is purely a function of time
d = 0 ;
for i = 1:length(omega),
    d = d + A(i) * sin(omega(i)*t + phi(i)) ;
end;

% calculate the derivatives (column!)
```

```
xdot(1,1) = -a*x(1) - b*x(2) + d ;
xdot(2,1) = x(1) ;
%-------------------------------------------------------------------
```

11 Sea Spectrum and Marine Vehicle Pitch Response

1. Make a plot of the spectrum for about one hundred frequencies from zero to 4 rad/s, with modal frequency $\omega_m = 1$ rad/s, and significant wave height $H_{1/3} = 0.90$m.

   See the top graph in Figure 2 for the wave spectrum.

2. Confirm that the area under the spectrum is equal to E_S, by making a numerical integration. You can then take this E_S to double-check $H_{1/3}$.

   The area under the curve E is 0.0502; it was supposed to be 0.0506. Note in the MATLAB code that I randomize the actual frequencies used; this is to avoid related frequencies (as seen in the first problem above). The corresponding significant height is 0.896m; pretty close to the desired value of 0.90m.

3. Make ten minutes worth of this wave-like data, using a sampling period of 0.1 seconds, and show a plot, with the original $H_{1/3}$ maximum and minimum levels indicated.

   See the middle graph in Figure 2 for a time-domain realization of this spectrum.

4. Compute the parameter E_Y from the area under $Y(\omega)$, and so estimate the "significant height" of the pitch motion.

   The significant height of the pitch motion is 0.26 radians, or about fifteen degrees. The lower graph in the figure shows the multiplication in frequency space.

```
%------------------------------------------------------------------------
%  Bretschneider Sea Spectra and Vehicle Pitch Response

clear all;

dfreq = .04;        % frequency resolution for creating waves, rad/s
maxfreq = 3.99 ;    % highest wave frequency made, rad/s
dt = .1 ;           % sampling period, s
tff = 600 ;         % final time, s
wmodal = 1 ;        % modal frequency (used for Bretschnieder construction)
E = .05 ;           % spectra parameter for Bretschnieder:
                    % E = sig_height^2 / 16

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% no user parameters below this point
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

time = 0:dt:tff;        % time vector
n = length(time) ;      % number of samples
freq = dfreq:dfreq:maxfreq;           % imposed frequencies
                                      % (near zero to maxfreq)
```

```
freq = freq + (.5-rand(1,length(freq)))*dfreq ;
        % add random components in freq vector

disp(sprintf('Imposed dfreq/available resolution: %g (best if below one)',...
    dfreq/(2*pi/dt/length(time)))) ;

if (pi/dt < max(freq)),
    disp('Must have higher sampling rate to avoid aliasing! -- ABORT.');
    break ;
end;

% make up a Bretschneider spectrum and then get out the amplitudes
% for the example case
B = wmodal^4*1.25 ;
H = 4*sqrt(E) ;
A = 4*B*E ;
sBret = A./freq.^5.*exp(-B./freq.^4);

figure(1);clf;hold off;
subplot('Position',[.2 .2 .5 .2]);
plot(freq,sBret,'LineWidth',2);
xlabel('rad/s');
title('Bretschneider Spectrum, H_{1/3} = 0.90m');
print -deps bret_veh1.eps

disp(sprintf('E:  %g vs. %g.', E, sum(sBret)*dfreq));

amp = sqrt(2*sBret*dfreq) ;
% Note that sBret is invariant with dfreq, but amp definitely changes
% with dfreq, according to PNA def. of spectrum given in the problem.

% make up the time series, with random phase
phase = 2*pi*randn(length(freq),1);
x = zeros(size(time));
for i = 1:length(freq)
    x = x+amp(i)*cos(freq(i)*time+phase(i));
end

figure(2);clf;hold off;
subplot(211);
plot(time,x); xlabel('Time, seconds')
hold on;
plot([0 tff],[1 1]*H/2,'r--',[0 tff],-[1 1]*H/2,'r--');
legend('Simulation','+/- H_{1/3}/2');
```

```
print -deps bret_veh2.eps

% Here is the transfer function, and the pointwise frequency multiply of
% F*F' with S
i = sqrt(-1);
F = (.4*i*freq + .3) ./ (-freq.^2 + i*freq*1 + 3) ;
FF = F.*conj(F) ;

figure(3);clf;hold off;
subplot('Position',[.2 .2 .5 .5]);
plot(freq,FF,freq,sBret,'--',freq,FF.*sBret*20,':','LineWidth',2) ;
legend('FF*','S','20 x S x FF*',1);
xlabel('rad/s');
print -deps bret_veh3.eps

% Compute the area under the curve and from it get the significant
% response height
FFS = FF.*sBret ;
EFFS = sum(FFS)*dfreq ;
FFSsig = 4*sqrt(EFFS) ;
disp(sprintf('Significant response height:  %g.', FFSsig));

%-------------------------------------------------------------------------
```

12 Ranging Measurements in Three-Space

The global positioning system (GPS) and some acoustic instruments provide long-baseline navigation - wherein a number of very long range measurements can be used to triangulate. We call the item that we want to track the *Target*, and the nodes in the navigation system the *Satellites*. The locations of the satellites are assumed to be well known, and what we measure during tracking are ranges from the satellites to the target. In a plane, you can appreciate that two satellites would provide two range measurements, and the target could then be located on one of two points that form the intersection of two circles.

1. For a planar setting, how many satellites are required to uniquely locate a target at an arbitrary location, and how these should be laid out? Include sketches as needed to explain your reasoning.

   Three satellites give three ranges, corresponding with three circles. The intersection of circles is - in the best case - a single point, the unique localization result. There are several notable conditions where you'll get bad results with three satellites. If they are colinear, you get no information about location in the direction perpendicular to the line. When noise is included (as below), we don't want to be even close to colinear: we want low aspect-ratio ("not too long and thin") triangles. More fundamentally, if the target is located on or near the line connecting any two satellites, then it is as if we have lost one satellite - the second range measurement does us very little good.

2. Consider a problem now in three-space. Two satellites are given, with locations [X,Y,Z] of $[500, 500, 1000]m$ and $[-500, 500, 1000]m$. The target ranges are measured at $724m$ and $768.2m$ respectively, and suppose we know also that the target is at an altitude z of less than $1000m$. Can you make an estimate for the target location $[x, y, z]$ in three-space? If so, give it.

   No, you cannot make an estimate, because you only have two constraints. Think of each range measurement as a sphere in three-space; the intersection of two spheres is a circle. Without more information, you can't say what is the $[x, z]$ location, although the y location is easy to get.

3. Augment these two satellites with third, having position $[500, -500, 1000]m$; the corresponding range measurement is $649.8m$. Can you make an estimate for the target location in three space? If so, give it.

   Yes, we have enough information now to do a full localization. See the attached code for a numerical approach. Note that at least one student figured out the answer through algebraic manipulation, despite the fact that the solution satisfies three nonlinear equations! The location of the target is $[33, -51, 950]m$. There is another solution that will satisfy the three range equations, $[33, -51, 1050]m$; it is the mirror through the plane of the satellites.

4. Almost all sensors have noise, and such range-based navigation systems are no exception. What is the sensitivity of your best computed target location above to a

one-meter error in each of the three range measurements? (Perturb the range measurements separately.) Give an explanation for what you see happening, using sketches.

Adding one meter onto each of the three ranges in turn gives solutions of $[32.3, -51.7, 950.2]m$, $[33.8, -51.0, 943.3]m$, *and* $[33.0, -50.3, 943.3]m$, *respectively. The worst error here is about* $6.7m$, *pretty bad for a one-meter range error. It occurs because the target is fairly close to the plane of the satellites; as alluded to in the planar case, these measurements provide only poor-quality information about the direction normal to the plane, and it does not stand up to noise.*

5. Find a location for a fourth satellite, that will bring down the sensitivity of the localization to noisy range measurements. Demonstrate that the whole system is now better behaved with noise, and explain why, using sketches if necessary.

   Since the problem is that the target is close to the plane of satellites, a logical solution is to put a satellite far from the plane. Indeed, if we put one at $[0, 0, 0]m$, *we create a very nice tetrahedron-like shape, and the errors from perturbations in all four channels go to* $[32.3, -51.7, 950.0]m$, $[33.4, -51.4, 949.9]m$, $[32.7, -50.7, 950.0]m$, $[33.0, -51.1, 951.0]m$. *The positioning error now is on a par with the range error, and this system is considerably more robust against noise.*

   In the algorithm, you notice that we are here solving four equations in three unknowns. As written, this is an over-determined problem and we are performing a least-squares fit: the sum of squares of the range equation errors is minimized. I use the fact that the given ranges are Euclidean distances. The MATLAB command `fminsearch()` *(synonymous with* `fmins()` *or* `fsolve()` *in some versions) does multi-variable function minimization well enough for this problem. For this, use as your error function (to be minimized) the sum of the squared errors of all the range equations.*

These questions are in the flavor of the geometric dilution of precision (GDOP) problem in GPS navigation: the performance we get from the system is very strongly related to the physical arrangement of the satellites and target. Try the Wikipedia page on GDOP.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Three-dimensional Ranging
% MIT 2.017 FSH Sept 2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
global X Y Z R ;

X = [500 -500 500 0 ] ; % Satellite locations in Cartesian space
Y = [500 500  -500  0 ] ;
Z = [1000 1000 1000 0 ] ;
```

```
x = 33 ; y = -51 ; z = 950 ; % true target location

for i = 1:length(X), % calculate the ranges
    R(i) = sqrt((X(i)-x)^2 + (Y(i)-y)^2 + (Z(i)-z)^2) ;
end;

% plot the layout of sensors
figure(1);clf;hold off;
for i = 1:length(X),
    plot3([X(i) X(i)],[Y(i) Y(i)],[0 Z(i)]);
    hold on;
    plot3(X(i),Y(i),Z(i),'o','LineWidth',2);
end;
plot3([x x],[y y],[0 z],'r');
plot3(x,y,z,'rs','LineWidth',2);
grid;
xlabel('x, m'); ylabel('y, m'); zlabel('z, m');

% apply per-channel range errors to see what happens to the estimates
R(4) = R(4) + 1 ;

[posCalc] = fminsearch('rangeError',[0 0 0]); % search for the minimum
                                              % error solution

err = rangeError(posCalc) ;                   % get the consistency error
                                              % for this solution
disp(sprintf('Solution Consistency Error:  %g.', err)) ;

% here's the error with respect to the true solution
disp(sprintf('Solution Error wrt Actual:  %g.', norm(posCalc-[x y z],2)));

% add the points to the plot
xCalc = posCalc(1) ; yCalc = posCalc(2) ; zCalc = posCalc(3) ;
plot3([xCalc xCalc],[yCalc yCalc],[0 zCalc],'m');
plot3(xCalc,yCalc,zCalc,'m*','LineWidth',2);
text(xCalc+60, yCalc, zCalc, 'Target Estimate');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [err] = rangeError(pos);
global X Y Z R ;
```

```
x = pos(1) ; y = pos(2) ; z = pos(3) ;

err = 0 ;
for i = 1:length(R),
    err = err + ( R(i)^2 - (X(i)-x)^2 - (Y(i)-y)^2 - (Z(i)-z)^2 )^2 ;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

13 Numerical Solution of ODE's

In simulating dynamical systems, we frequently solve ordinary differential equations. These are of the form

$$\frac{dx}{dt} = f(t, x),$$

where the function f has as arguments both time and the state x. The time argument is used if the system is time-dependent (time-varying), but unnecessary if the system is time-invariant. Solving the differential equation means propagating x forward in time from some initial condition, say $x(t = 0)$, and typically the solution will be given as a vector of $[x(0), x(\Delta t), x(2\Delta t), \cdots]$, where Δt is a fixed time step.

The Taylor series is used to derive most of the simple formulas for solving ODE's. The general Taylor series expansion of the generic function g in two variables is

$$\begin{aligned} g(t + \Delta t, x + \Delta x) &= g(t, x) + \frac{\partial g}{\partial t}\Delta t + \frac{\partial g}{\partial x}\Delta x + \\ &\quad \frac{1}{2!}\frac{\partial^2 g}{\partial t^2}\Delta t^2 + \frac{1}{2!}\frac{\partial^2 g}{\partial t \partial x}\Delta t \Delta x + \frac{1}{2!}\frac{\partial^2 g}{\partial x \partial t}\Delta x \Delta t + \frac{1}{2!}\frac{\partial^2 g}{\partial x^2}\Delta x^2 + \cdots. \end{aligned}$$

In the above formula, when the arguments of g and its derivatives are not shown, we mean that it is to be evaluated at (t, x), that is g alone means $g(t, x)$ and so on. We use this shorthand below in several places. The simplest of all the ODE methods is *forward Euler*, created by setting $g = x$ and looking only at the first two terms on the right-hand side of the Taylor series:

$$x(t + \Delta t) = x(t) + \frac{dx}{dt}\Delta t = x(t) + \Delta t f$$

This formula says that x at the next time instant is the current x plus the time step times the slope *evaluated at the current* x. Referring to the Taylor series, we see that the forward Euler does not do anything with the second-order terms (Δt^2 and beyond), and so we say the method is second-order accurate in the step, which will turn out to be first-order accurate when you solve a real problem with many steps. First-order means that if you halve the time step, you can expect about half the error in the overall simulation.

An alternative, the Runge-Kutta methods are popular workhorses, and implemented in the MATLAB commands `ode23()` and `ode45()`. Let's take a look at the first of these: The rule is

$$\begin{aligned} k_1 &= \Delta t f \\ k_2 &= \Delta t f(t + \Delta t/2, x(t) + k_1/2) \\ x(t + \Delta t) &= x(t) + k_2. \end{aligned}$$

We see that k_1 is the same change in x as given by the forward Euler rule. k_2 is another guess for the change in x, but with the slope calculated at the approximate midpoint $[t +$

$\Delta t/2, x(t) + k_1/2]$. You can guess that this will be a somewhat more accurate result. Note, however, that we have to evaluate the function twice in each time step, as opposed to once each time step for the forward Euler method.

Here are your two problems:

1. The listed Runge-Kutta algorithm is third-order accurate in the step and second-order accurate in the whole: in each step, we get a cancelation of all the second-order terms in the Taylor series! Can you figure out how to show this?

   Solution: We have to show that the Runge-Kutta formula captures up to second-order terms in the Taylor series. First, the formula for k_2 is itself approximated as a Taylor series:

$$\begin{aligned} k_2 &= \Delta t f(t + \Delta t/2, x + k_1/2) \\ &= \Delta t \left[f + \frac{\Delta t}{2}\frac{\partial f}{\partial t} + \frac{k_1}{2}\frac{\partial f}{\partial x} + h.o.t. \right] \\ &= \Delta t \left[f + \frac{\Delta t}{2}\frac{\partial f}{\partial t} + \frac{\Delta t}{2}\frac{\partial f}{\partial x} f + h.o.t. \right], \end{aligned}$$

   where we again assume evaluation of functions at $x(t)$ and t when the argument is omitted. The first relation comes from the bivariate Taylor series, and the second by substituting in k_1. The acronym h.o.t. stands for "higher-order terms." Next, write out $x(t + \Delta t)$ according to the last line of the RK2 rule:

$$\begin{aligned} x(t + \Delta t) &= x + k_2 \\ &= x + \Delta t f + \frac{\Delta t^2}{2}\left[\frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} f + h.o.t. \right] \end{aligned}$$

   You recognize the first two terms in the square brackets here as the total derivative of (bivariate) f with t, which is the second derivative of x with time. How does this result look compared to the direct Taylor series for x? We have for the univariate Taylor series applied to x:

$$x(t + \Delta t) = x + \Delta t \frac{dx}{dt} + \frac{\Delta t^2}{2}\frac{dx^2}{dt^2} + h.o.t.$$

   So the RK2 formula does indeed capture the first- and second-order terms of the Taylor series. Note that the h.o.t. that appears when we worked with k_2 is <u>not</u> the same as the h.o.t. that appears in the direct expansion of x - so we cannot claim that the third-order terms are captured in the RK2 rule. RK4 (shown below) captures both the third- and fourth-order terms!

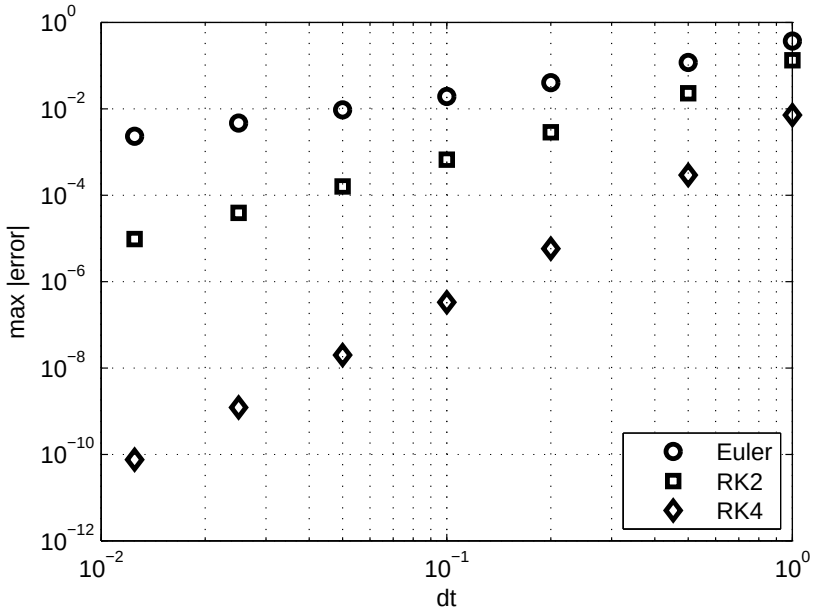
2. Compare the error convergence rates, for the forward Euler and the second-order Runge-Kutta on the simple problem $dx/dt = -x$, $x(t = 0) = 1$. To do this, you will make several simulations with each method, over the time scale $[0, 5]$, and with

$\Delta t = [0.0125\ 0.025\ 0.05\ 0.10\ 0.20\ 0.50\ 1.00]$. For each, record the maximum absolute error with respect to the exact solution (a decaying exponential), and then make a summary plot of Δt versus error, including both methods. You should be able to see the first-order convergence of forward Euler and the second-order convergence of RK2.

As an aside, forward Euler has *stability problems* that are more severe than for the Runge-Kutta family, and this is another reason why forward Euler should usually not be your first choice.

Please do not use the canned MATLAB ODE functions to solve the second problem. I am asking you to make the explicit calculations for programming practice, to see the error rates first-hand, and because we sometimes don't have MATLAB available where needed, e.g., on an embedded processor.

Solution: See MATLAB results and code below. The Euler method is first-order: a ten-fold reduction in Δt gives a ten-fold reduction in error. RK2 is second-order: the ten-fold reduction in Δt gives a one-hundred-fold reduction in error. Most impressive, the RK4 is fourth-order so we get a ten-thousand-fold improvement for a ten-fold reduction in Δt. Note that RK4 requires four evaluations per time step, and RK2 requires two. But it is usually well worth it! Finally, notice that the RK2 and the forward Euler both give awful errors with $\Delta t = 1$. This is no surprise, since the time constant of the system is one also. The RK4 gives us a much better result here, the result of those extra evaluations.




```
Errors for the Different dt's and Methods:
dt: 0.013     Euler: 2.31e-003   RK2: 9.67e-006   RK4: 7.56e-011
dt: 0.025     Euler: 4.65e-003   RK2: 3.90e-005   RK4: 1.22e-009
dt: 0.050     Euler: 9.39e-003   RK2: 1.59e-004   RK4: 2.00e-008
```

```
dt: 0.100     Euler: 1.92e-002   RK2: 6.62e-004   RK4: 3.33e-007
dt: 0.200     Euler: 4.02e-002   RK2: 2.86e-003   RK4: 5.80e-006
dt: 0.500     Euler: 1.18e-001   RK2: 2.27e-002   RK4: 2.91e-004
dt: 1.000     Euler: 3.68e-001   RK2: 1.32e-001   RK4: 7.12e-003
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Show the error properties of forward Euler vs.
% second-order and fourth-order Runge-Kutta integration schemes.

function simEulerRK2
clear all;

tfinal = 5 ;    % final simulation time
dtvector = [.0125 .025 .05 .1 .2 .5 1] ; % vector of dt's to try
xinitial = 1 ; % initial value of x

figure(1);clf;hold off;
figure(2);clf;hold off;
subplot('Position',[.2 .2 .5 .5]);

disp('Errors for the Different dt''s and Methods:');
fn = fopen('simEulerRK2.dat','w');
fprintf(fn,'Errors for the Different dt''s and Methods:\n');

for dt = dtvector,  % cycle through all the dt's

    n = tfinal/dt ;             % the number if steps to take
    timevector = 0:dt:n*dt ;    % vector of times (length n+1)
    exact = exp(-timevector);   % exact solution to the diff eqn:
                                % x' = f(t,x)

    xEuler(1) = xinitial ; % initial value for the Euler rule
    xRK2(1) = xinitial ; % initial value for the RK2 rule
    xRK4(1) = xinitial ; % initial value for the RK4 rule

    % do the Euler version
    for i = 1:n,
        t = timevector(i) ;
        xEuler(i+1) = xEuler(i) + dt*f(t,xEuler(i)) ;
    end;

    % do the RK2 version
    for i = 1:n,
        t = timevector(i) ;
        k1 = dt*f(t,xRK2(i)) ;
        k2 = dt*f(t+dt/2, xRK2(i)+k1/2) ;
        xRK2(i+1) = xRK2(i) + k2 ;
    end;
```

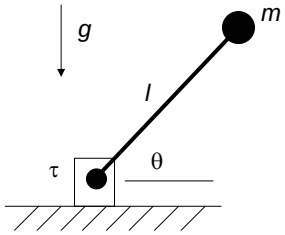
```
    % do the RK4 version!
    for i = 1:n,
        t = timevector(i) ;
        k1 = dt*f(t,xRK4(i)) ;
        k2 = dt*f(t+dt/2, xRK4(i)+k1/2) ;
        k3 = dt*f(t+dt/2, xRK4(i)+k2/2) ;
        k4 = dt*f(t+dt, xRK4(i)+k3) ;
        xRK4(i+1) = xRK4(i) + (k1 + 2*k2 + 2*k3 + k4) / 6 ;
    end;

    figure(1);
    plot(timevector,log10(abs(xEuler-exact)),'b',...
        timevector,log10(abs(xRK2-exact)),'r',...
        timevector,log10(abs(xRK4-exact)),'g','LineWidth',2) ;
    hold on;
    grid;

    disp(sprintf(...
        'dt: %5.3f     Euler: %6.2e   RK2: %6.2e   RK4: %6.2e',...
        dt, max(abs(xEuler-exact)), max(abs(xRK2-exact)), ...
        max(abs(xRK4-exact))));

    fprintf(fn,...
        'dt: %5.3f     Euler: %6.2e   RK2: %6.2e   RK4: %6.2e\n',...
        dt, max(abs(xEuler-exact)), max(abs(xRK2-exact)), ...
        max(abs(xRK4-exact)));

    figure(2);
    loglog(dt, (max(abs(xEuler-exact))),'o',...
        dt,(max(abs(xRK2-exact))),'s', ...
        dt, (max(abs(xRK4-exact))),'d','LineWidth',2);
    hold on;
    if dt == max(dtvector),
        legend('Euler','RK2','RK4',4);
        xlabel('dt');
        ylabel('max |error|');
        grid;
    end;

    clear n timevector exact xEuler xRK2 xRK4 t k1 k2 k3 k4 ;
end;
fclose(fn);
figure(2);
print -deps simEulerRK2.eps
```

```
% here is the function definition
function [dxdt] = f(t,x) ;
    dxdt = -x ;
```

%%%

14 Pendulum Dynamics and Linearization

Consider a single-link arm, with length l and all the mass m concentrated at the end. A motor at the fixed pivot point supplies a controllable torque τ. As drawn, a positive torque drives the arm counter-clockwise, so as to drive the arm angle θ positive. The arm is free to swing around the full 360 degrees; gravity pulls the arm downward. There is no damping.



1. Derive and state the equation of motion for this system.

   Solution: It is a basic rotary moment of inertia with a gravity effect and input torque. We will get

   $$ml^2\ddot{\theta} = \tau - mgl\cos\theta.$$

2. For each of the linearization angles [-90,0,90] degrees, answer the following:

   (a) What is the static torque needed to support the arm in this configuration?

   Solution: For the three angles given, the static torque is the amount needed to just balance the gravity torque. These values are [0, mgl, 0]; note that the upward and downward configurations do not take any static torque to maintain.

   (b) Assuming that the static torque is continuously applied, but that there is no dynamic torque, what is the equation of motion for small deviations from the static angle? We are looking for a differential equation in the perturbation of θ, say $\tilde{\theta}$.

   Solution: The key here is to write the angles and the torques as being comprised of a static ($\bar{\theta}, t\bar{a}u$) and a dynamic part ($\tilde{\theta}, \tilde{\tau}$):

   $$\begin{aligned} ml^2\frac{d}{dt}\left(\bar{\theta}+\tilde{\theta}\right) &= \bar{\tau}+\tilde{\tau}-mgl\cos(\bar{\theta}+\tilde{\theta})+d \\ ml^2\frac{d}{dt}\tilde{\theta} &= \bar{\tau}-mgl\cos\bar{\theta}\cos\tilde{\theta}-mgl\sin\bar{\theta}\sin\tilde{\theta}+d. \end{aligned}$$

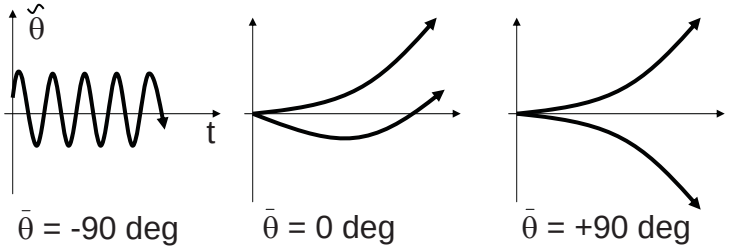
   Along with a trigonometric identity, we used the facts that dynamic torque $\tilde{\tau}$ is zero, and that static angle $\bar{\theta}$ has zero second derivative. We can see that $\cos\tilde{\theta}$

linearizes to simply one, and so the first two terms on the right-hand side in the second line give us precisely the static torques of part (i). The term d represents a torque perturbation. We are left with the following linear equations for the three static angles [-90, 0, 90] degrees, respectively:

$$\begin{aligned} mgl\ddot{\tilde{\theta}} &= -mgl\tilde{\theta} + d \\ mgl\ddot{\tilde{\theta}} &= d \\ mgl\ddot{\tilde{\theta}} &= mgl\tilde{\theta} + d. \end{aligned}$$

(c) In words and in an annotated graph, show what is the response of the equilibrium system to a (small) torque impulse. Be sure to take into account the fact that the system could respond differently to impulses of opposite directions.

Solution: The first differential equation above is that of a second-order, undamped oscillator. A perturbation to the equilibrium will lead to oscillations of constant size. The second equation above is a "double integrator," like a mass on a sliding contact; an impulsive d will set the arm spinning at constant speed, with no damping or restoring force. Of course, as the arm travels our linearizing assumptions (namely that $\tilde{\theta}$ is small) fail, and we see some other interesting behavior. If the perturbation is upward, the static torque will be larger than the gravity torque, and the arm will accelerate in the positive direction, up and over the top! If the perturbation is negative, we can again expect an acceleration in the positive direction for the same reasons, again going up and over. It doesn't get stuck at zero degrees because there is no damping and inertia will carry it through. Finally, the upward static configuration is just plain unstable because it is as if we have a negative spring term - any perturbation will cause the arm to fall over.



15 Bouncing Robot

You are asked to explore the simple dynamics of a bouncing robot device, using simulation. There is a single mass with a very light helical spring attached on the bottom of it; the mass is $40kg$, and the spring constant is $10400N/m$. The spring additionally has a little bit of damping, $35Ns/m$. The spring is NOT attached to the ground. The initial condition has zero vertical velocity and a $15cm$ compression of the spring.

1. Formulate the system dynamics with an annotated drawing and a statement of the governing equations.

   Solution: Let us say that vertical position $y = 0$ corresponds with the spring in its natural, uncompressed state, and it is just touching the ground. Then when $y < 0$, there is a compression force pushing the mass upward at ky, and the associated damping $b\dot{y}$. When $y > 0$, there is no spring force at all. In all cases, there is a steady gravity force acting. We have then:

$$\begin{aligned} m\ddot{y} + b\dot{y} + ky &= -mg \text{ when } y < 0 \text{ , or} \\ m\ddot{y} &= -mg \text{ when } y > 0. \end{aligned}$$

2. Is this a linear system - why or why not? Since we are not forcing the system, you can answer the question by considering how the response changes as you scale the initial conditions.

   Solution: The system is nonlinear. If we apply a very large preload before release, we will get a large hopping response, whereas if we apply a very small preload the mass may not hop at all, because the spring doesn't fully unload.

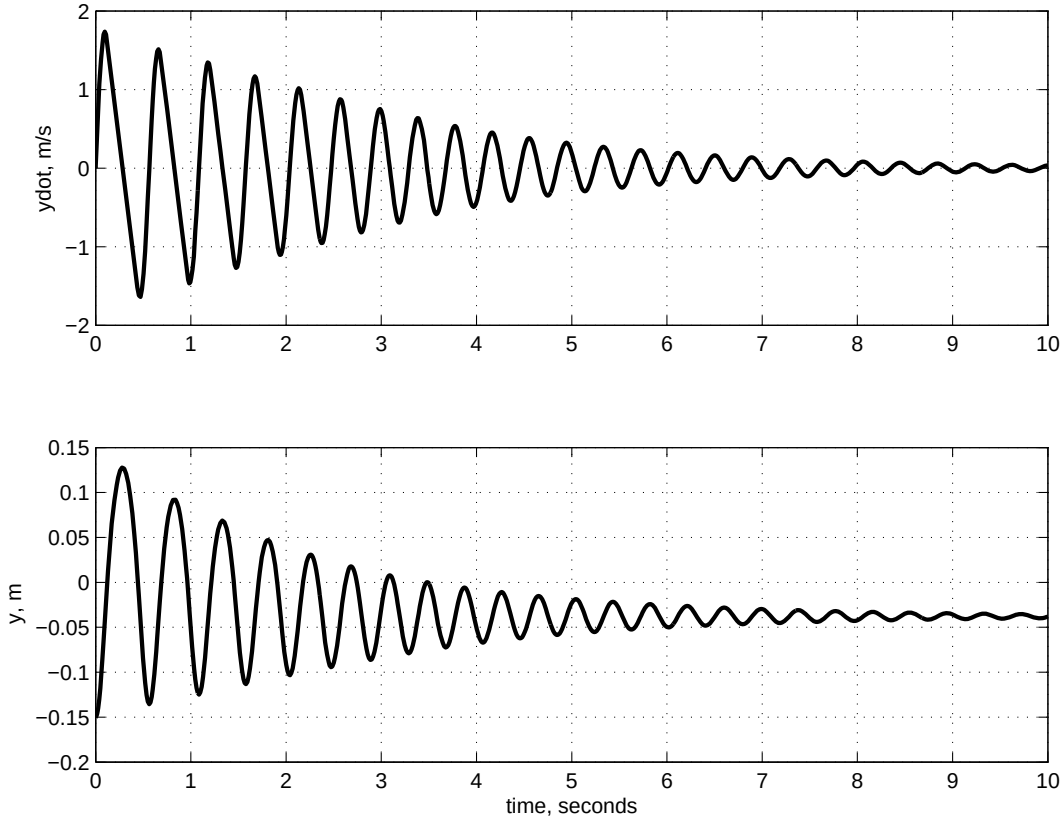
3. Run a simulation to get the response up to time ten seconds. Show in a plot both the vertical position and the velocity as a function of time.

   Solution: See that attached code and figure. Note that when the spring is unloaded, the velocity is changing linearly with time (constant acceleration), whereas when the spring is loaded, a sinusoidal velocity profile occurs.

4. Answer the specific question: At what time after the release does the lower end of the spring stop leaving the ground?

   Solution: The spring does not leave the ground again after 3.48 seconds.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulate a bouncing mass

function bouncingMass

clear all;

[t,y] = ode45(@func,[0 10],[0 -0.15]) ;

figure(1);clf;hold off;
subplot(211);
plot(t,y(:,1),'LineWidth',2); grid;
ylabel('ydot, m/s');
subplot(212);
plot(t,y(:,2),'LineWidth',2); grid;
ylabel('y, m');
xlabel('time, seconds');

function [ydot] = func(t,y) ;
                % all units MKS
m = 40 ;        % mass
k = 10400 ;     % spring constant
g = 9.81 ;      % gravity
b = 35 ;        % spring damping

% first state is vertical velocity, second is position

if y(2) < 0,            % the spring is in action ...
    ydot(1,1) = 1/m * ( -k*y(2) - b*y(1)) - g ;
else,                   % ... or not
    ydot(1,1) = -g ;
end;

ydot(2,1) = y(1) ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```


ydot, m/s
2
1
0
−1
−2
0
1
2
3
4
5
6
7
8
9
10
y, m
0.15
0.1
0.05
0
−0.05
−0.1
−0.15
−0.2
time, seconds

16 Road Vehicle on Random Terrain

An autonomous, four-wheeled road vehicle travels over an uncertain terrain. You are asked to characterize the vertical motion of the center of gravity, and the pitching (nose-up vs. nose-down) response of the vehicle, as it crosses the terrain, using a *linear model*. We know that the vehicle chassis has a mass of $m = 42kg$ and a pitching moment of inertia of $J = 25kg\ m^2$. Consider the tires and suspension to have zero mass, but the front and back suspension systems are each modeled as having a spring of stiffness $k = 1000N/m$, in parallel with a linear damper having coefficient $b = 100N/(m/s)$. The distance between the front and rear wheels is $2l = 0.9m$. The vehicle travels at a speed U, which we will vary below. Because we consider U to be constant, you can use it directly to map between the horizontal terrain coordinate x and time t in this problem.

The terrain roughness has been described only statistically (e.g., using an orbiting camera or some other remote sensing system). Its content is given by the following harmonic components:

n	wave number, 1/m ($k_n = 2\pi/\lambda_n$)	amplitude, m (a_n)
1	0.250	0.01
2	0.275	0.02
3	0.300	0.02
4	0.325	0.04
5	0.350	0.03
6	0.375	0.02
7	0.400	0.01
8	0.425	0.03
9	0.450	0.05
10	0.475	0.01
11	0.500	0.02

Here, k_n is the n'th wave number (spatial frequency) and λ_n is the n'th wavelength (spatial period). We model the ground elevation as a function of the horizontal coordinate:

$$h(x) = \sum_n a_n \sin(k_n x + \phi_n).$$

Notice that $2l << \min(\lambda)$; the vehicle is much shorter than the smallest wavelength, so you can approximate

$$\begin{aligned} h(x) &\approx [h(x+l) + h(x-l)]/2, \\ h'(x) = dh(x)/dx &\approx [h(x+l) - h(x-l)]/2l, \\ h''(x) = d^2h(x)/dx^2 &\approx [h'(x+l) - h'(x-l)]/2l, \quad \text{and so on.} \end{aligned}$$

1. Develop a model that describes the response of the vehicle to terrain variations $h(x)$. It should comprise two uncoupled second-order differential equations, one for vertical motion and the other for pitch. Make an annotated figure to go with the equations. You may like to use the fact that the vertical velocity of the ground *seen from a reference frame moving horizontally at velocity* U, is $Uh'(x)$.

   Newton's laws and some geometry give us

$$\begin{aligned} m\ddot{z} &= -k[(z+l\theta-h(x+l))+(z-l\theta-h(x-l))] \\ &\quad -b[(\dot{z}+l\dot{\theta}-Uh'(x+l))+(\dot{z}-l\dot{\theta}-Uh'(x-l))] \\ &\approx -k[2z-2h(x)]-b[2\dot{z}-2Uh'(x)] \end{aligned}$$

$$\begin{aligned} J\ddot{\theta} &= -kl[(z+l\theta-h(x+l))-(z-l\theta-h(x-l))] \\ &\quad -bl[(\dot{z}+l\dot{\theta}-Uh'(x+l))-(\dot{z}-l\dot{\theta}-Uh'(x-l))] \\ &\approx -kl[2l\theta-2lh'(x)]-bl[2l\dot{\theta}-2lUh''(x)]. \end{aligned}$$

   As advertised, these equations are uncoupled, except through the input $h(x)$ *and its derivatives.*

2. What are the damped and undamped natural frequencies of the two systems? What are the damping ratios? List all six numbers, with units.

   For the vertical motion, the undamped natural frequency is $\omega_n = \sqrt{2k/m} \approx 6.9 rad/s$, *and the damping ratio is* $\zeta = 2b/2m\omega_n \approx 0.35$, *giving a damped natural frequency of* $\omega_d = \omega_n\sqrt{1-\zeta^2} \approx 6.5 rad/s$. *For the pitch motion, the undamped natural frequency is* $\omega_n = \sqrt{2l^2k/J} \approx 4.0 rad/s$, *and the damping ratio is* $\zeta = 2l^2b/2J\omega_n \approx 0.20$, *giving a damped natural frequency of* $\omega_d = \omega_n\sqrt{1-\zeta^2} \approx 3.9 rad/s$. *Both of these modes are rather lightly damped and so may exhibit resonance behavior.*

3. Create a set of elevation data $h(x)$ based on the above (spatial) frequency content, and plot it. Consider the range of x from zero to twenty times the longest wavelength; your spacing in x should be no more than one-tenth of the smallest wavelength, to get a good picture. Use a random phase angle ϕ_n for each of your components.

   See the attached figures and code. Note that I used fifty cycles of the longest wavelength (not twenty).

4. What is the maximum amplitude of elevation $h(x)$ in the terrain data you created? What is the maximum amplitude of the *slope*?

   The maximum amplitude shown is about $0.21m$, *with a maximum slope of about* $4.8°$. *These values come out differently each time the program is run because the phases are determined randomly.*

5. Now run your vehicle model over this terrain at $U = 5m/s$. What is the maximum amplitude of the vertical motion? What is the maximum amplitude of the pitch motion?

The maximum vertical motion amplitude is about $0.23m$ *and the pitch is about* $6.4°$ *- both of these are larger than the terrain itself, indicating that some dynamic effect is occurring.*

6. Repeat this calculation for speeds of 10, 15, 20, and $25m/s$. Make a tabular listing for the vehicle's maximum amplitudes of vertical motion and pitch, as a function of U. Are the motions consistent with the systems' resonant frequencies and damping properties?

   The vertical motion amplitudes for $U = [5, 10, 15, 20, 25]m/s$ *are approximately* $[0.23, 0.29, 0.34, 0.29, 0.23]m$*, respectively. The peak amplification is* $0.34/0.23$ *or almost fifty percent, consistent with the damping ratio we found. The driving frequencies from the terrain are given by* $\omega_n = Uk_n$*, so that* $15m/s \times 0.45rad/m = 6.75rad/s$ *- a high-amplitude harmonic input (0.05m) is occurring very close to the damped resonant frequency. The pitch motion amplitudes are approximately* $[6.4, 9.9, 4.9, 2.2, 1.3]°$*. The peak amplification is over two, consistent with the lower damping ratio in pitch. The slope of any particular terrain harmonic goes as* $k_n a_n$*, so again the input at* $k = 0.45rad/m$ *is the dominant one.* $10m/s \times 0.45rad/m = 4.5rad/s$*, again pretty close to our damped natural frequency.*

7. What different effects could we expect if the wheelbase became long compared to some of the terrain harmonic components? (No calculations.)

   We would start to see spatial aliasing. This would grotesquely deform our neat explanations above, and invalidate the approximations we used for the slopes and average values of $h(x)$*. This latter point could, however, be easily fixed in the code simply by carrying out the full expressions, that is, using* $h(x + l)$*,* $h(x - l)$*, and so on.*

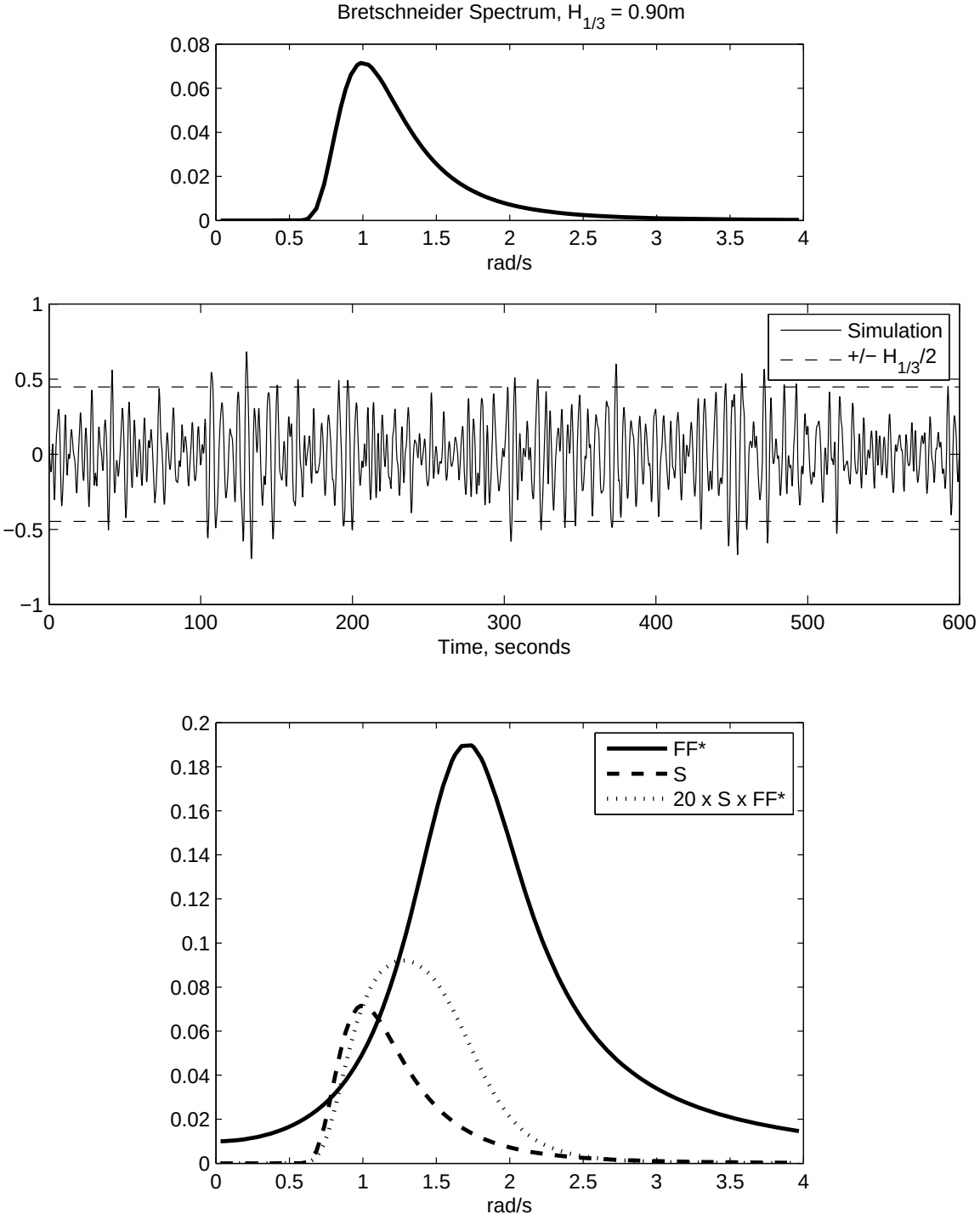
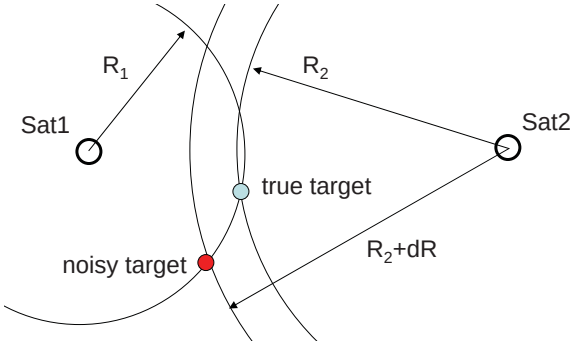
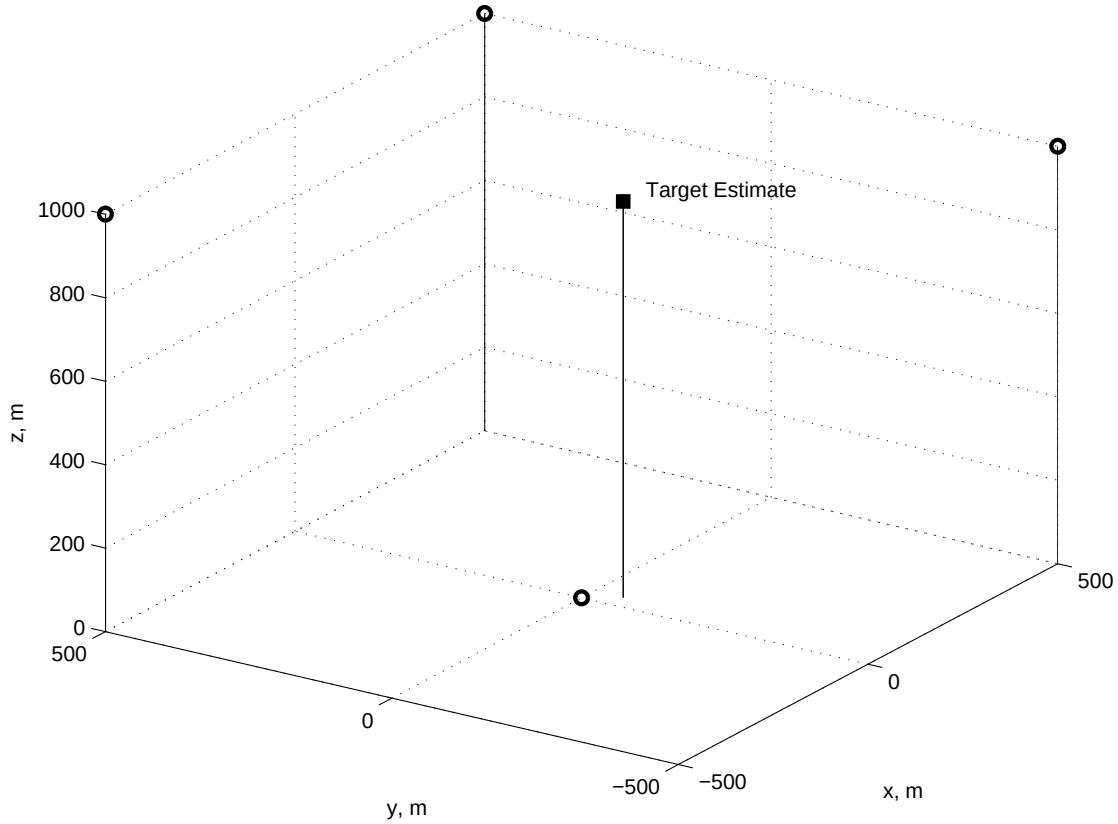
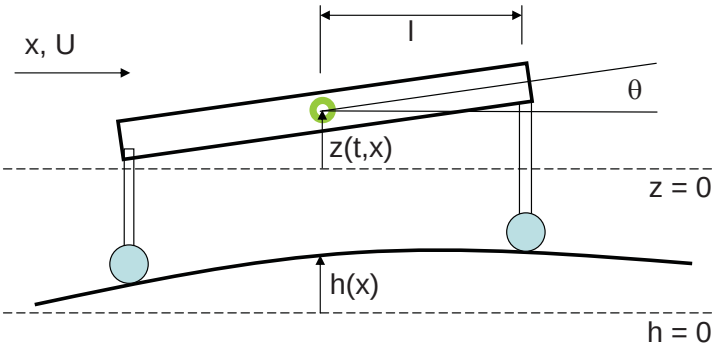
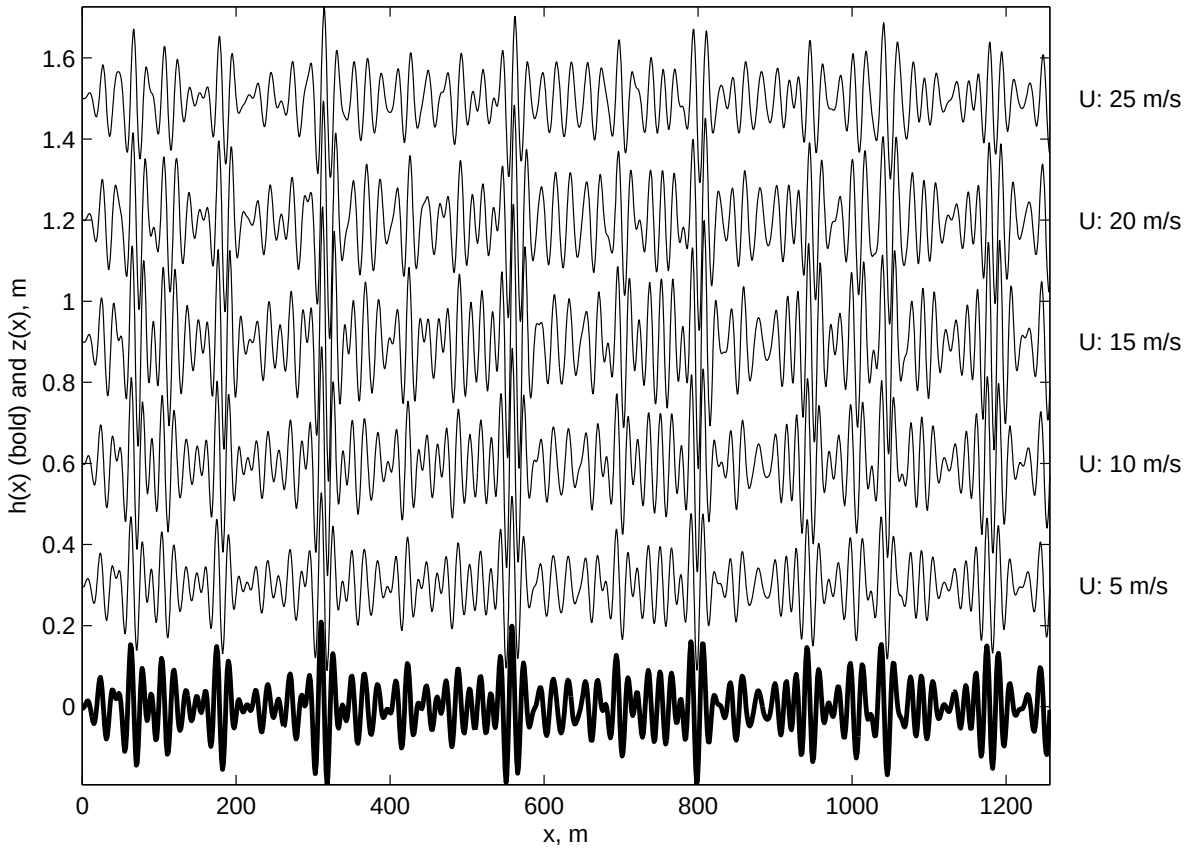


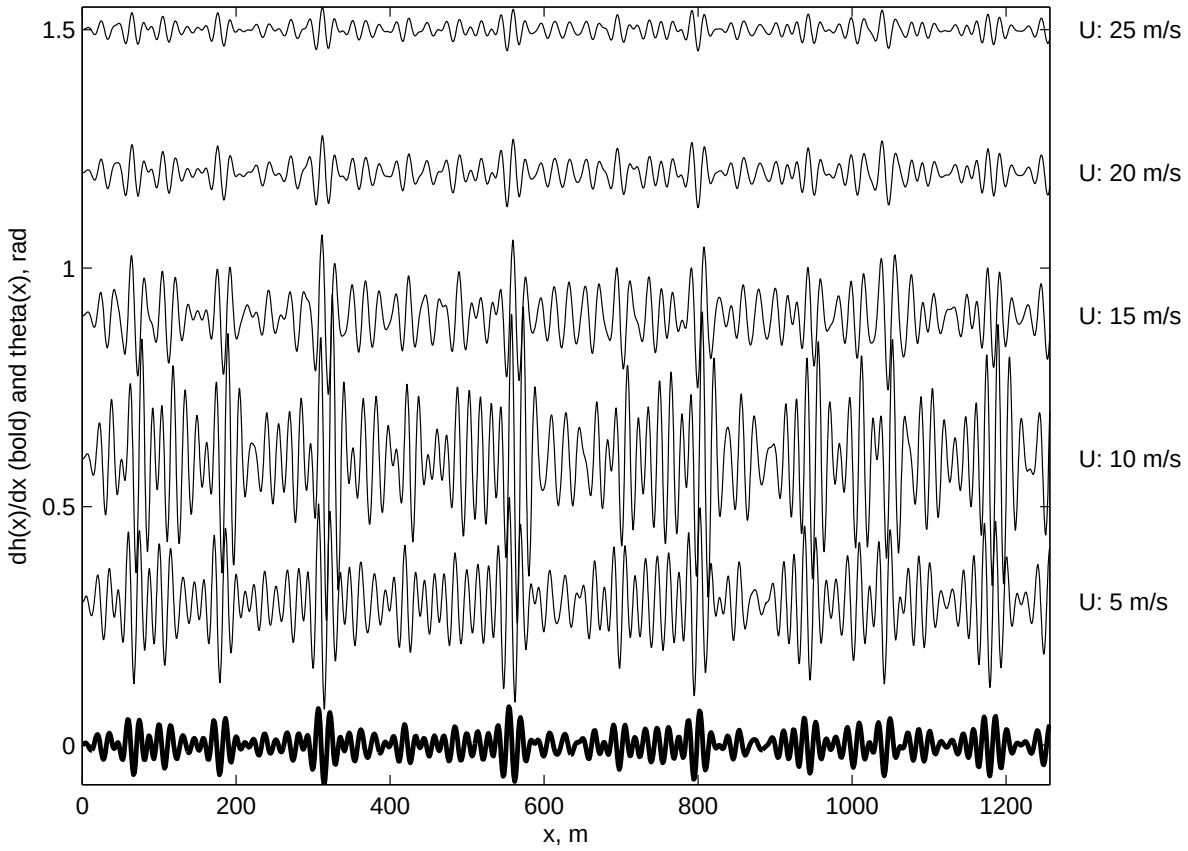
Figure 2: Bretschneider spectrum and vehicle pitch response spectra.


R1
R2
Sat1
Sat2
true target
noisy target
R2+dR


Target Estimate
1000
800
600
400
200
0
z, m
500
0
y, m
−500
−500
0
500
x, m


x, U
l
θ
z(t,x)
z = 0
h(x)
h = 0


h(x) (bold) and z(x), m
x, m
0
200
400
600
800
1000
1200
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
U: 25 m/s
U: 20 m/s
U: 15 m/s
U: 10 m/s
U: 5 m/s


1.5
1
0.5
0
dh(x)/dx (bold) and theta(x), rad
0
200
400
600
800
1000
1200
x, m
U: 25 m/s
U: 20 m/s
U: 15 m/s
U: 10 m/s
U: 5 m/s

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Land Vehicle Moving Over Rough Terrain
% MIT 2.017 FSH Sept 2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
global U m J l k b phi a kk;

k = .25:.025:.5 ; % terrain wavenumbers
a = [.01 .02 .02 .04 .03 .02 .01 .03 .05 .01 .02] ; % amplitudes

m = 42 ; % mass of body
J = 25 ; % rotary moment of inertia about centroid
l = .45 ; % half-width between wheels, m
kk = 1000 ; % strut stiffness, N/m
b = 100 ; % strut damping coefficient

phi = 2*pi*rand(length(k),1) ; % set the random phase angles for this run

% set up and calculate a few derived items

vertDrawingScale = .3 ; % drawing parameters
pitchDrawingScale = .3 ;

if length(a) ~= length(k),
    disp('Inconsistent k and a!');
    break;
end;

lam = 2*pi./k ; % wavelengths

dk = mean(diff(k));
k = k + rand(size(k))*dk/2 ; % randomize the wavenumbers a little bit

dx = min(lam)/10; % pick x-spacing to get ten points in the fastest cycle
xvec = 0:dx:max(lam)*50; % a vector of x-values we'll look at; this
                         % gets 50 cycles of the slowest harmonic

% Characterize the resonance and damping properties of the two LTI systems

disp('--------------------------');
wnVert = sqrt(2*kk/m) ;
disp(sprintf('Undamped natural freq. of vertical system:  %g rad/s', wnVert));
```

```
zetaVert = 2*b / 2 /sqrt(2*kk*m);
disp(sprintf('Damping ratio of vertical system:  %g', zetaVert)) ;

if zetaVert < 1,
    wdVert = wnVert*sqrt(1-zetaVert^2) ;
    disp(sprintf('Damped natural freq. of vertical system:  %g rad/s', wdVert));
else,
    disp('Vertical system is overdamped.');
    wdVert = NaN ;
end;

disp(' ');

wnPitch = sqrt(2*l^2*kk/J) ;
disp(sprintf('Undamped natural freq. of pitch system:  %g rad/s', wnPitch));

zetaPitch = 2*l^2*b / 2 / sqrt(2*l^2*kk*J) ;
disp(sprintf('Damping ratio of pitch system:  %g', zetaPitch)) ;

if zetaPitch < 1,
    wdPitch = wnPitch*sqrt(1-zetaPitch^2) ;
    disp(sprintf('Damped natural freq. of pitch system:  %g rad/s', wdPitch));
else,
    disp('Pitch system is overdamped.');
    wdPitch = NaN ;
end;

% build up the elevation profile for this set of phase angles
for i = 1:length(xvec),
    x = xvec(i) ;
    h(i) = 0 ;
    dhdx(i) = 0 ;
    for j = 1:length(k),
        h(i) = h(i) + a(j)*sin(k(j)*x + phi(j)) ;
        dhdx(i) = dhdx(i) + a(j)*k(j)*cos(k(j)*x + phi(j)) ;
    end;
end;

figure(1);clf;hold off;
subplot('Position',[.15 .15 .7 .75]);
plot(xvec,h,'LineWidth',2) ;
xlabel('x, m');
ylabel('h(x) (bold) and z(x), m');
```

```
figure(2);clf;hold off;
subplot('Position',[.15 .15 .7 .75]);
plot(xvec,dhdx,'LineWidth',2);
xlabel('x, m');
ylabel('dh(x)/dx (bold) and theta(x), rad');

disp(' ');
disp(sprintf(...
   'Terrain:   max|h| %5.3f m   max|dh/dx| %4.2f rad / %4.2f deg',...
    max(abs(h)), max(abs(dhdx)), max(abs(dhdx))*180/pi));

i = 1 ;
for U = [5 10 15 20 25],
    % simulate the system  behavior over this terrain
    clear t s ;
    [t,s] = ode45('vehicleOnTerrainDeriv',[0 max(xvec)/U], [0 0 0 0]');

    figure(1);
    hold on;
    plot(t*U,s(:,2) + vertDrawingScale*i);
    axis('tight');
    text(max(t*U)*1.03,mean(s(:,2)) + vertDrawingScale*i,...
        sprintf('U: %d m/s',U));
    figure(2);
    hold on;
    plot(t*U,2*s(:,4) + pitchDrawingScale*i) ;
    axis('tight');
    text(max(t*U)*1.03,mean(s(:,4)) + vertDrawingScale*i,...
        sprintf('U: %d m/s',U));

    disp(sprintf(...
        'U %2d m/s:  max|z| %5.3f m   max|theta| %4.2f rad / %4.2f deg',...
        U, max(abs(s(:,2))), max(abs(s(:,4))), max(abs(s(:,4)))*180/pi));

    pause(.1);
    i = i + 1 ;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dsdt] = vehicleOnTerrainDeriv(t,s) ;
global U m J l k b phi a kk;

x = U*t ; % horizontal location of the vehicle
```

```
% build up the elevation and its derivatives at the current time and x
h = 0 ;
dhdx = 0 ;
ddhddx = 0 ;
for j = 1:length(k),
    h = h + a(j)*sin(k(j)*x + phi(j)) ;
    dhdx = dhdx + a(j)*k(j)*cos(k(j)*x + phi(j)) ;
    ddhddx = ddhddx - a(j)*k(j)^2*sin(k(j)*x + phi(j)) ;
end;

% parse out the state vector
zdotIn = s(1);
zIn = s(2) ;
thetadotIn = s(3) ;
thetaIn = s(4) ;

% here are the dynamic equations
zddot = -kk/m*(2*zIn - 2*h) - b/m*(2*zdotIn - 2*U*dhdx);
zdot = zdotIn ;
thetaddot = -kk*l/J*(2*l*thetaIn - 2*l*dhdx) - ...
    b*l/J*(2*l*thetadotIn - 2*l*U*ddhddx) ;
thetadot = thetadotIn ;

% build the state vector derivative
dsdt = [zddot zdot thetaddot thetadot]' ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

17 Dynamics Calculations Using the Time and Frequency Domains

1. *Don't turn anything in on this part.* The FFT (Fast Fourier Transform) is a standard method for decomposition of a signal into its harmonic components. You need to know a few specific items about it:

   (a) The length of the input signal x - in the time domain - is N. The FFT itself doesn't care about what is the time step between x_n and x_{n+1}, although the spacing must be uniform; below, we'll call it Δt. Hence x_1 occurs at time 0, x_2 is at time Δt, and so on, up to x_N taken at time $(N-1)\Delta t$.

   (b) The formal definition of the transform and its inverse in MATLAB is

$$\begin{aligned} X_k &= \sum_{n=1}^{N} x_n e^{-i2\pi(k-1)(n-1)/N}, 1 \le k \le N; \\ x_n &= \frac{1}{N}\sum_{k=1}^{N} X_k e^{i2\pi(k-1)(n-1)/N}, 1 \le n \le N, \end{aligned}$$

   where $i = \sqrt{-1}$. So the FFT takes a uniformly-spaced time signal x and makes a uniformly-spaced frequency signal X, of the same length. It is like a Fourier integral because of the exponential: $e^{iz} = \cos z + i \sin z$. Furthermore, you notice that the FFT (the transform that makes X) is a summation between the time-domain signal x and an exponential with imaginary argument, which has unity magnitude. Thus, the size of the elements in X are roughly the size of x, multiplied by N. The inverse FFT has the factor $1/N$ out front so as to cancel this effect out; `ifft(fft(x)) = x;`. Because of this arrangement, however, you can't take the FFT of two signals, multiply them (to effect a convolution), take the inverse FFT, and expect to get the right scaling - each FFT induces a scaling of N, but the IFFT only corrects for one of them. Remembering this point will save you a lot of trouble in using the FFT in MATLAB!

   You can see this scaling for example with
   `x = sin(0:0.1:100) ; X=fft(x);plot(abs(X));`. Here, you plot the absolute value (`abs()`) which is synonymous with magnitude, because `X` is complex.

   There is another scaling factor to notice, but this is easier; because the Fourier integral involves both negative and positive frequencies, it puts half the area on each side.

   (c) What are the uniformly-spaced frequencies that go with X? For the case of odd N (typical if the final time T is an even multiple of Δt), we have:

$$\omega_k = \left[\begin{array}{cccccccc} 0 & 1 & \cdots & \frac{N-1}{2} & \frac{1-N}{2} & \cdots & -1 \end{array} \right] \frac{2}{N-1}\frac{\pi}{\Delta t}.$$

   If you count these, you will see that there are N of them, exactly one frequency to go with each element of X. More interesting are the values, which start from

zero and go to the Nyquist rate $\pi/\Delta t$, and then jump back to the negative of the Nyquist rate, and go to *almost* zero. Together, these cover completely the range of negative Nyquist rate to positive Nyquist rate. The FFT can't see beyond the Nyquist rate, so we see that, in general, increasing N but keeping ΔT constant will give more resolution to a Fourier decomposition.

You can try making up the frequency vector that goes with the example above: Do the indices of the peak value in `X` correspond with plus and minus one radian per second?

2. Now we use the FFT as a tool for analyzing system response to both specific and, later, to stochastic forcing signals.

   First, find the impulse response of the system $x'' + 0.1x' + x = u$, using your simulation based on `ode45`. You'll have to construct the impulse explicitly in the derivative call, with something like this:

   ```
   if t < epsilon, u0 = 1/epsilon ; else, u0 = 0 ; end;
   ```

   You can try values of $\epsilon < 0.1$ or so for this system. The response we are looking for is x, not x', so you'll have to pick one column only out of the vector that `ode45` returns; call it `h`. Use an initial time of zero in the simulation, and final time $T = 100$ seconds.

3. Regularize the impulse response you calculated by first making a uniformly spaced time vector like this: `treg = 0:dt:T`; a value for `dt` of 0.1 or so is OK. This `treg` is contrasted with the time vector returned by `ode45`, which is *not* uniformly spaced, because MATLAB is automatically adjusting the time step as it does the integration for you. Use the spline function to make `h` line up with the uniform time vector `treg`: `[hreg] = spline(t,h,treg) ;`.

4. Now, make up a new input u, that we will use in the rest of this problem: $u = 1$ if $t < T/2$, $u = 0$ if $t \geq T/2$. You need to put this in as an input in the derivative function for the simulation, and you need to also create it as a vector `ureg` that lines up with `treg`.

5. Now we are ready to make the computation of the system response to the input u in three different ways! The **first** is the simplest - just run the simulation with the input u. Make a plot of x vs. t.

6. The **second** method is to convolve the input u with the impulse response h. We have to use a consistent time base, and that is why we made `ureg` and `hreg`. MATLAB has a command called `conv` which you can use; the convolution integral is the same as this vector convolution if you multiply the latter by `dt`; i.e., we have to calculate the *area* under the curve.

   Plot the first N elements of the convolution result versus `treg`. It should look almost the same as what you found from the direct simulation.

7. The **third** method is to multiply the FFT of the impulse response with the FFT of the input u. In this case, you will do a vector pointwise multiplication (`.*` is the syntax)

of `U = fft(ureg)` and the FFT of `h`. You could execute `fft(hreg)` of course, which builds off your original simulation. But the crucial observation here is that we already know the transform of the impulse response - it is the same as the transfer function!

$$H = \frac{1}{-\omega^2 + 0.1\sqrt{-1}\omega + 1}.$$

This is the calculation you will make, and you have to use the oddly-ordered frequency vector that is defined in (a) in order to make it work properly.

In summary for this step then, point-wise multiply the vector U times the vector H, then take the inverse FFT to get a third calculation for the response x when the system is driven by u. The IFFT result plotted versus `treg` should come out to be very close to the two results computed before. (Note you will get a little complaint from the plotting command, which doesn't know what to do with the very small imaginary parts of the IFFT result - ignore it.)

8. Why didn't we have to worry about the N-scaling of the FFT when we did the IFFT?

The overlay plot for the three methods is given in the plot below; code is attached. In the last question, we don't have to worry about the N-scaling in the IFFT because $H(j\omega)$ is not computed with the FFT and so doesn't have the factor of N. Thus the IFFT "undoes" the N-scaling introduced by the FFT of U, in the same way that `ifft(fft(x)) = x`.

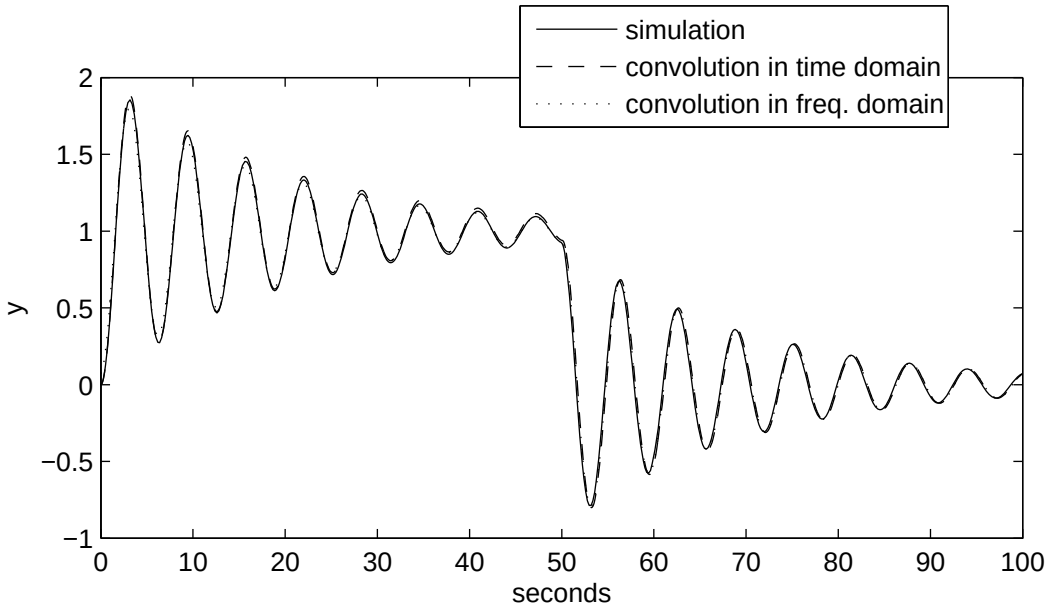


Figure 3: Overlay of system response by three different methods.

```
%-------------------------------------------------------------------------------
% Show the equivalence of time-domain simulation, convolution, and
% frequency-analysis of an LTI system.

% Franz Hover MIT Mechanical Engineering

clear all; close all;
global m b k epsilon T ;

m = 1 ;          % mass of oscillator
b = .1 ;         % damping
k = 1 ;          % stiffness

epsilon = .1 ;  % time duration used for delta function
dt = .1 ;        % uniform time step
T = 100 ;   % final time; make this an even multiple of dt !!!
            % Also, if T is big, we can minimize finite length effects

treg = 0:dt:T ;    % regular time vector
N = length(treg) ;
ureg = [ones(1,(N+1)/2),zeros(1,(N-1)/2)] ;
    % regular input vector (half ones, half zeros)
wreg = [0:(N-1)/2 , (1-N)/2:-1] / (N-1) * 2*pi/dt ;
    % regular frequency vector

% first, get the time domain impulse response
[th,h] = ode45('tripleMethod_dxdt1',[0 T],[0 0]);
h = h(:,2) ; % pick out just the second column - it goes with state x
[hreg] = spline(th,h,treg) ; % regularize h to align with treg

% 1.  make the response to u using the simulation directly
[t1,y1] = ode45('tripleMethod_dxdt2',[0 T],[0 0]);
y1 = y1(:,2) ;

% 2.  make the response using convolution in time domain
y2 = conv(ureg,hreg) * dt ;

% 3.  make the response using the FFT - convolution in freq. domain
U = fft(ureg) ;
H = ones(size(wreg))./(-m*wreg.^2 + sqrt(-1)*wreg*b + k) ;
Y = U.*H ;
y3 = ifft(Y) ;
```

```
% ... and compare!
figure(1);clf;hold off;
subplot('Position',[.2 .2 .6 .4]);
plot(t1,y1,treg,y2(1:N),'--',treg,(y3),':') ;
legend('simulation','convolution in time domain','convolution in freq. domain');
xlabel('seconds');
ylabel('y');

print -depsc tripleMethod.eps

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%---------------------------------------
function [ydot] = tripleMethod_dxdt1(t,y);
global m b k epsilon ;
if t < epsilon,
    u = 1/epsilon ;
else,
    u = 0 ;
end;
ydot(1,1) = -b/m*y(1) - k/m*y(2) + u ;
ydot(2,1) = y(1) ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%----------------------------------------
function [ydot] = d2_1_dxdt2(t,y);
global m b k T ;
if t < T/2,
    u = 1 ;
else,
    u = 0 ;
end;
ydot(1,1) = -b/m*y(1) - k/m*y(2) + u ;
ydot(2,1) = y(1) ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

18 Deck Flooding Calculation with Short-Term Statistics

Consider a fixed platform subject to storm waves. The waves we consider are those of Sea State 5, for which the modal period is about 9.7 seconds and the significant wave height is about 3.3 meters. Assume that the waves are not affected by the presence of the structure; this is valid for fixed structures which have very small waterplane area. Use the Bretschneider spectrum in what follows.

1. How high must the deck be to suffer a submergence only once every ten minutes? Once per hour? Once per day?

   I get wave elevations of [2.4 2.8 and 3.5] meters. The formulas are:

$$\begin{aligned} M_i &= \int_0^\infty \omega^i S(\omega) d\omega \\ \eta(T) &= \sqrt{-2M_0 \log\left(\frac{2\pi}{T w_{modal}}\right)} \end{aligned}$$

2. What is the average lowest value of the 1/100'th highest waves? Of the 1000'th highest waves? Of the 1/10000'th highest waves?

   I get [2.5, 3.0, 3.5] meters; note these are elevations also. The formulas are:

$$\begin{aligned} \epsilon &= \sqrt{1 - \frac{M_2^2}{M_0 M_4}} \\ z &= \sqrt{1-\epsilon^2} \\ \eta(N) &= \sqrt{2M_o \log\left(\frac{2zN}{1+z}\right)} \end{aligned}$$

3. Comment on how these two ways of expressing statistics are consistent or not.

   These two measures are consistent in two respects. First, the time scales are similar - i.e., there are about 8900 such waves in one day. Second, the average lowest value among the 1/N'th highest is intuitively very close to the highest wave seen in N observations. Note that this average lowest value among the 1/N'th highest waves, the 1/N'th maximum, is in contrast with the average 1/N'th highest value, which is quite a bit larger since it includes waves of arbitrarily large amplitude.

19 Aliasing

Consider sampling the sinusoidal process: $x(t) = 0.9\sin(2\pi f_o t)$.

1. When we sample the signal at frequency f_s (Hz) or ω_s (rad/s), the Nyquist rate of $f_N = f_s/2$ is the maximum frequency the sampling can detect. See the figure for some examples; undersampling of 1.7 indicates that the process frequency exceeds the Nyquist rate by a factor of 1.7. What is the required sampling rate to detect $f_o =$ 20kHz? Extrapolating from the figure, is this sampling rate good enough to observe the process with reasonable fidelity?

   The required rate is twice the process rate, or 40kHz. This rate does not give good phase or amplitude information of the 20kHz signal; about 200-400kHz would be much better, giving five to ten points per cycle.

2. Suppose that you can only sample at $f_s = 5.5kHz$. What is the time step? For the time interval $t = [0\ 0.002]$, plot $x(t)$ sampled at a high enough rate that you can clearly see the sine wave, and overlay it with the signal sampled at f_s. What is the apparent (*aliased*) frequency of the sampled signal?

   A simple rule to predict this aliased frequency is: decrement f_o by f_s enough times to get within the observable frequency range of $[-f_N,\ f_N]$. The absolute value of this result is the aliased frequency.

   Sampling at 5.5kHz gives a time step of 0.182 milliseconds. From the top subplot in Figure 4, this gives 2kHz aliased. Applying the formula above, we find $20kHz - 4 \times 5.5kHz = -2kHz$.

3. Repeat the above analysis for $f_s = 6.5kHz$.

   Sampling at 6.5kHz gives a time step of 0.154 milliseconds. From the middle subplot and the calculation, we get 0.5kHz.

4. Repeat the above analysis for $f_s = 7.5kHz$.

   Sampling at 7.5kHz gives a time step of 0.133 milliseconds. From the bottom subplot and the calculation, this gives 2.5kHz aliased.

```
%-----------------------------------------------------------------------
% Aliasing - explore several different undersampling sins.

% Franz Hover MIT Mechanical Engineering

clear all;

tfinal = .002 ;
```

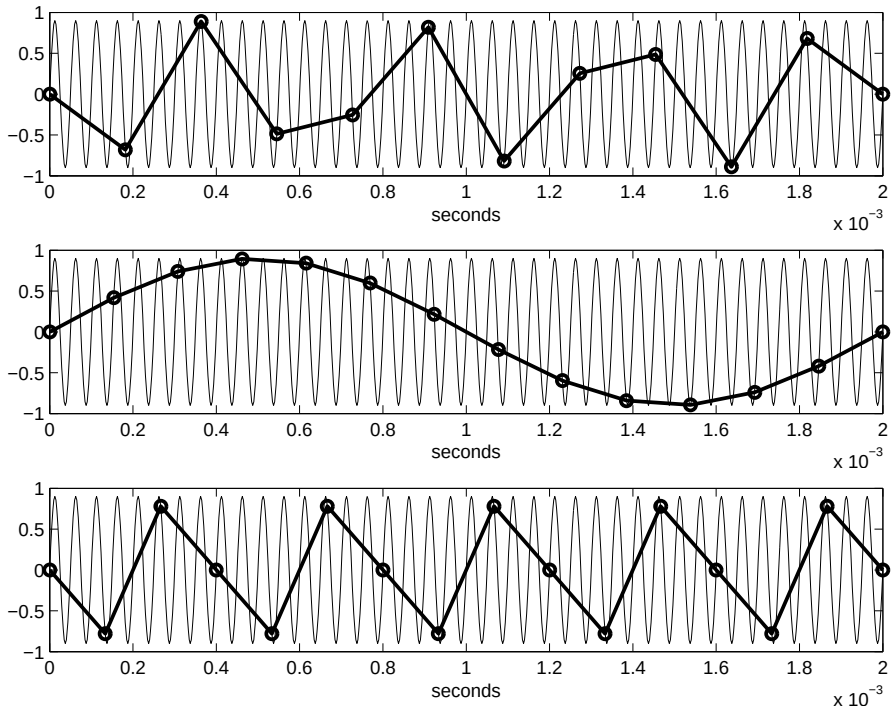


Figure 4: Aliasing: sampling a 20kHz signal at 5.5, 6.5, and 7.5kHz. A dangerous business!

```
w0 = 20000*2*pi;
ws = 7500*2*pi; % change the sampling rate here [5500 6500 7500]

dtfi = 2*pi/w0/20 ; % fine time scale
tfi = 0:dtfi:tfinal ;

dt = 2*pi/ws ; % sampling time scale
t = 0:dt:tfinal ;

xfi = .9*sin(w0*tfi) ; % here are the signals
x = .9*sin(w0*t) ;

figure(1);clf;hold off;
subplot(3,1,1);
plot(tfi,xfi);
hold on;
plot(t,x,'ro-','LineWidth',2);
xlabel('seconds');
disp('Save the figure if necessary.');
```

```
nfi = length(xfi);
fxfi = fft(xfi)/nfi;
wfi = [0:nfi-1]/(nfi-1)*2*pi/dtfi ;

n = length(x);
fx = fft(x)/n ;
w = [0:n-1]/(n-1)*2*pi/dt;

figure(2);clf;hold off;
plot(w/2/pi,abs(fx),wfi/2/pi,abs(fxfi));

hold on;
plot(mod(w0,ws)/2/pi,0,'r*','LineWidth',2);
plot(mod(-w0,ws)/2/pi,0,'r*','LineWidth',2);

disp(sprintf('Sampling freq:         %g Hz.', ws/2/pi));
disp(sprintf('Aliased frequencies:               %g  %g Hz.', ...
    mod(w0,ws)/2/pi,mod(-w0,ws)/2/pi));

disp('Save figure if you want.');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

20 Computations on Recorded RP Data

Load the data file `homework4.dat` from the course site. The first column is time t and the second column is a measured signal $h(t)$, in Volts.

1. What is the variance of h, from the simple time formula? What is the significant (average 1/3-highest) height based on this variance?

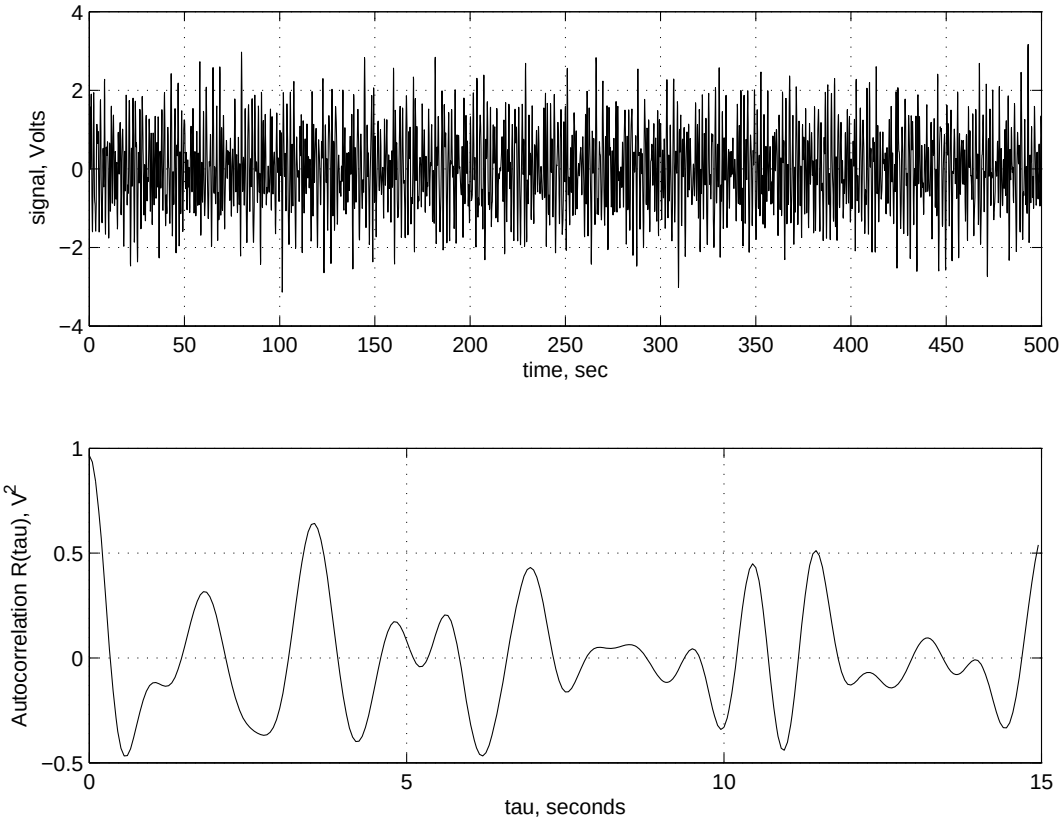
   The variance is the time average of $h(t)^2$, because the signal has zero mean; it comes out to $0.97V^2$, so that the significant height is $3.93V$.

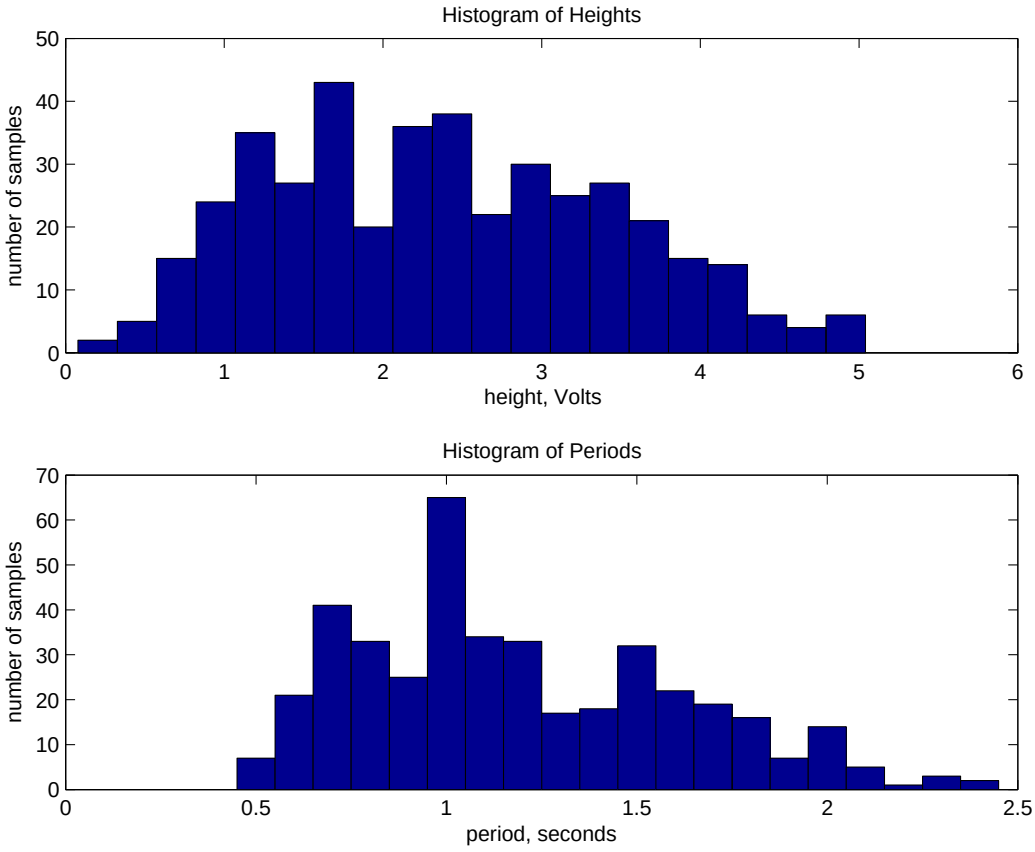
2. Write a program to count the waves using zero upcrossings as the endpoints for cycles. How many full waves do you get? What is the significant height from these observations? What is the average period? Make histograms (20 bins please) of the heights and of the periods observed.

   There are 415 complete waves, the one-third average highest height is $3.63V$, and the average period is $1.20sec$. The height histogram confirms a Rayleigh-like distribution (although 415 data are not really enough to be conclusive). Note the significant height calculated here is a little low because this is not a very narrow-band process. You can see this by looking at the signal close-up: lots of false peaks.

3. Calculate and plot the autocorrelation function $R(\tau)$ for $\tau = [0, 15]sec$.

   See the plot. Notice the value at $\tau = 0s$ is $0.97V^2$, which is of course the variance calculated another way, $\frac{1}{2}\sum a_i^2$, where the amplitudes a_i are given in the program.


signal, Volts
time, sec
Autocorrelation R(tau), V2
tau, seconds


Histogram of Heights
number of samples
height, Volts
0
10
20
30
40
50
0
1
2
3
4
5
6
Histogram of Periods
number of samples
period, seconds
0
10
20
30
40
50
60
70
0
0.5
1
1.5
2
2.5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analyze time-domain data
% MIT 2.017 FSH October 2009

clear all ;

dt = .05 ; % time step
t = 0:dt:10000*dt; % time vector

%w = [1 2 3 4 5 6 7 8 9 10] ;
%w = w + randn(size(w))*.3 ;
w = [1.1871    1.6613    3.4060    3.7160    5.4306    6.6176    ...
    7.1664    7.7864    8.9320    9.7945] ; % slightly perturbed freqs

a = [.1 .4 .6 .7 .6 .5 .4 .3 .2 .1] ; % amplitudes

%phi = 2*pi*rand(1,length(w));
phi = [6.0486    6.2545    5.8839    4.8933    1.1329    5.9219    ...
    6.0299    1.3844    4.9657    4.5073] ; % random phases

VPure = sum(a.^2)/2 ; % area under the curve = variance
HsigPure = 4*sqrt(VPure);

% build up the time-domain signal
n = length(t) ;
h = zeros(1,n) ;
for i = 1:length(w),
    h = h + a(i)*sin(w(i)*t + phi(i)) ;
end;

figure(1);clf;hold off;
subplot(211);
plot(t,h) ;
xlabel('time, sec');
ylabel('signal, Volts');
grid;

Vt = 1/n*sum(h.*h) ; % the time average
Hsigt = 4*sqrt(Vt) ;

% calculate the heights from zero up-crossings
[ind] = find( (h(1:end-1) < 0) & (h(2:end) > 0)) ;
for i = 1:length(ind)-1,
    height(i) = max(h(ind(i):ind(i+1))) - min(h(ind(i):ind(i+1)));
```

```
end;

figure(2);clf;hold off;
subplot(211);
hist(height,20);
title('Histogram of Heights');
xlabel('height, Volts');
ylabel('number of samples');

heightSort = -sort(-height) ;
nHeights = length(height) ;
disp(sprintf('Calculated %d heights.', nHeights));
topThirdHeights = heightSort(1:round(nHeights/3)) ;
avgOneThirdHeight = mean(topThirdHeights);

periods = diff(ind)*dt ;
meanPeriod = mean(periods);

figure(2);subplot(212);
hist(periods,20);
title('Histogram of Periods');
xlabel('period, seconds');
ylabel('number of samples');

disp(sprintf('Significant heights:  %g %g %g V.', HsigPure, Hsigt,...
    avgOneThirdHeight));
disp(sprintf('Mean period is %g seconds.', meanPeriod));

% calculate the autocorrelation function
for i = 1:n,
    R(i) = 1/(n-i+1)*sum( h(1:(n+1-i)).*h(i:n)) ;
end;

figure(1);subplot(212);
plot(t(1:300),R(1:300)) ;
xlabel('tau, seconds');
ylabel('Autocorrelation R(tau), V^2');
grid;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

21 Hurricane Winds

A 1959 paper by Isaac Van der Hoven gives the spectrum of wind speeds during Hurricane Connie, measured on a tower at Brookhaven National Laboratory. His curve for $S^+(\omega)$ is approximated by the points below:

Frequency, cycles/hr	$S^+(\omega)$ m^2/s
0	0.00
10	0.50
14	0.65
20	1.00
32	2.80
50	3.10
72	2.80
100	2.00
141	1.60
200	1.20
316	0.80
500	0.60
717	0.50
1000	0.40
1410	0.20
2000	0.00

This one-sided spectrum is given in units of m^2/s, i.e., velocity squared divided by ω (rad/s), so that the area under it is equal to the variance. The mean wind speed during most of the hurricane was $13m/s$, but for one hour at the peak it was $20m/s$.

1. Make a plot of this spectrum data - $S^+(\omega)$ vs. ω (rad/s).

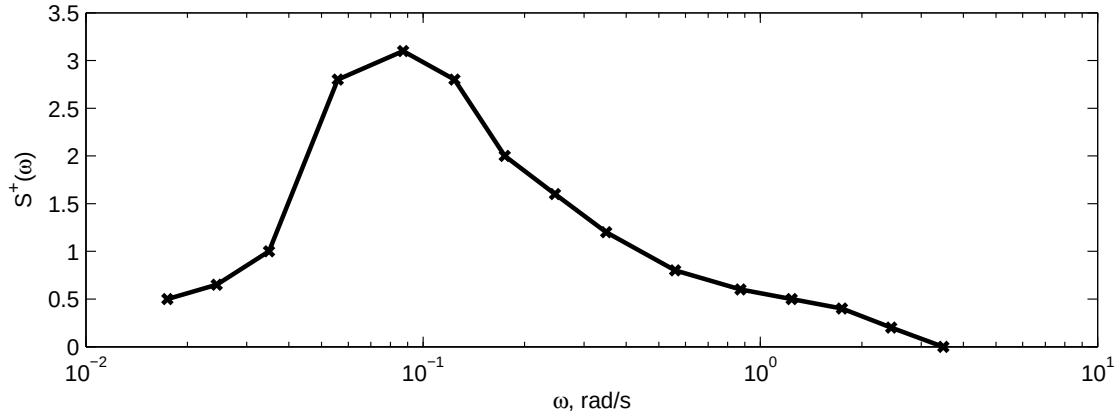
2. What is the characteristic frequency of the windspeed fluctuations? What is the approximate standard deviation of wind velocity, and the significant amplitude $\bar{a}^{1/3}$?

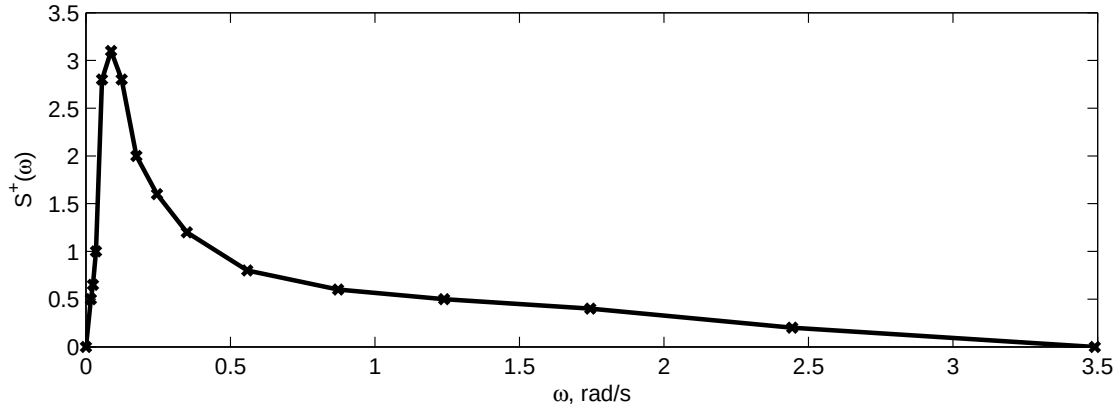
   Solution: The peak frequency is apparently at about fifty cycles per hour, or one cycle per 72 seconds. To get σ and $\bar{a}^{1/3} = 2\sigma$, we have to get the area under the spectrum. The attached code shows how to do this - see also the worked example on the Bretschneider spectrum. The standard deviation here is $1.35m/s$, leading to a significant amplitude of $2.7m/s$. This is a fluctuation of plus or minus 15-20% from the mean speeds during the hurricane.

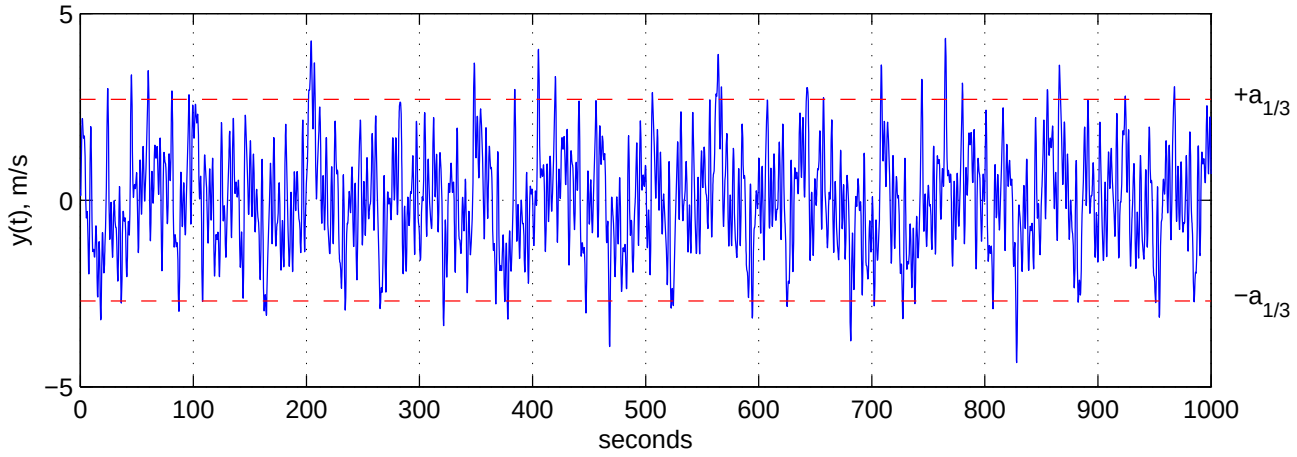
3. Generate a sample trace of time-domain data, with a time step of 0.1 seconds, and a duration of one thousand seconds. Note that for each frequency bin of width $\delta\omega$, we have $a_i^2/2 = S^+(\omega_i)\delta\omega$. This gives you the amplitudes for each center frequency

you use; impose a fixed random phase angle for each component, add the components together, and you are done.

Plot plus and minus $\bar{a}^{1/3}$ on top of your trace, and label.







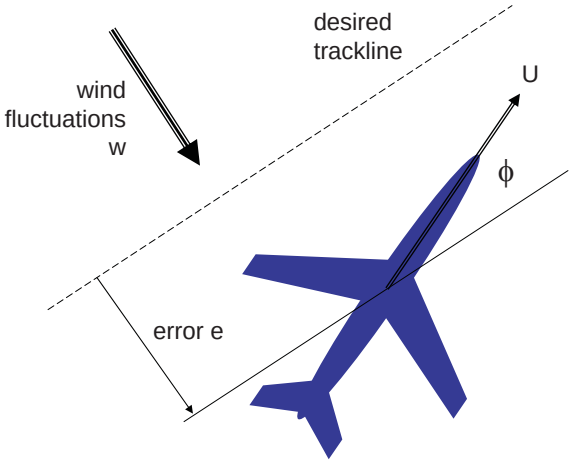
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explore the wind spectrum for Hurricane Connie, after
% van der Hoven 1959.

clear all;

cph = [0 10 14 20 32 50 71 100 141 200 ...
        320 500 710 1000 1400 2000] ; % freq., in cycles per hour
S = [0 .5  .65 1, 2.8 3.1, 2.8  2 ...
    1.6 1.2 .8 .6 .5 .4 .2 0] ; % spectrum to go with cph frequencies

w = cph*2*pi/3600 ; % freq., radians/second

figure(1);clf;hold off;
subplot(212);
plot(w,S,'x-','LineWidth',2) ;
xlabel('\omega, rad/s');
ylabel('S^+(\omega)');
subplot(211);
semilogx(w,S,'x-','LineWidth',2);
xlabel('\omega, rad/s');
ylabel('S^+(\omega)');
print -deps hurricaneWindSpectrum1.eps

widths = ([0 diff(w)] + [diff(w) 0])/2 ; % make the strip widths
var = sum(S.*widths) ; % the variance

stddev = sqrt(var);
asig = 2*stddev ;
disp(sprintf('The stddev is %g m/s and the sig. amp. is %g m/s',...
    stddev,asig));

% compute the amplitudes that go with each frequency, and pick
% some random phase angles, uniformly distributed in [0,2*pi]
for i = 1:length(widths),
    a(i) = sqrt(2*S(i)*widths(i)) ;
    ph(i) = rand*2*pi ;
end;

dt = .1 ; % time step

% a typical two-loop construction to generate the time series
t0 = clock ;
for j = 1:10001, % loop through the times
```

```
    z(j) = 0 ;
    t(j) = (j-1)*dt ; % time
    for i = 1:length(widths), % add up the components
        z(j) = z(j) + a(i)*cos(w(i)*t(j) + ph(i)) ;
    end;
end;
disp(sprintf('The two-loops took %g seconds.', ...
    etime(clock,t0)));

% NOTE:  here is a better way to do the above double loop.  It is
% vectorized and will run much faster (about 100x here) !
t0 = clock ;
j = 1:10001 ;
z = zeros(size(j));
t = dt*(j-1) ;
for i = 1:length(widths),
    z = z + a(i)*cos(w(i)*t + ph(i)) ;
end;
disp(sprintf('The vectorized version took %g seconds.', ...
    etime(clock,t0)));

figure(2);clf;hold off;
subplot(211);
plot(t,z) ;
grid;
hold on;
plot([min(t) max(t)], asig*[1 1],'r--');
plot([min(t) max(t)], -asig*[1 1],'r--');
text(1020,asig,'+a_{1/3}');
text(1020,-asig,'-a_{1/3}');
xlabel('seconds');
ylabel('y(t), m/s');
print -depsc hurricaneWindSpectrum2.eps ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

22 Aircraft in Winds

This problem builds on the previous one: you will use the same wind gust spectrum, and study the response of an air vehicle that is being buffeted by it. Here is the spectrum of the wind again, along with a picture of the aircraft:

Frequency, cycles/hr	$S_w^+(\omega)$ m^2/s
0	0.00
10	0.50
14	0.65
20	1.00
32	2.80
50	3.10
72	2.80
100	2.00
141	1.60
200	1.20
316	0.80
500	0.60
717	0.50
1000	0.40
1410	0.20
2000	0.00



The aircraft has a forward speed of $U = 60m/s$, or about 120 knots. It employs GPS navigation and a simple feedback loop to stay on a given straight-line path. In particular, the feedback law is $\phi = 0.003 \times e$, where ϕ is the controlled heading of the vehicle relative to the desired line of travel, and e is the cross-track error, i.e., the position of the aircraft in the direction normal to the desired line of travel. Thus, a cross-track error of ten meters leads to a heading command of 0.03 radians, which points the vehicle back toward the track. We assume that wind gusts do not affect the heading of the craft directly, but that there is a very good heading controller, that will make the vehicle follow the desired ϕ very closely.

1. Making the linearizing assumption that ϕ is small, and including the feedback law, what is the differential equation relating wind gust velocity $w(t)$ to cross-track error $e(t)$? What is the transfer function $E(j\omega)/W(j\omega)$? Note that the aircraft has negligible mass. *Hint: it is a low-pass filter.*

   Solution: The linearization is $\dot{e} = -U\phi + w$ or $\dot{e} = -60 \times 0.003 \times e + w$. Note we have not allowed for any sideslip in the the vehicle - that is, it travels exactly in the direction it is pointing. The transfer function taking the disturbance to the error is $F(j\omega) = E(j\omega)/W(j\omega) = 1/(j\omega + 0.18)$.

2. Make a plot of one thousand seconds of $w(t)$ just as in Homework 4, and show the corresponding system output $e(t)$, in meters. For this, you need to solve the differential equation numerically.

   See attached code and the first two plots.

3. Using the Wiener-Khinchine relation, make a plot of $S_e(\omega)$. Then calculate the significant error $\bar{e}_{1/3}$, and plot the plus and minus values of this on your time domain plot above.

   The W-K relation says that

   $$S_e(\omega) = |F(j\omega)|^2 S_w(\omega),$$

   so all you need is a pointwise multiplication in frequency space of the input spectrum $S_w(\omega)$ by the squared magnitude of the transfer function, $|F(j\omega)|^2$. See the code and spectrum plot. The significant value of the error amplitude $\bar{e}_{1/3}$ is 7.44 meters.

4. What happens to $\bar{e}_{1/3}$ as you increase or decrease the feedback gain from 0.003, say to zero and to 0.006?

   Setting the heading gain to zero means there is no corrective action, and the cross-track error just follows the integral of the disturbance (in the manner of a random walk). Increasing the gain to 0.006 makes the control action stronger; the error is reduced and the frequency content of the error seems higher. In fact, the higher gain is reducing more of the low-frequency part of the error. See the last two sets of plots.

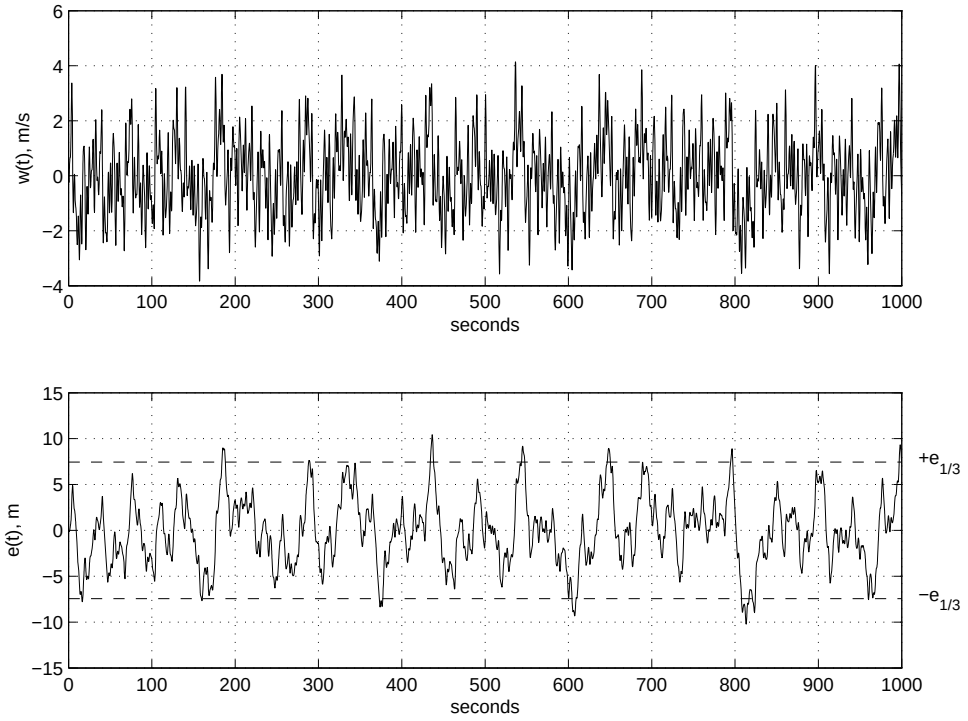


Figure 5: Time series of wind input and error output, for gain of 0.003.

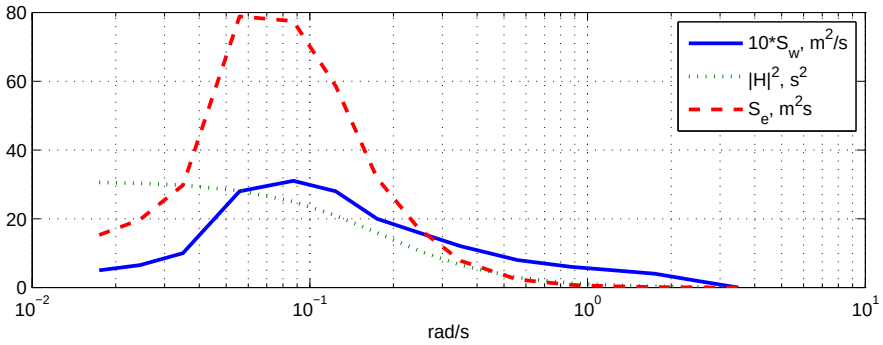


Figure 6: Input and output spectra, with gain 0.003.

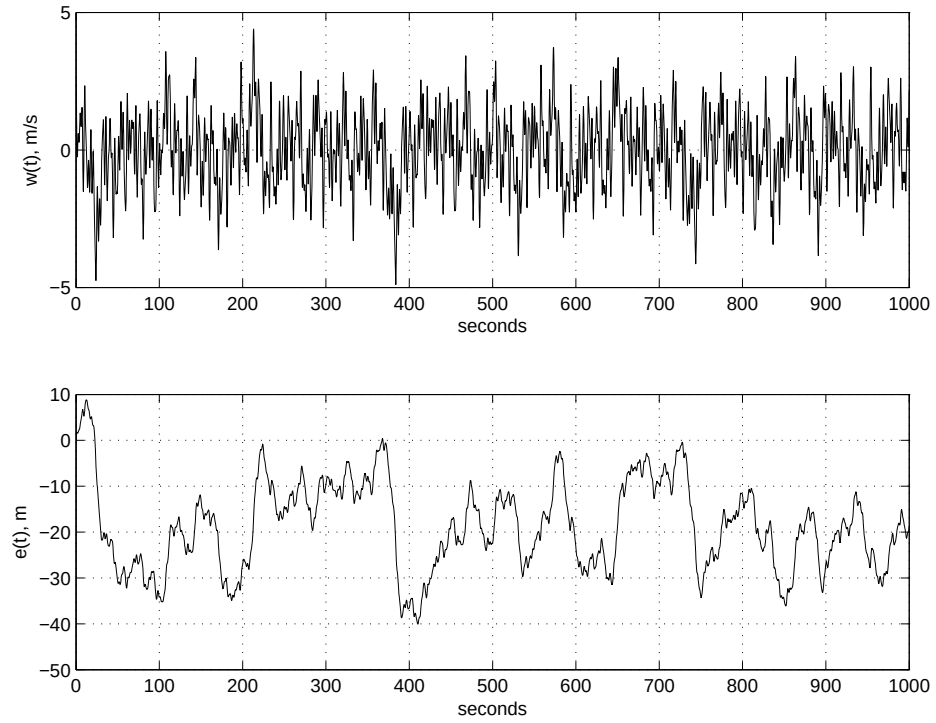


Figure 7: Time series of wind input and error output, for gain of zero - the wind is simply integrated. There is no significant amplitude because $|F|$ is unbounded at low frequencies, and hence the output spectrum is also unbounded here.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aircraft response under simple trackline control

function aircraft

clear all;

U = 60 ; % aircraft speed
k = 0.003 ; % heading gain, rad/m

cph = [0 10 14 20 32 50 71 100 141 200 ...
        320 500 710 1000 1400 2000] ; % freq., in cycles per hour
Sw = [0 .5  .65 1, 2.8 3.1, 2.8  2 ...
    1.6 1.2 .8 .6 .5 .4 .2 0] ; % spectrum to go with cph frequencies

omega = cph*2*pi/3600 ; % freq., radians/second

widths = ([0 diff(omega)] + [diff(omega) 0])/2 ; % make the strip widths
```

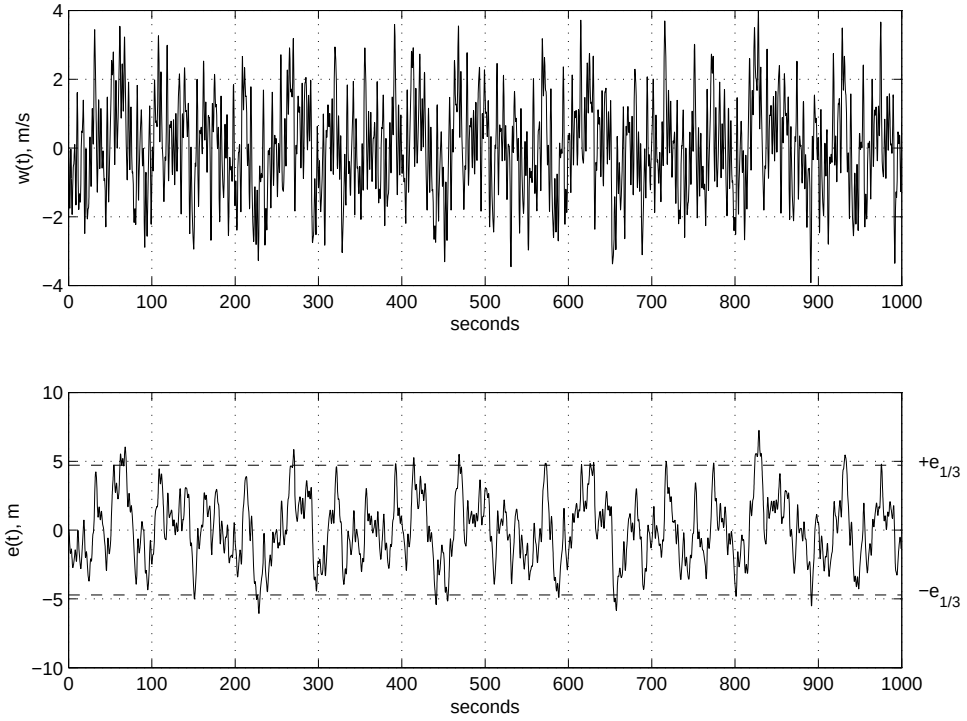


Figure 8: Time series of wind input and error output, for gain of 0.006.

```
% compute the amplitudes that go with each frequency, and pick
% some random phase angles, uniformly distributed in [0,2*pi]
for i = 1:length(widths),
    a(i) = sqrt(2*Sw(i)*widths(i)) ;
    ph(i) = rand*2*pi ;
end;

dt = .1 ; % time step

j = 1:10001 ;
w = zeros(size(j));
t = dt*(j-1) ;
for i = 1:length(widths),
    w = w + a(i)*cos(omega(i)*t + ph(i)) ;
end;

figure(1);clf;hold off;
subplot(211);
plot(t,w) ;
grid on; xlabel('seconds'); ylabel('w(t), m/s');
```

```
e(1) = 0 ;
for i = 2:length(t),
    k1 = dt*f( t(i-1), e(i-1), w(i-1), U, k);
    k2 = dt*f( t(i-1)+dt/2, e(i-1)+k1/2, (w(i-1)+w(i))/2, U, k );
    e(i) = e(i-1) + k2 ;
end;

figure(1);
subplot(212);
plot(t,e);
grid on; xlabel('seconds'); ylabel('e(t), m');

for i = 1:length(omega),
    H(i) = 1 / (sqrt(-1)*omega(i) + U*k) ;
    H2(i) = H(i)*conj(H(i)) ;
end;

Se = H2 .* Sw ;

evar = sum(Se.*widths);
esig = 2*sqrt(evar);
disp(sprintf('Significant value of e:  %g m', esig));

figure(1);
subplot(212); hold on;
plot([min(t) max(t)], esig*[1 1],'r--');
plot([min(t) max(t)], -esig*[1 1],'r--');
text(1020,esig,'+e_{1/3}');
text(1020,-esig,'-e_{1/3}');

figure(2);clf;hold off;
subplot(211);
semilogx(omega, 10*Sw, omega, H2,':', omega, Se,'--', 'LineWidth',2) ;
legend('10*S_w, m^2/s', '|H|^2, s^2','S_e, m^2s');
xlabel('rad/s');
grid;

function [edot] = f(t, e, w, U, k)
    edot = -U*k*e + w ;
return ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

23 Identification of a Response Amplitude Operator from Data

Load the data file `homework5.dat` from the course site. The first column is time t in seconds, the second column is a measured input signal $u(t)$, in Volts, and the third column is a measured output signal $y(t)$, also in Volts. $u(t)$ has no significant frequency content above $2rad/s$.

Your main task is to estimate the Response Amplitude Operator (RAO) $H(\omega)$ and the underlying transfer function $G(j\omega)$ of the system that created $y(t)$ when driven by $u(t)$. Your deliverables are:

1. A plot of the data-based RAO $H(\omega)$ versus frequency.
2. The specific transfer function that best describes the data, e.g., $G(j\omega) = 6.5/(j\omega+2.2)$. Recall that $H(\omega) = |G(j\omega)|^2$.
3. A plot of the step response of $G(j\omega)$, making the assumption that the system is causal.

Some hints:

- The recommended overall procedure is to calculate the autocorrelation function for each signal, compute the Fourier transforms of these, and then divide according to the Wiener-Khinchine relation so as to recover an empirical $H(\omega) = |G(j\omega)|^2$.
- A detailed discussion on use of the FFT in MATLAB is given in the worked problem entitled *Dynamics Calculations Using the Time and Frequency Domains*. This includes specification of the frequency vector that goes with an FFT, which is very important for the current problem.
- The FFT of an autocorrelation function is supposed to be all real because $R(\tau)$ has only the cosine phases represented. Yet look at the signal `fft(cos(t));` and you will see that it not only has some imaginary parts, but also a lot of content NOT at frequency one. Some of this appears because numerical Fourier transforms assume that the time-domain signal is periodic - that the end reconnects with the beginning. This often gives a discontinuity, which the transform tries to take care of with the extra stuff you see. Some of this messiness also comes up because the FFT is given only at discrete frequencies. When your signal actually has other frequencies, the FFT spreads things out a bit.

  The first problem can be reduced by multiplying your time-domain signal by a cosine window such as $1/2 * (1 - cos(2 * pi * [0 : n - 1]/(n - 1)))$. When you plot it out, you can see directly that there won't be any discontinuity between the beginning and end. The second problem above doesn't affect the answers you get on this homework; I assure you that the RAO is pretty smooth.

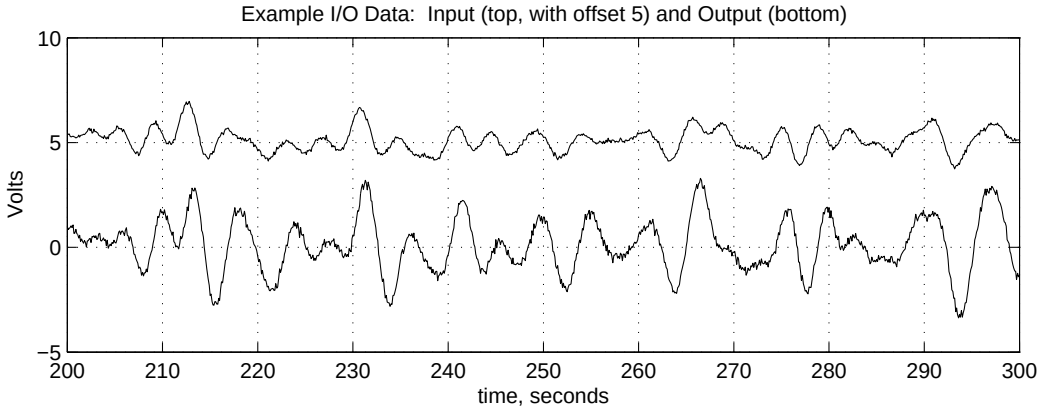
- Super-duper hint: The true transfer function has either no zeros or one zero (which could be in the right-half plane), and one or two poles, both in the left-half plane (since the system is clearly stable). It will take a little effort to get a good fit with your data, because you are searching in four parameters. But think about the damping ratio and undamped frequency and you will already be in the right ballpark. You might even like to automate the fit procedure by writing it as an optimization problem!

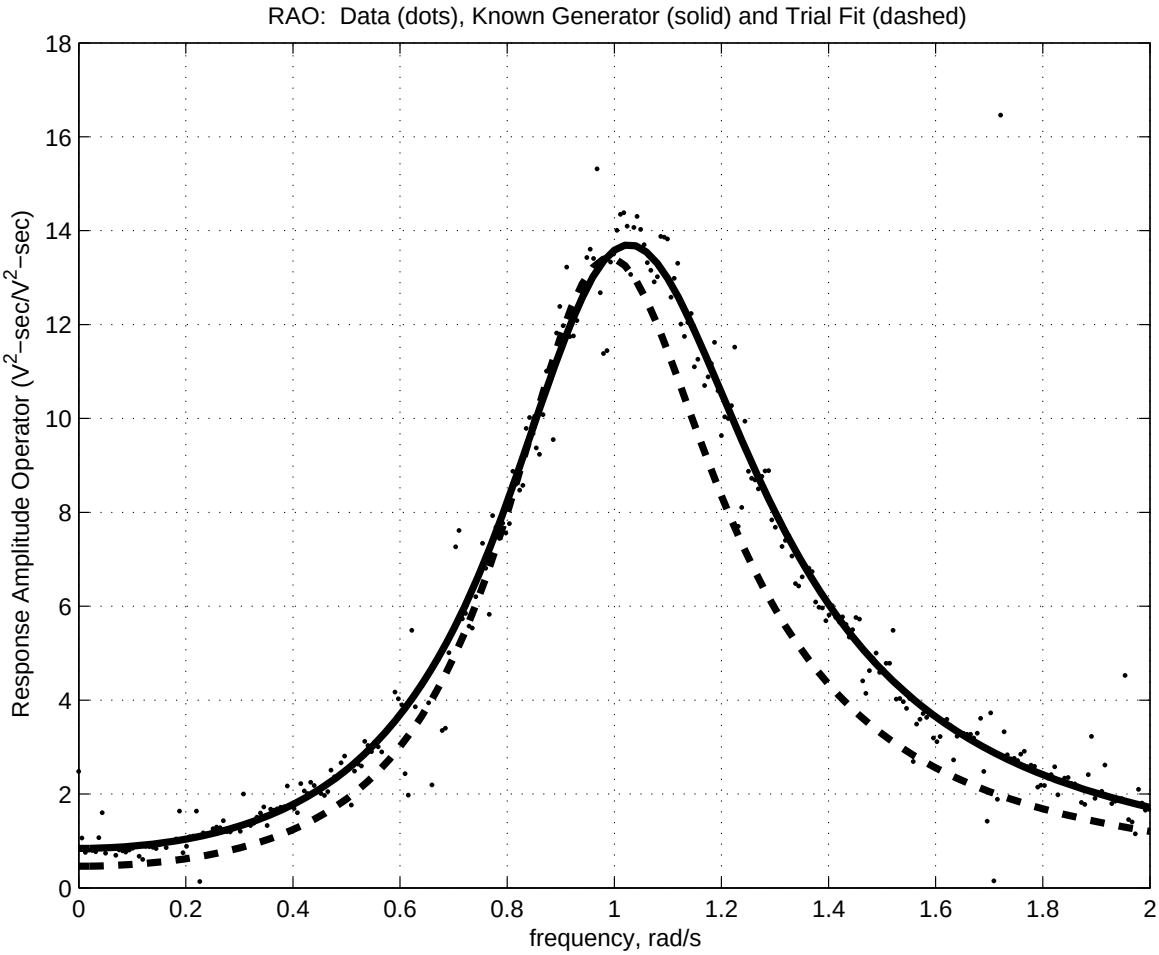
Solutions

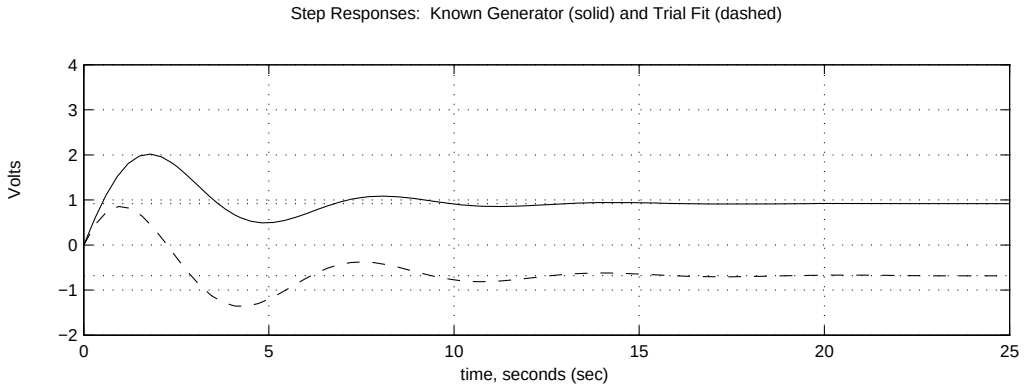
1. See the attached code and figures. The input is quite broad-band: in fact, the frequency content is uniform from zero to two radians per second, and zero above (except for the added noise). There is a resonance effect going on at around a six-second period, with fairly light damping. The empirical RAO is the *dots* in the second figure; the lines are the known and trial fits of specific transfer functions. The low-frequency RAO is about one, and at high frequencies it goes to zero (although the plot given doesn't go to high enough frequencies to see this). I note that because the input has no frequency content above $2 rad/s$, the sensor noise will muddy everything up above this frequency, and the RAO is basically undefined in this range.

2. The generating transfer function is $G(j\omega) = (2j\omega + 1)/(-\omega^2 + 0.6j\omega + 1.09)$, and it is causal so you can replace the $j\omega$'s with s's if you like. This has a left-half-plane zero at $s = -0.5$, and (of course) two lightly-damped poles near $s = \pm j$. The magnitudes of this transfer function fit the RAO from data quite well. A second fit is plotted also, $G_2(j\omega) = (1.7j\omega - 0.68)/(-\omega^2 + 0.5j\omega + 1)$, which is based on the trying to match the natural frequency and damping ratio first, and then looking at the magnitude and a possible zero.

3. The step response of the system is a trick question. The RAO only tells us the magnitude information of the system, and nothing about the phase. Notice the sign change in the numerator of G_2; it fits the RAO just fine, but is a somewhat different system when you look at the step response!


Example I/O Data: Input (top, with offset 5) and Output (bottom)
Volts
time, seconds
10
5
0
−5
200
210
220
230
240
250
260
270
280
290
300


RAO: Data (dots), Known Generator (solid) and Trial Fit (dashed)
Response Amplitude Operator (V2−sec/V2−sec)
frequency, rad/s
0
2
4
6
8
10
12
14
16
18
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Step Responses: Known Generator (solid) and Trial Fit (dashed)
Volts
time, seconds (sec)
4
3
2
1
0
−1
−2
0
5
10
15
20
25

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Extract an RAO from I/O data
% MIT 2.017 FSH October 2009

clear all ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This first part of code generates the noisy i/o data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dt = .1 ; % time step, sec
t = 0:dt:10000*dt; % time vector

%dw = .02 ; % frequency increment, rad/s
dw = .04 ; % frequency increment, rad/s

w = dw:dw:100*dw ; % frequency vector
w = w + randn(size(w))*dw/5 ; % randomize a little to prevent cycling

a = .08*ones(size(w)) ; % amplitudes (uniform)

phi = 2*pi*rand(1,length(w)); % random phase angles

%uNoise = .1 ; % input noise level
%yNoise = .1 ; % output noise level
uNoise = .001 ; % input noise level
yNoise = .001 ; % output noise level

% build up the input signal
n = length(t) ;
u = zeros(1,n) ;
for i = 1:length(w),
    u = u + a(i)*sin(w(i)*t + phi(i)) ;
end;
u = u + randn(1,n).*std(u)*uNoise ; % add noise to the input

% to make the output, define an LTI system, and run input through it
%sys = zpk(-0.5,[-.3+sqrt(-1), -.3-sqrt(-1)],2);
sys = zpk([],[-1,0],2);
y = lsim(sys,u,t) ;
y = y' ;
y = y + randn(1,n).*std(y)*yNoise ; % add noise to output

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Here is where the student's work begins
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(1);clf;hold off;subplot(211);
plot(t,u+5,'k',t,y,'k') ;
a = axis ; axis([200 300 a(3:4)]); % show just a part
title('Example I/O Data:  Input (top, with offset 5) and Output (bottom)');
xlabel('time, seconds');
ylabel('Volts');
grid;

% calculate the autocorrelation functions
Ru = zeros(n,1);
Ry = zeros(n,1);
for i = 1:n,
    Ru(i) = 1/(n-i+1)*sum( u(1:(n+1-i)).*u(i:n)) ;
    Ry(i) = 1/(n-i+1)*sum( y(1:(n+1-i)).*y(i:n)) ;
end;

% multiply each R by a full cosine window to clean up the spectra
Ru = Ru * 1/2 .*(1-cos([0:1:n-1]/(n-1)*2*pi))';
Ry = Ry * 1/2 .*(1-cos([0:1:n-1]/(n-1)*2*pi))';

% here are the spectra
fRu = real(fft(Ru)) ; % (throw out imag part because Ru and
fRy = real(fft(Ry)) ; % Ry are known to only have cosine phase)

fw = [0:1:n-1]/n*2*pi/dt ; % frequency vector to go with fRu and fRy

% A key point here is that the scaling of the fft is the same for both
% fRy and fRu, so their division will give the correct results for the
% RAO

figure(2);clf;hold off;
%plot(fw,abs(fRy./fRu),'k.');
semilogy(fw,abs(fRy./fRu),'k.');
ylabel('Response Amplitude Operator (V^2-sec/V^2-sec)');
xlabel('frequency, rad/s');

% put the known |tf|^2 on top of the RAO data, for reference
%wb = [0:.02:2];
wb = 0:dw:max(w);
[magbdum,phaseb] = bode(sys,wb);
magb(1,1:length(wb)) = magbdum(1,1,:);
```

```
hold on;
%plot(wb,magb.^2,'k','LineWidth',3);
semilogy(wb,magb.^2,'k','LineWidth',3);
grid;

% put also a trial fit based mainly on damping ratio and natural frequency;
% also, we have to add a zero and get the gain right to make things work out
%zeta = .25 ;
%wn = 1 ;
zeta = 5 ;
wn = .1 ;
sys2 = zpk([],roots([1 2*zeta*wn wn^2]),2.2);
[magbdum2,phaseb2] = bode(sys2,wb);
magb2(1,1:length(wb)) = magbdum2(1,1,:);  % (transform a 3d to a 2d array)
%plot(wb,magb2.^2,'k--','LineWidth',3);
semilogy(wb,magb2.^2,'k--','LineWidth',3);
%a = axis ; axis([min(wb) max(wb) 0 18]); % don't show anything above wb
a = axis ; axis([min(wb) max(wb) 0 1e6]); % don't show anything above wb

title('RAO:  Data (dots), Known Generator (solid) and Trial Fit (dashed)');

figure(3);clf;hold off;subplot(211);
%step(sys,'k',30);
step(sys,'k',5);
hold on;
%step(sys2,'k--',30);
step(sys2,'k--',5);
grid;
xlabel('time, seconds');
ylabel('Volts');
title('Step Responses:  Known Generator (solid) and Trial Fit (dashed)');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

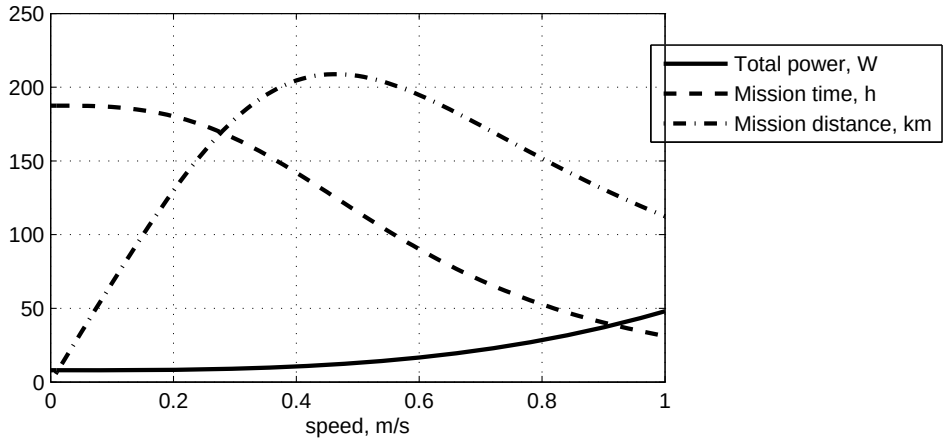
24 AUV Mission Optimization

Tradeoffs and optimization are demonstrated by the following simple example. An autonomous robot carries an energy source of $E = 1.5$ kiloWatt-hours. At operating speed U, the drive motors consume $P_d = 40U^3$ Watts. Additionally, no matter what the speed, the "hotel" load of the robot - the computers, sensors, communication equipment, etc. - consumes $P_h = 8$ Watts. What speed should the robot travel so as to achieve the greatest distance, how long does this mission last, and how far does the robot go in this case?

The drive power is $P_d = 40U^3$, and the hotel load is $P_h = 8$, so the total power is $P = 40U^3 + 8$, in Watts. The duration is the available energy divided by the total power, or

$$T = \frac{1500 \times 3600}{40U^3 + 8}$$

and the distance is simply $D = TU$. We see that D can be written then as a function of U, with a linear term in the numerator, and a cubic term plus a constant in the denominator. The solution can be obtained by either setting the $dD/dU = 0$, or looking at the function calculated numerically. The function is shown in the following plot, along with the optimum point. From the graph, the optimum speed is about 0.46m/s, the distance is 209km, and the duration is 127 hours.




```
%-----------------------------------------------------------------
% Electrical Power Budget on an AUV

clear all;

Uvec = 0:.01:1; % set up a vector of speeds, m/s
```

```
for i=1:length(Uvec),
    Pd(i) = 40*Uvec(i)^3 ; % propulsion power, W
    Ph(i) = 8 ;  % hotel power, W
    Pt(i) = Pd(i)+Ph(i) ; % total power, W

    T(i) = 1500 / Pt(i); % time of mission, h
    X(i) = T(i)*3600*Uvec(i) ; % distance traveled, m
end;

figure(1);clf;hold off;
subplot('Position',[.2 .2 .5 .4]);
plot(Uvec,Pt,'-',Uvec,T,'--',Uvec,X/1000,'-.','LineWidth',2) ;
legend('Total power, W', 'Mission time, h', 'Mission distance, km');
grid;
xlabel('speed, m/s');

%--------------------------------------------------------------------
```

25 Geometry Optimization

Give the formulas for the radius and length of a closed, half-cylindrical tank (i.e., having a "D"-like cross-section and flat ends) that will hold a given volume V. Your objective is to make the tank with minimum surface area. What are the specific dimensions if the volume is $100m^3$?

The volume and surface area of the tank are given respectively by:

$$\begin{aligned} V &= \frac{\pi}{2} r^2 l \\ A &= \pi r l + \pi r^2 + 2rl \end{aligned}$$

With the volume fixed, we seek the condition where the variation in A is zero, that is, $\partial A/\partial r = \partial A/\partial l = 0$, subject to the volume constraint. Following the example of Middendorf and Engelmann (1998), we work this out using Lagrange multipliers, even though substitution as in the first problem could also be used.

The area itself is the cost function, and we augment it as

$$\begin{aligned} A' &= A + \lambda(V - \pi r^2 l/2) \\ A' &= \pi r l + \pi r^2 + 2rl + \lambda(V - \pi r^2 l/2) \end{aligned}$$

Now we have to set the derivatives of A' with respect to r and l to zero:

$$\begin{aligned} \frac{\partial A'}{\partial r} = 0 &= \pi l + 2\pi r + 2l - \lambda \pi r l \\ \frac{\partial A'}{\partial l} = 0 &= \pi r + 2r - \lambda \pi r^2/2 \end{aligned}$$

These relations along with the constraint $V - \pi r^2 l/2 = 0$, form three equations in three unknowns. Use the second equation to write λ in terms of r, and then use the third to write l in terms of r, and then substitute both of these into the first equation, to end up with our explicit solution for radius:

$$r = \left[\frac{V(2+\pi)}{\pi^2}\right]^{1/3}$$

This is exactly what you get if you use substitution to solve the problem, and in that case, we would put this r back into the constraint equation to solve for l (a little messy). But the Lagrange technique gives us more insight than this: if we use the first and second of the three equations above, we can eliminate λ to find that $l = 2\pi r/(2+\pi)$. This is a new property of the optimum solution, and we have simply

$$l = \frac{2\pi}{2+\pi}\left[\frac{V(2+\pi)}{\pi^2}\right]^{1/3}$$

If $V = 100m^3$, we find $r = 3.735m$ and $l = 4.564m$.

26 Min-max Multi-Objective Optimization

Select the best candidate solution below [A-F], based on min-max optimization. This exercise presents some of the considerations you might encounter in the purchase of a large machine, such as an engine. The objectives are to minimize weight and cost, while maximizing the other attributes. A high score in reputation means the company and product have a good reputation, and a high score in specification means that the product fits the application very well. Hence, candidate D has a terrific machine on paper for the application, but with a rather poor reputation relative to the others. Do you agree with what the min-max calculation finds?

	A	B	C	D	E	F
Weight (N)	1260	1190	1470	1540	952	1358
Cost (dollars)	24700	23920	28860	33800	24700	27300
Reputation (nom=1)	1.1	0.7	1.2	0.5	0.8	0.9
Warranty (years)	5	6	8	5	3	6
Efficiency (pct.)	0.51	0.52	0.38	0.36	0.45	0.41
Specifications (nom=1)	1	1.1	1.2	1.4	0.85	1.1

The maximum normalized deviation from peak performance, for each of the six candidates is [0.72, 0.71, 0.88, 1.0, 1.0, 0.69], and so Candidate F is the winner. See the attached code. You might not agree with this result. Perhaps we can weight the deviations to distinguish between those properties that are important vs. those that are not? The purist will say that the less important performance measures shouldn't even make it onto the chart!

```
%-----------------------------------------------------------------------
% minMax optimization
%
clear all;

% give properties for each candidate

weight = [.9 .85 1.05  1.1 .68  .97] * 1400 ;
cost = [.95 .92 1.11 1.3 .95  1.05] * 26000 ;
reputation = [1.1 .7 1.2 .5  .8 .9] * 1 ;
warranty = [5 6 8 5 3 6] ;
efficiency = [.91 .92 .78 .76 .85 .81]-.4;
spec = [1 1.1 1.2 1.4 .85  1.1] ;

% write this data to a file for latex
f = fopen('minMax.dat','w');
fprintf(f,'Weight (N)');fprintf(f,' & %g',weight) ;
fprintf(f,'\\\\ \n');
```

```
fprintf(f,'Cost (dollars)');fprintf(f,' & %g',cost) ;
fprintf(f,'\\\\ \n');
fprintf(f,'Reputation (nom=1)');fprintf(f,' & %g',reputation) ;
fprintf(f,'\\\\ \n');
fprintf(f,'Warranty (years)');fprintf(f,' & %g',warranty) ;
fprintf(f,'\\\\ \n');
fprintf(f,'Efficiency (pct.)');fprintf(f,' & %g',efficiency) ;
fprintf(f,'\\\\ \n');
fprintf(f,'Specifications (nom=1)');fprintf(f,' & %g',spec) ;
fclose(f);

% make matrix:  each candidate gets a column, each attribute is a row.
perf = [-weight ; -cost ; reputation; warranty ; efficiency ; spec]

meanPerf = sum(perf')' ;

maxPerf = max(perf')' ; % the maximum for each attribute
minPerf = min(perf')' ; % minimum for each attribute
rangePerf = maxPerf-minPerf ; % range for each attribute

devPerf = maxPerf*ones(1,size(perf,2)) - perf ; % deviation from peak,
                                                % all cands. and attributes

normDevPerf = devPerf ./ (rangePerf*ones(1,size(perf,2)))
    % normalize deviations with attribute ranges

maxNormDevPerf = max(normDevPerf)       % get the max deviation for the
                                        % candidates

[junk,ind] = sort(maxNormDevPerf) ;
disp(sprintf('The best min-max candidate is Candidate %d', ind(1)));

%-----------------------------------------------------------------------
```

27 Walking Robot Constraints

The "Big Pig" project, which involves a pig-like quadruped robot, wants to understand how to control the device when it operates on a slippery surface. Viewed from above, each foot is at a location described by the vector $\vec{r}_i$, comprising the x (positive forward) and y (positive to the left) coordinates relative to the center of mass; the vertical direction is z. Clearly static stability is possible only if the centroid is contained within one of the triangles created by the planted foot locations. Hence, slow walking can be achieved by moving the centroid into different triangle sets, and moving the feet in turn. Also in every such configuration, the roll and pitch moments have to balance to zero. Things get harder during actual locomotion, because we do not want slippage on any of the feet. This occurs when force in the horizontal plane exceeds a critical fraction of the vertical (normal) force: when $F_{i,x} \geq \mu F_{i,z}$, where μ is the Coulomb friction coefficient. A similar constraint would apply of course in the y-direction.

Here we take on a simplified, three-leg problem that lays out some very practical issues in large and slowly moving walking robots. Consider the statically stable condition `r = [[1.1, 0.8];[1.2,-0.7];[-1.5,-0.7]]` meters, where each pair gives the x and y coordinates of three planted legs (forward left, forward right, back right). The weight of the Big Pig is 4600 Newtons, and the value of μ is 0.12. Along with the non-slip constraint for each foot in the x-direction, we have a mechanical design constraint that the x force magnitude on any foot cannot exceed 250 Newtons. Also, since a foot cannot "pull" on the ground, each $F_{i,z}$ has to be positive. This is a total of nine inequality constraints.

There are also three equality constraints. First, the sum of the vertical forces has to equal the vehicle weight. Second, the sum of moments due to the vertical forces in the pitch direction (nose-up or nose-down) has to be equal to zero. Third, due to the construction of the robot, moments caused by the actuators that make x-direction forces have to equalize front and back: $F_{fl,x} + F_{fr,x} - F_{br,x} = 0$.

Note that forces in the y-direction do not play any role in our problem.

The Question: What is the maximum forward force ($\sum_{i=1}^{3} F_{i,x}$) that can be exerted on the chassis without slipping? Please give the the three F_z's and the three F_x's that go with your answer.

There are three equalities and nine inequalities, for six unknowns. We write:

$$\begin{aligned}
F_{1,z} + F_{2,z} + F_{3,z} &= 4600 \text{ (sum of vertical forces supports the weight)} \\
r_{1,x}F_{1,z} + r_{2,x}F_{2,z} + r_{3,x}F_{3,z} &= 0 \text{ (sum of pitch moments due to vertical forces is zero)} \\
F_{1,x} + F_{2,x} - F_{3,x} &= 0 \text{ (actuator moment constraint)} \\
-F_{1,z} &\leq 0 \text{ (vertical forces positive only for each foot)} \\
-F_{2,z} &\leq 0 \\
-F_{3,z} &\leq 0 \\
-\mu F_{1,z} + F_{1,x} &\leq 0 \text{ (no-slip condition for each foot)}
\end{aligned}$$

$$\begin{aligned} -\mu F_{2,z} + F_{2,x} &\leq 0 \\ -\mu F_{3,z} + F_{3,x} &\leq 0 \\ F_{1,x} &\leq 250 \text{ (absolute horizontal load limit for each foot)} \\ F_{2,x} &\leq 250 \\ F_{3,x} &\leq 250. \end{aligned}$$

Since the solution lies on a vertex in this linear problem, we have to consider all possible combinations of the three equality constraints, and three of the nine inequalities. We solve a 6x6 $Ax = b$ problem for each one, and the number of combinations is $9!/(9-3)!3! = 84$. Solutions are made infeasible by either of the following problems: 1) the 6x6 linear problem cannot be solved because the A-matrix is singular, or 2) the solution violates one of the other six inequalities. I get six feasible solutions:

```
#1 Feasible Solution:
    s[    0.00 2555.56 2044.44 |   0.00 245.33 245.33 ] -> 490.67 N
#2 Feasible Solution:
    s[    0.00 2555.56 2044.44 |  -4.67 250.00 245.33 ] -> 490.67 N
#3 Feasible Solution:
    s[ 2653.85   -0.00 1946.15 | 233.54   0.00 233.54 ] -> 467.08 N
#4 Feasible Solution:
    s[ 2653.85   -0.00 1946.15 | 250.00 -16.46 233.54 ] -> 467.08 N
#5 Feasible Solution:
    s[ 2083.33  549.38 1967.28 | 250.00 -13.93 236.07 ] -> 472.15 N
#6 Feasible Solution:
    s[  490.38 2083.33 2026.28 |  -6.85 250.00 243.15 ] -> 486.31 N
```

and the best of them is 490.67 N. Note the variety of feasible solutions and how close they are to one another!

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solve the Big Pig three-leg, no-slip problem.

% FSH MIT 2.017 March 2008
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

% radius vectors for the feet, meters [x y]
r = [[1.1, 0.8];[1.2,-0.7];[-1.5,-0.7]] ;

W = 4600 ; % weight of the chassis
mu = .12 ; % Coulomb friction coefficient
```

```
maxXForce = 250 ; % maximum x-force allowed on any leg

% plot the layout of the feet
figure(1);clf;hold off;
plot(r(:,1),r(:,2),'ro',0,0,'bs','LineWidth',2);
axis('equal');
grid;

% Unknowns are the three Fz's, and the three Fx's;
% let the vector of unknowns be x = [Fz1 Fz2 Fz3 Fx1 Fx2 Fx3].
% We write everything in the form of Ax = b, where the top part
% has the equalities, and the lower part has the inequalities of the
% form Ax <= b

% three equality constraints first - the first two rows of A and b
A(1,1:3) = ones(1,3) ; % sum of Fz's equals the weight
b(1,1) = 4600 ;
A(2,1:3) = r(:,1)' ; % sum of pitch moments due to Fz's equals zero
b(2,1) = 0 ;
A(3,4:6) = [1 1 -1] ; % sum of moments due to feet actuators is zero
b(3,1) = 0 ;

% now nine inequality constraints - rows 4-12 of A and b
A(4:6,1:3) = -eye(3,3) ; % Fz's are positive only
b(4:6,1) = zeros(3,1) ;
A(7:9,1:3) = -mu*eye(3,3) ; % no slippage:  F_xi - mu*Fzi <= 0
A(7:9,4:6) = eye(3,3) ;
b(7:9,1) = zeros(3,1) ;
A(10:12,4:6) = eye(3,3) ; % upper limit on x-load for each foot
b(10:12,1) = maxXForce*ones(3,1) ;

% We have six unknowns - so at each vertex we have to solve six
% equations.  There are twelve equations total, of which we must
% use the first three (equalities).  We have nine inequalities,
% so there will be 9!/(9-3)!3! = 84 combinations of three
% inequalities to try out.  Re the lecture notes,
I = 9;
E = 3;
n = 6;

% Count out some index sets for the equations we will use and not use in
% each linear system we solve.  These are SPECIFIC to the problem as
% written, i.e., only three equations to use, out of nine possible
```

```
% each row indexes the set of equations to use
ct = 0 ;
for i = 4:10,
    for j = i+1:11,
        for k = j+1:12,
            ct = ct + 1 ;
            indUse(ct,:) = [i j k] ;
        end;
    end;
end;

% each row of indUnused is the set of equations to _not to use_
% (also problem SPECIFIC as written, to three used inequalities)
for i = 1:length(indUse),
    ct = 0 ;
    for j = E+1:I+E,
        if indUse(i,1) ~= j & indUse(i,2) ~= j & indUse(i,3) ~= j,
            ct = ct + 1 ;
            indUnused(i,ct) = j ;
        end;
    end;
end;

% make a plot just to make sure we have no mistakes! (the figure should
% be nothing but a straight line)
figure(2);clf;hold off ; hold on;
for i = 1:length(indUnused),
     plot(sort([indUnused(i,:),indUse(i,:)]));
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here is the linear programming engine, in a general form

% now go through the various n-dimensional subproblems
Atemp(1:E,:) = A(1:E,:) ; % first E rows for all cases - equalities
btemp(1:E,1) = b(1:E,1) ;

for i = 1:length(indUse),
    % make up the other rows we need
    Atemp(E+1:n,:) = A(indUse(i,:),:) ;
    btemp(E+1:n,:) = b(indUse(i,:),1) ;

    % solve the system if Atemp can be inverted
    if rank(Atemp) == n,
```

```
        xtemp(:,i) = Atemp \ btemp ; % (see MATLAB help - solves A*x=b)

        % check against all the unused constraints
        for j = 1:I+E-n,
            if A(indUnused(i,j),:)*xtemp(:,i) > b(indUnused(i,j),:),
                xtemp(:,i) = NaN*ones(n,1); % mark bad ones with NaN
                break ;
            end;
        end;
    else, % if we can't invert Atemp, we cannot get a solution
        xtemp(:,i) = NaN*ones(n,1) ;
    end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% display results and optimum Big Pig x-load
ct = 0 ;
f = fopen('bigPigSolution.dat','w');
fprintf(f,'\n\\begin{verbXXXatim}\n');
for i = 1:length(xtemp),
    if ~isnan(xtemp(1,i)),
        ct = ct + 1 ;
        disp('-------------------------------------------------------');
        disp(sprintf('#%d Feasible Solution [%g %g %g  |  %g %g %g]',...
            ct,xtemp(:,i)'));
        disp(sprintf('has x-load of %g N.',sum(xtemp(4:6,i)')));
        fprintf(f,'#%d Feasible Solution: \n',ct);
        fprintf(f,...
           '    s[ %7.2f %7.2f %7.2f | %6.2f %6.2f %6.2f ] -> %6.2f N\n',...
            xtemp(:,i)',sum(xtemp(4:6,i)'));
    end;
end;
fprintf(f,'\\end{verXXXbatim}\n');
fclose(f) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

28 Floating Structure in Waves

We consider the pitch and heave dynamics of a large floating structure in a random sea. You can consider this a two-dimensional problem.

The structure has two main, identical struts that pierce the water: each has area A_w of two hundred square meters, and their centers are separated by a distance L of fifty meters. The mass center of the structure is at the mid-point. The mass m is 1000 tons, and the mass moment of inertia about the centroid is $J = 4.0 \times 10^5 ton \cdot m^2$. Each hull has an apparent linear damping in the vertical direction of $b = 60kN \cdot s/m$.

The horizontal motion of the structure is nearly zero. The vertical excitation force exerted at each of the struts may be approximated as the stiffness (provided by the strut's water-plane area) times $\eta - \zeta$, where η is the wave elevation at the location of the strut's center, and ζ is the vertical displacement of the strut. Make linearizations where needed. Note we do not take into account any added mass forces in this problem. Also, we assume that the mass center is low on the water, so that the pitching moment is generated exactly by the net loss of flotation on one side and/or the increase of flotation on the other.

For the wave description, we use the Bretschneider spectrum; it is given by

$$\begin{aligned} S(\omega) &= \frac{A}{\omega^5}\, e^{-B/\omega^4}, \text{ where} \\ \omega_m &= \text{ modal (or peak) frequency, rad/s} \\ B &= 1.25\omega_m^4; \qquad A = 4BE_S; \qquad E_S = H_{1/3}^2/16. \end{aligned}$$

In SeaState 5, we take the modal period as 9.7 seconds, and the significant wave height $H_{1/3}$ as 3.3m. We assume that the waves are all traveling in the same direction, from negative x toward positive x.

1. Write a pair of differential equations, that express the heave motion of the center of mass (say $z(t)$), and the pitch motion (say $\phi(t)$), in terms of the wave elevations at the two struts. Hint: use the fact that

$$\zeta(t, -L/2) = z(t) - \phi(t)L/2, \quad \text{and so on.}$$

Solution: We have, using the hint,

$$\begin{aligned} m\ddot{z} &= \rho g A_w[\eta(-L/2) - (z - L\phi/2) + \eta(L/2) - (z + L\phi/2)] - \\ &\quad b[(\dot{z} - L\dot{\phi}/2) + (\dot{z} + L\dot{\phi}/2)] \\ &= \rho g A_w[\eta(-L/2) + \eta(L/2) - 2z] - 2b\dot{z} \\ J\ddot{\phi} &= \rho g A_w \frac{L}{2}[-\eta(-L/2) + (z - L\phi/2) + \eta(L/2) - (z + L\phi/2)] + \\ &\quad \frac{L}{2} b[(\dot{z} - L\dot{\phi}/2) - (\dot{z} + L\dot{\phi}/2)] \\ &= \rho g A_w \frac{L}{2}[\eta(L/2) - \eta(-L/2) - L\phi] - \frac{L^2}{2} b\dot{\phi}. \end{aligned}$$

Because of cancelations in this symmetric situation, we end up with decoupled equations - i.e., the pitch motion does not affect the heave motion, and vice versa.

2. What are the structure's natural frequencies in heave and in pitch? Do these seem like a good design?

   Solution: The natural frequency in heave, obtained from the above equation with no wave excitation ($\eta = 0$) is $\sqrt{2\rho g A_w/m} = 1.98 rad/s$, or a period of $3.2 sec$. The natural frequency in pitch is $\sqrt{\rho g A_w L^2/2J} = 2.48 rad/s$, or a period of $2.5 sec$. These are quite fast compared to the frequencies that we expect in big seas - a three-second wave in the open ocean is generally quite small in amplitude, less than one meter. So it seems like a good design from the resonance point of view.

3. Give a general expression for the wave elevation at one strut, as a function of the wave elevation at the other strut. To do this, write down the elevations for only one wave, at frequency ω. You will use the dispersion relation to work out the wavelength λ, and hence derive a frequency-dependent phase angle.

   Solution: We have

$$\begin{aligned} \eta(L/2) &= \eta(-L/2)\cos\left(\frac{2\pi L}{\lambda}\right) \\ &= \eta(-L/2)\cos\left(\frac{L\omega^2}{g}\right), \end{aligned}$$

   where the second expression is found by substituting the dispersion relation.

4. Insert this result into your set of differential equations, and come up with an ODE for the heave being driven by waves (referenced to $x = -L/2$), and another for the pitch being driven by waves. Note your answers will have a "weird" term similar to $\cos(\omega^2)$, which you can carry directly into the frequency domain (because the Fourier transform is an integration over time!).

   Solution:

$$\begin{aligned} m\ddot{z} + 2b\dot{z} + 2\rho g A_w z &= \rho g A_w(1 + \cos(L\omega^2/g))\eta(-L/2) \\ J\ddot{\phi} + \frac{L^2}{2}b\dot{\phi} + \frac{L^2}{2}\rho g A_w \phi &= -\frac{L}{2}\rho g A_w(1 - \cos(L\omega^2/g))\eta(-L/2). \end{aligned}$$

5. What are the two transfer functions

$$\frac{z(j\omega)}{\eta(j\omega, x = -L/2)} \quad \text{and} \quad \frac{\phi(j\omega)}{\eta(j\omega, x = -L/2)} \; ?$$

   Solution:

$$\begin{aligned} \frac{z(j\omega)}{\eta(j\omega, x = -L/2)} &= \frac{\rho g A_w[1 + \cos(L\omega^2/g)]}{-m\omega^2 + 2bj\omega + 2\rho g A_w} \\ \frac{\phi(j\omega)}{\eta(j\omega, x = -L/2)} &= -\frac{\frac{L}{2}\rho g A_w[1 - \cos(L\omega^2/g)]}{-J\omega^2 + \frac{L^2}{2}bj\omega + \frac{L^2}{2}\rho g A_w}. \end{aligned}$$

6. Make a labeled plot of the Bretschneider wave spectrum for these SS5 conditions.

   See the attached graph.

7. What are the significant heights (double amplitudes) of the heave and the pitch motions? Based on plots of the output spectra, about what are the dominant frequencies of these two motions?

   Solution: The significant heave height is 3.30m and the significant pitch height is 0.121rad. The output spectra plots (attached) are interesting because the dominant frequency in the heave direction is close to the resonance, around 1.9rad/s, or 3.3 seconds period. On the other hand, the pitch direction has a dominant frequency much lower, at about 0.75rad/s, or 8.4 seconds period - this is closer to the excitation frequency. You see that the effect of the different wavelengths gives rise to many peaks in the responses, and this ultimately controls the output spectra shapes.

8. What are the heave and pitch amplitudes expected to be exceeded in ten minutes? one hour? one day?

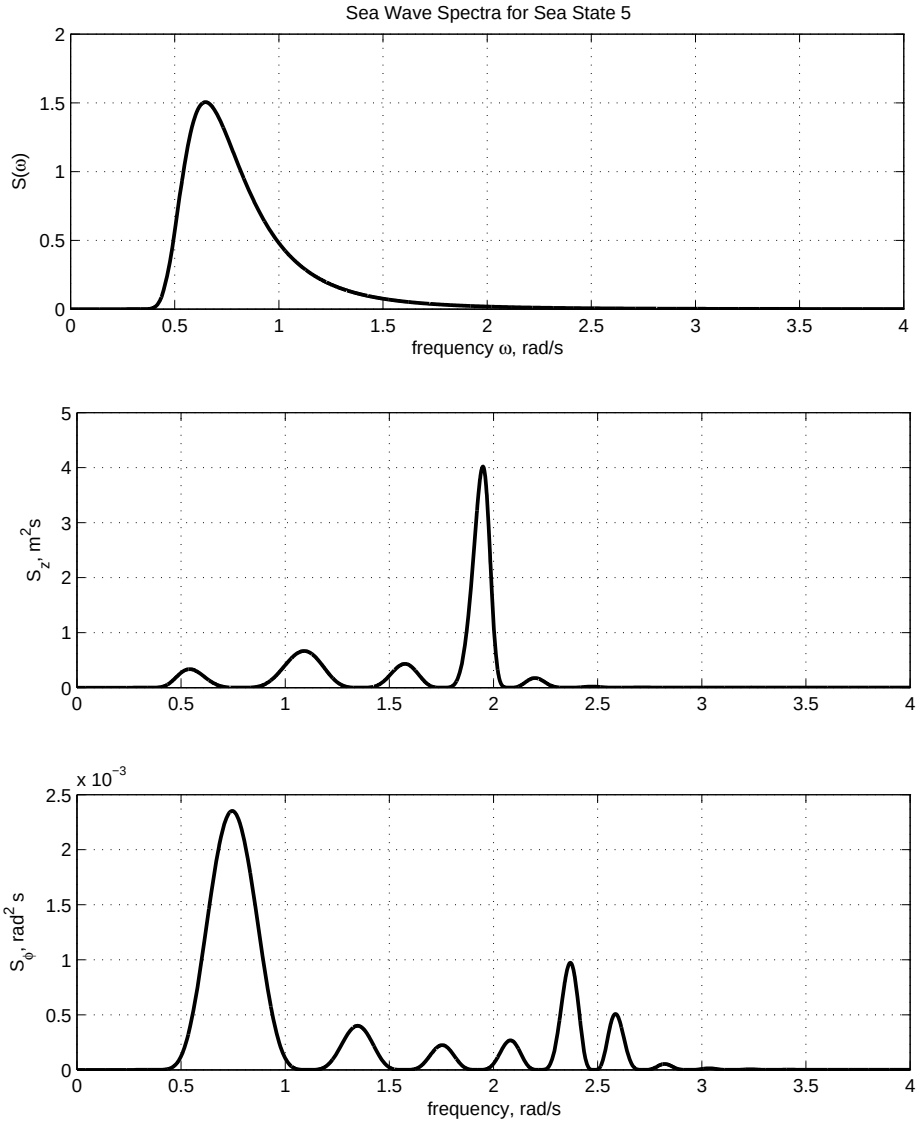
   Solution: For heave the amplitudes that will be exceeded are [2.63 3.06 3.70] meters; for pitch, we find [0.094 0.110 0.134] radians. The formulas are (for the z part):

$$\begin{aligned} M_{iz} &= \int_0^\infty \omega^i S_z(\omega) d\omega \\ \bar{T}_z &= 2\pi\sqrt{\frac{M_{0z}}{M_{2z}}} \\ \bar{f}(A_z) &= [1/600 \quad 1/3600 \quad 1/86400] \ Hertz \\ A_z &= \sqrt{-2M_{0z}\log\left(\bar{f}(A_z)\bar{T}_z\right)}. \end{aligned}$$

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Heave and pitch response of a large floating structure

clear all;

wm = 2*pi/9.7 ; % modal frequency of waves, rad/s
Hsig = 3.3 ; % significant wave height, m
wvec = 0.001:0.001:4 ; % vector of frequencies to consider

% make up the Bretschneider spectrum
for j = 1:length(wvec),
    w = wvec(j) ;
    S(j) = 5/16 * wm^4 / w^5 * Hsig^2 * exp(-5 * wm^4 / 4 / w^4) ;
end;
```

```
% check that we got the right formula!
disp(sprintf(...
    'Square Root of Integral of Area of S:  %g;  Hsig/4:  %g', ...
    sqrt(sum(S)*mean(diff(wvec))), 1/4*Hsig));

% plot the spectrum
figure(1);clf;hold off;
subplot(211);
plot(wvec,S,'LineWidth',2);
grid;
title('Sea Wave Spectra for Sea State 5');
xlabel('frequency \omega, rad/s');
ylabel('S(\omega)');

print -deps bigStructure.eps

% give the physical parameters of the structure
L = 50 ; % distance between flotation centers, m
m = 1e6 ; % material mass, kg
J = 4e8 ; % material rotary moment of inertia, kg-m^2
Aw = 200 ; % waterplane area per hull, m^2
rho = 1000 ; % water density, kg/m^3
g = 9.81 ; % gravity, m/s^2
b = 60000 ; % linear damping coefficient in vertical direction,
            % per hull, N/(m/s)

% compute the numerical transfer functions from wave elevation at
% x=-L/2 to height (z) and pitch angle (phi).  The units are
% meter/meter and rad/meter
for i = 1:length(wvec),
    w = wvec(i) ;
    eta2z(i) = (1 + cos(L*w*w/g))*rho*g*Aw / ...
        (-m*w^2 + sqrt(-1)*w*2*b + 2*g*Aw*rho) ;
    eta2phi(i) = -(1 - cos(L*w*w/g))*rho*g*Aw*L/2 / ...
        (-J*w^2 + sqrt(-1)*w*L^2/2*b + rho*g*Aw*L^2/2) ;
end;

% (undamped) natural frequencies
wnz = sqrt(2*g*Aw*rho/m) ;
wnphi = sqrt(rho*g*Aw*L^2/2/J) ;
disp(sprintf('Natural frequencies:  %g rad/s (heave)', wnz));
disp(sprintf('                      %g rad/s (pitch)', wnphi));

% get the spectra of the heave motion and the pitch angle.
```

```
% (Take only the real part because there are some parasitic,
% tiny imaginary parts that are annoying.)
Sz = real(S.*eta2z.*conj(eta2z)) ;
Sphi = real(S.*eta2phi.*conj(eta2phi)) ;

% plot these spectra
figure(2);clf;hold off;
subplot(211);
plot(wvec,Sz,'LineWidth',2);
grid;
ylabel('S_z, m^2s');
subplot(212);
plot(wvec,Sphi,'LineWidth',2);
grid;
ylabel('S_{\phi}, rad^2 s');
xlabel('frequency, rad/s');
print -deps bigStructure2.eps

% compute the significant heights of the heave and the pitch motions
zsig = 4*sqrt(sum(Sz)*mean(diff(wvec))) ;
phisig = 4*sqrt(sum(Sphi)*mean(diff(wvec))) ;
disp(sprintf('Significant z height:  %g m', zsig));
disp(sprintf('Significant phi height:  %g rad',phisig));

% compute the ten-minute, one-hour, and one-day extreme amplitudes
M0z = sum(Sz)*mean(diff(wvec)); % moments of the spectra
M2z = sum(Sz.*wvec.^2)*mean(diff(wvec));
M0phi = sum(Sphi)*mean(diff(wvec));
M2phi = sum(Sphi.*wvec.^2)*mean(diff(wvec));

Tbarz = 2*pi*sqrt(M0z/M2z) ; % average period
Tbarphi = 2*pi*sqrt(M0phi/M2phi) ;

fbarA = [1 1 1]./[10*60  60*60  24*60*60] ; % event frequencies, Hz

Az = sqrt(-2*zsig^2/16*log(Tbarz*fbarA));
Aphi = sqrt(-2*phisig^2/16*log(Tbarphi*fbarA));

disp('Ten-minute, one-hour, and one-day amplitudes:');
disp(sprintf('Heave:  %g  %g  %g m', Az(1),Az(2),Az(3)));
disp(sprintf('Pitch:  %g  %g  %g rad', Aphi(1),Aphi(2),Aphi(3)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```


Sea Wave Spectra for Sea State 5
S(ω)
frequency ω, rad/s
S_z, m^2s
x 10^{-3}
S_ϕ, $rad^2 s$
frequency, rad/s

29 Flight Control of a Hovercraft

You are tasked with developing simple control systems for two types of hovercraft moving in the horizontal plane. As you know, a hovercraft rests on a cushion of air, with very little ground resistance to motion in the surge (body-referenced forward), sway (body-referenced port), and yaw (taken positive counterclockwise viewed from above) degrees of freedom. The simplified dynamic equations of motion are:

$$\begin{aligned} \dot{u} &= -u + F_u \\ \dot{v} &= F_v \\ \dot{r} &= M, \end{aligned}$$

where the surge and sway velocities are u and v and the control forces in the u- and v-directions are F_u and F_v, respectively. The yaw rate is r and the control moment is M.

There are two types of craft we will consider: 1) one where F_u, F_v, and M are all commanded independently, e.g., using independent thrusters, and 2) one where $F_v = p_v \delta F_u$ and $M = p_\phi \delta F_u$. This second case is encountered if there is a surge thruster (or set), whose exhaust pushes on a rudder with (body-referenced) angle δ. To create a lateral force or moment, you have to have some F_u and some rudder action. We will use the values $p_v = 0.2$ and $p_\phi = -0.2$ in this problem. Notice negative sign in p_ϕ; it means that a positive rudder action, counterclockwise if viewed from above, leads to a negative body torque, that is, clockwise. This is typical when the rudder is behind the thruster, and near the back of the craft.

The dynamic equations are augmented with some kinematic relations to evolve the location of the craft in a fixed frame:

$$\begin{aligned} \dot{X} &= u\cos\phi - v\sin\phi \\ \dot{Y} &= u\sin\phi + v\cos\phi \\ \dot{\phi} &= r, \end{aligned}$$

where X and Y denote the location in Cartesian coordinates, and ϕ is the yaw angle. See the figure below.

Create a six-state model for each vehicle type from the above equations, and perform the following tasks:

1. By putting in different settings and combinations of F_u, F_v, M, δ, convince yourself that for each of the two cases, the behavior is like a hovercraft. In the first case, turns occur completely independently of translational motion in the X, Y plane, whereas for the second, lateral force and turning torque occur together, and they scale with both δ and F_u.

2. For Case 1: From a full-zero starting condition $\vec{s} = [u, v, r, X, Y, \phi] = [0, 0, 0, 0, 0, 0]$, the objective is to move to to $\vec{s}_{desired} = [u, v, r, X, Y, \phi]_{desired} = [0, 0, 0, 1m, 1m, \pi rad]$

under closed-loop control. To do this, inside the derivative call for your simulation, make three error signals:

$$\begin{aligned} e_X &= X - X_{desired} \\ e_Y &= Y - Y_{desired} \\ e_\phi &= \phi - \phi_{desired}. \end{aligned}$$

Then position errors in the body-referenced frame are a simple rotational transformation

$$\begin{aligned} e_u &= \cos\phi\ e_X - \sin\phi\ e_Y \\ e_v &= \sin\phi\ e_X + \cos\phi\ e_Y. \end{aligned}$$

Your control law then concludes with:

$$\begin{aligned} F_u &= -k_{p,u}e_u - k_{d,u}u \\ F_v &= -k_{p,v}e_v - k_{d,v}v \\ M &= -k_{p,\phi}e_\phi - k_{d,\phi}r, \end{aligned}$$

where the k_p's and k_d's are gains proportional to the error and to the derivative of the error, in each channel, and choosing these is your main job. Start with small positive values and you will see your feedback system start to work! We say that this system is fully actuated, since you can independently control all the forces and the moment.

To see why the above rules work in a basic sense, consider the yaw direction only. With the feedback, the complete governing equation is

$$\begin{aligned} J\dot{r} &= -k_{p,\phi}(\phi - \phi_{desired}) - k_{d,\phi}r, \text{ or,} \\ J\ddot{\phi} + k_{d,\phi}\dot{\phi} + k_{p,\phi}\phi &= k_{p,\phi}\phi_{desired}. \end{aligned}$$

You see that this is a stable second-order oscillator whose parameters you get to tune, and that the steady-state solution is $\phi = \phi_{desired}$. For the yaw control problem, an important practical note is: don't let e_ϕ get outside of the range $[-\pi, \pi]$, or you will be turning the long way around. A couple of if/then's can make sure of this.

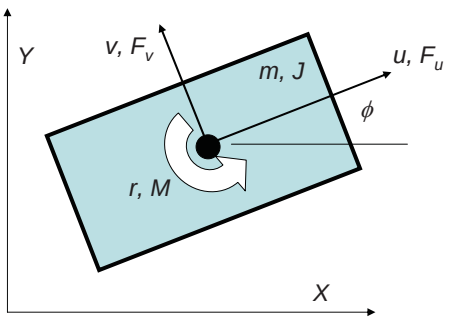
Provide a clear, marked listing of your control code (in one block please), time plots of all six channels of $\vec{s}$ through time, and an X, Y plot of the vehicle trajectory in the plane. These will show that the vehicle actually did what we wanted - to move from the origin to $\vec{s}_{desired}$.

3. Case 2: Develop a mission controller that maneuvers the vehicle from an all-zero starting condition to the state $[u, v, r, X, Y, \phi] = [1m/s, 0m/s, 0rad/s, 20m, 20m, \pi/2rad]$. In other words, the vehicle moves while rotating a quarter-turn, and passes through the desired X, Y location at speed and with no yaw rate.

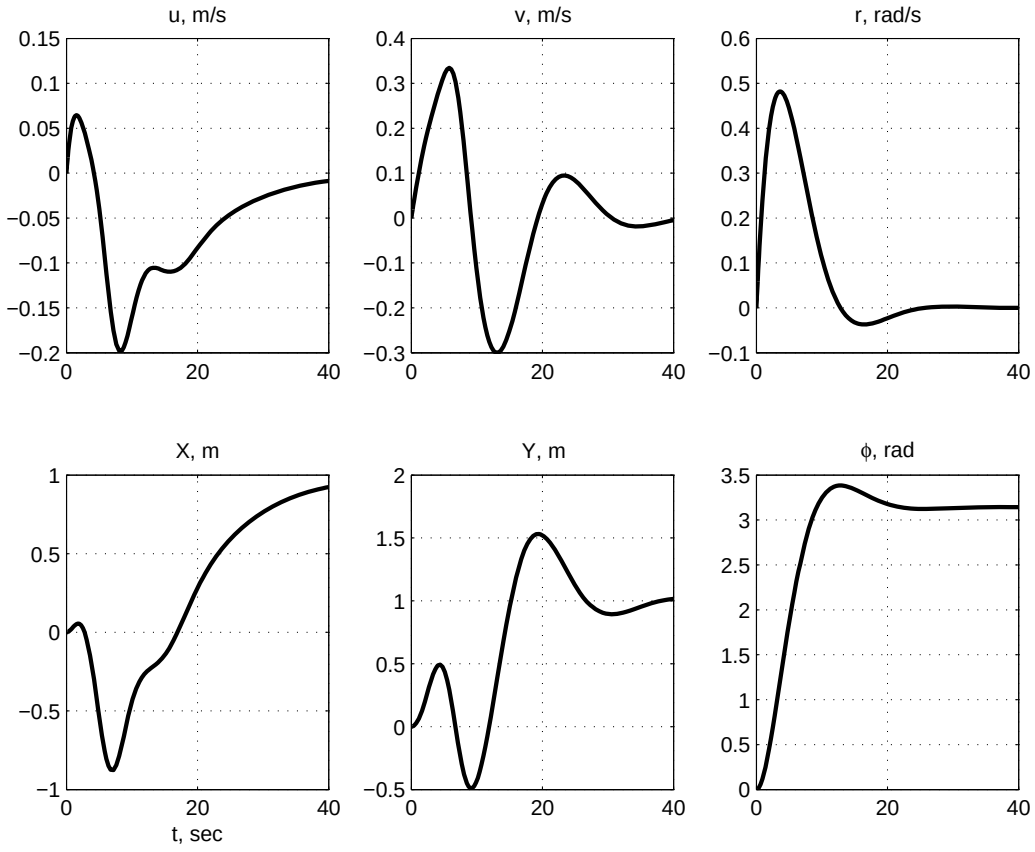
You can't use the same strategy as for Case 1, because the craft can only actuate sway and yaw together. We say that this system is *under-actuated*. It is not acceptable to create by hand a sequence of control actions that achieves the move, or to create an algorithm that uses specific times; they will never work in practice due to modeling errors and external disturbances. What I do recommend is that you use an intermediate *waypoint* on the way to the goal. This is a target X, Y that you go to first, so that the vehicle will then line up well with the goal. Airplanes are doing this sort of maneuver when they circle an airport in preparation for landing. Your mission procedure is to head for the intermediate waypoint until the craft arrives within a small radius of it (several meters, say), and then head to the goal. One well-chosen intermediate waypoint should be enough for this problem, but you could use more if you like.

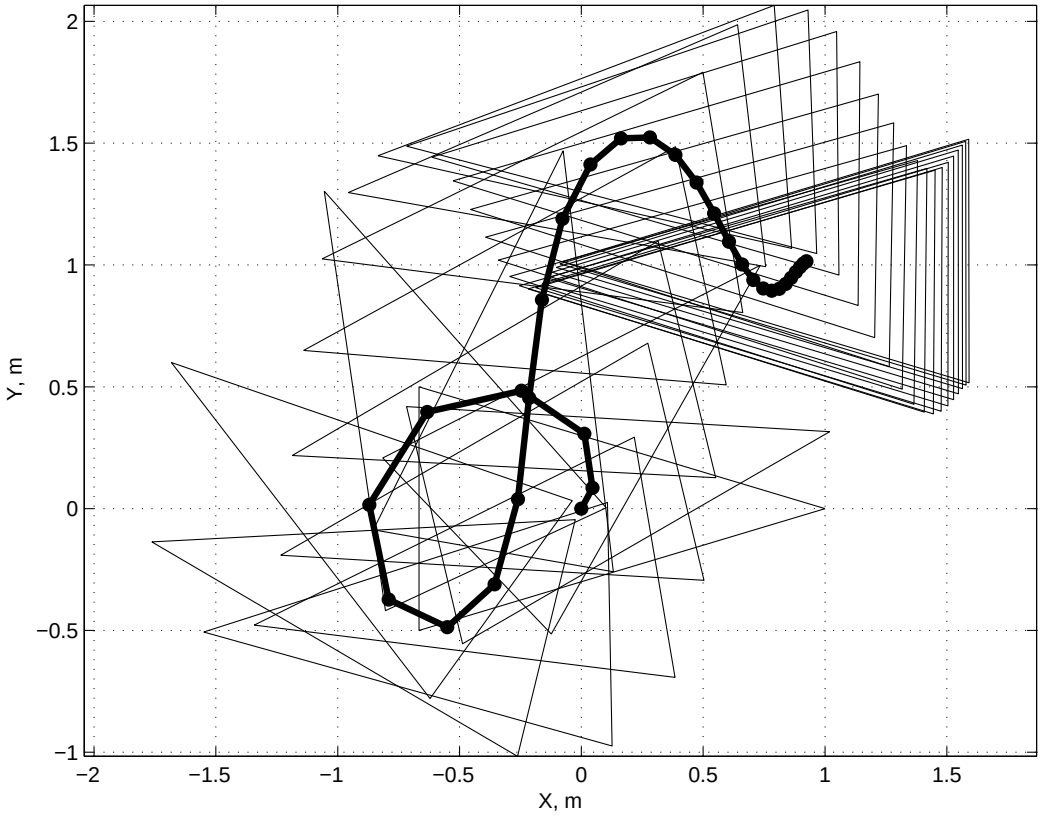
The low-level control strategy is also different from Case 1. Here, you will set $F_u = 1$ to move at a steady speed of $1m/s$, and then command δ so as to drive the vehicle to the desired X, Y. How to do this? Try setting $\phi_{desired} = atan2(Y - Y_{desired}, X - X_{desired})$. Remember that positive δ gives a negative moment.

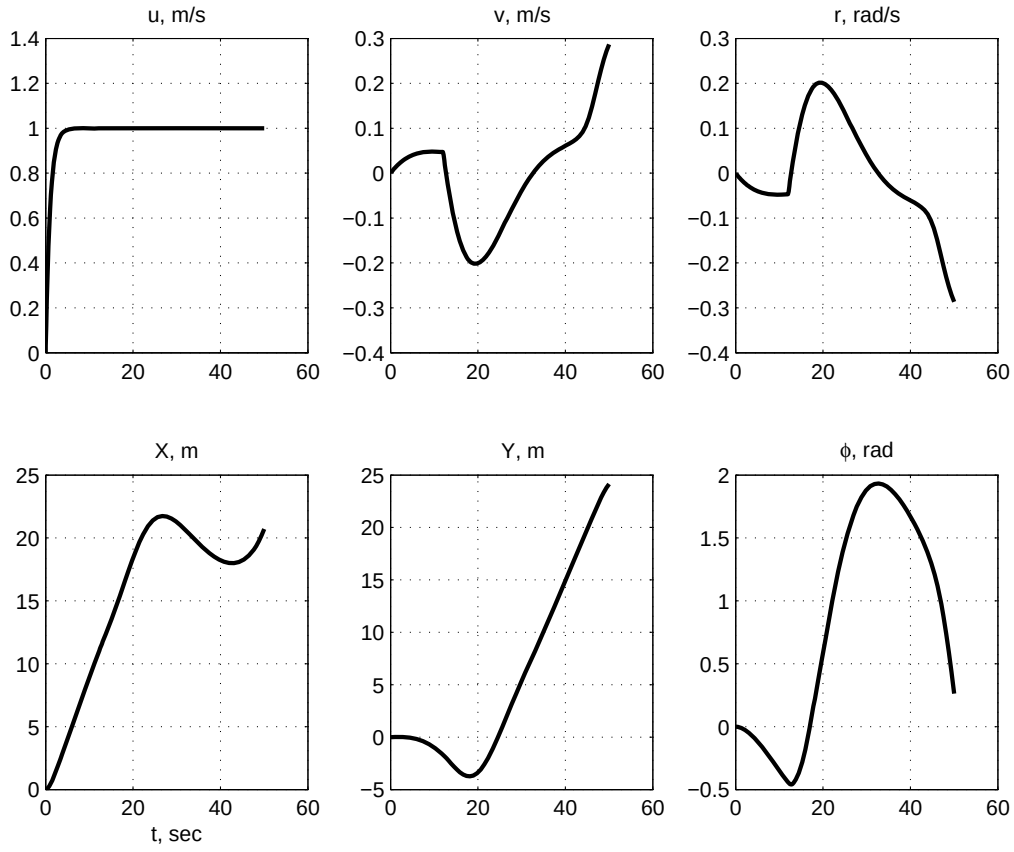
Show the same plots and listing here that you did for Case 1.

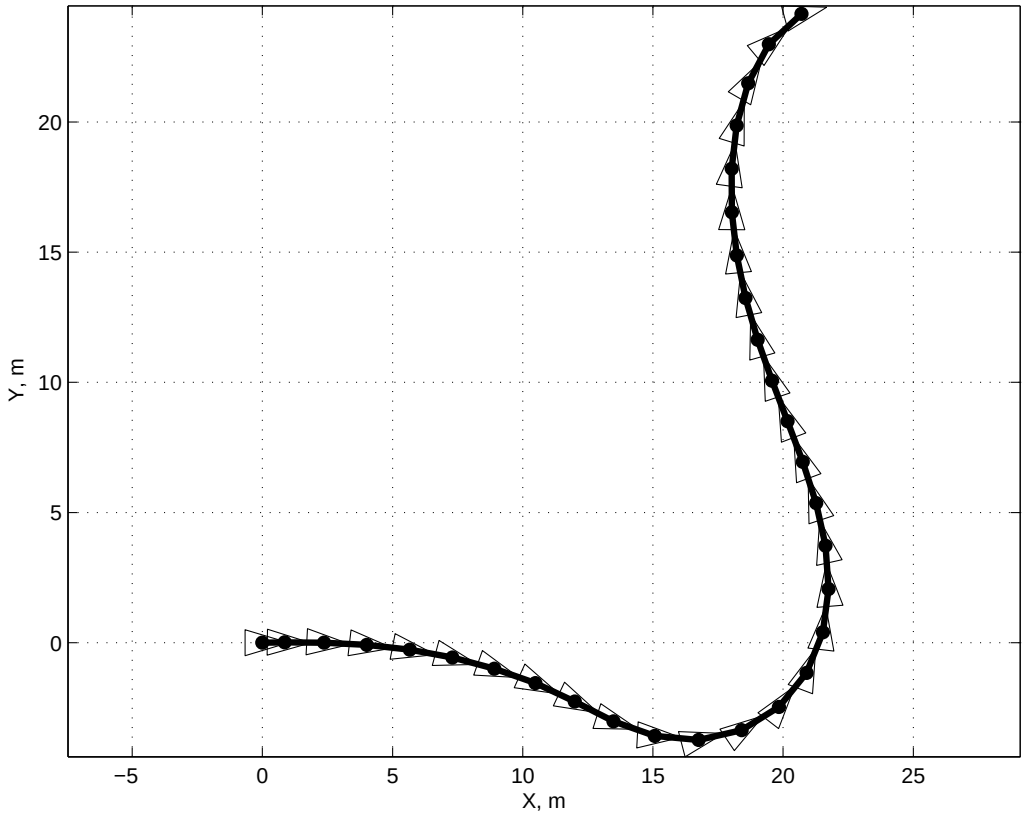


Plots and code are below; these generally play out the strategy described. In Case 1, one finds that setting the heading closed-loop behavior to be faster than that for X, Y may lead to better performance, because there is some coupling. Case 2 is effectively a point-and-go approach, with the critical use of an intermediate waypoint at $[X, Y] = [15, -5]m$ to line up the vehicle for the final approach to the goal.


u, m/s
v, m/s
r, rad/s
X, m
Y, m
ϕ, rad
t, sec


2
1.5
1
0.5
0
−0.5
−1
Y, m
−2
−1.5
−1
−0.5
0
0.5
1
1.5
X, m


u, m/s
v, m/s
r, rad/s
X, m
Y, m
ϕ, rad
t, sec


Y, m
X, m
20
15
10
5
0
−5
0
5
10
15
20
25

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Study dynamics and control of a hovercraft.
% FSH MIT 2.017 Oct 2009

clear all;
global captureRadius flag ; % for Case 2's intermediate waypoint

xCase = input('Case 1 or Case 2?  ');

captureRadius = 5 ; % used for switching waypoints in Case 2
flag = 0 ; % toggle for switching waypoints in Case 2 ;

if xCase == 1,
    tfinal = 40;
    [t,s] = ode45('hoverCraftDeriv1',tfinal,zeros(6,1));
else,
    tfinal = 50 ; % for Case 2
    [t,s] = ode45('hoverCraftDeriv2',tfinal,zeros(6,1));
end;

figure(1);clf;hold off;
for i = 1:6,
    subplot(2,3,i);
    plot(t,s(:,i),'LineWidth',2);
    grid;
end;
subplot(2,3,4);
xlabel('t, sec');
subplot(2,3,1);title('u, m/s');   subplot(2,3,2);title('v, m/s');
subplot(2,3,3);title('r, rad/s'); subplot(2,3,4);title('X, m');
subplot(2,3,5);title('Y, m');     subplot(2,3,6);title('\phi, rad');

tr = 0:tfinal/30:tfinal ; % regularly-spaced time
sr = spline(t,s',tr)';    % spline onto regularly-spaced time

figure(2);clf;hold off;
cphi = cos(sr(:,6)) ; sphi = sin(sr(:,6));
xbox = [1 -2/3 -2/3 1] ;
ybox = [0 1/2 -1/2 0];
center = [1 1 1 1];
plot(sr(:,4),sr(:,5),'k','LineWidth',3);
hold on;
for i = 1:length(cphi),
    plot(sr(i,4)*center + cphi(i)*xbox - sphi(i)*ybox, ...
```

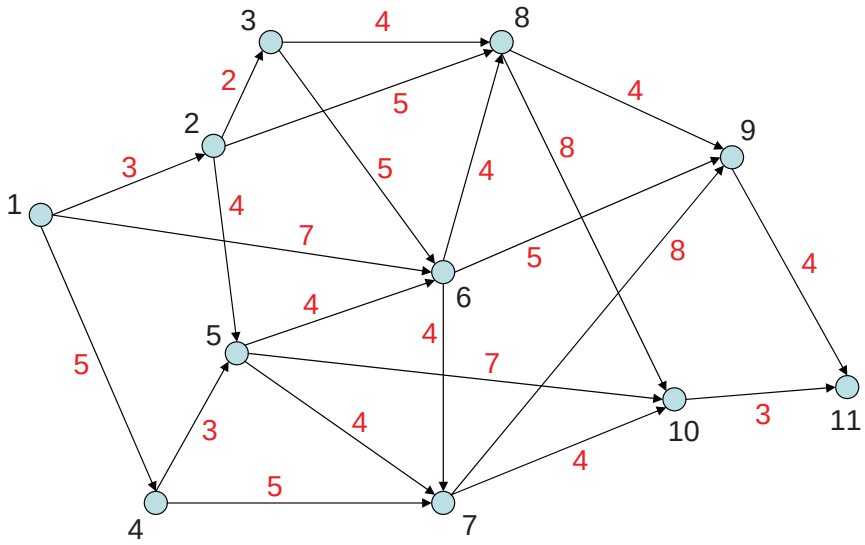
```
        sr(i,5)*center + sphi(i)*xbox + cphi(i)*ybox);
    plot(sr(i,4),sr(i,5),'.k','MarkerSize',20);
end;
axis('equal');
grid;
xlabel('X, m');
ylabel('Y, m');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [sdot] = hoverCraftDeriv1(t,s);

u = s(1) ; v = s(2) ; r = s(3) ; X = s(4) ; Y = s(5) ; phi = s(6) ;
cphi = cos(phi) ;  sphi = sin(phi) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%
% here is the control logic

desPhi = pi ;
errPhi = phi-desPhi ;
if errPhi < -pi, errPhi = errPhi + 2*pi ; end ;
if errPhi > pi, errPhi = errPhi - 2*pi ; end ;

desX = 1 ;
errX = X - desX ;
desY = 1 ;
errY = Y - desY ;

Fu = -.1*(errX*cphi - errY*sphi) ;
Fv = -.1*(errX*sphi + errY*cphi) - .3*v ;
M = -.1*(phi-desPhi)-.4*r ;

%%%%%%%%%%%%%%%%%%%%%%%%%%

udot = -u + Fu ;
vdot = Fv ;
rdot = M ;
Xdot = cphi*u - sphi*v ;
Ydot = sphi*u + cphi*v ;
phidot = r ;
sdot = [udot ; vdot ; rdot ; Xdot ; Ydot ; phidot] ;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [sdot] = hoverCraftDeriv2(t,s);
global flag captureRadius ;

u = s(1) ; v = s(2) ; r = s(3) ; X = s(4) ; Y = s(5) ; phi = s(6) ;
cphi = cos(phi) ;  sphi = sin(phi) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%
% here is the control logic
if ~flag,
    waypointX = 15 ;
    waypointY = -5 ;
else,
    waypointX = 20 ;
    waypointY = 20 ;
end;

desPhi = atan2(waypointY-Y,waypointX-X);
errPhi = phi-desPhi ;
if errPhi < -pi, errPhi = errPhi + 2*pi ; end ;
if errPhi > pi, errPhi = errPhi - 2*pi ; end ;

del = 0.2*errPhi + r ;

if norm([X-waypointX Y-waypointY]) < captureRadius,
    flag = 1 ;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%

Fu = 1 ;
Fv = .2*del*Fu ;
M = -.2*del*Fu ;

udot = -u + Fu ;
vdot = Fv ;
rdot = M ;
Xdot = cphi*u - sphi*v ;
Ydot = sphi*u + cphi*v ;
phidot = r ;
sdot = [udot ; vdot ; rdot ; Xdot ; Ydot ; phidot] ;
```

%%%

30 Dynamic Programming for Path Design

Given the transition costs in red, what are the maximum and minimum costs to get from node 1 to node 11? This situation is encountered when planning paths for autonomous agents moving through a complex environment, e.g., a wheeled robot in a building.



Solution: The minimum cost is 16 (path [1,6,9,11] or [1,2,8,9,11]) and the maximum value is 28 (path [1,4,5,6,7,9,11]!). The attached code uses value iteration to find these in two and five iterations, respectively.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Value iteration solution of deterministic dynamic programming.
% The program looks complicated only because I cover both
% minimization and maximization in the same program!

clear all;

ch = input('Find minimum (0) or maximum (1):  ');
if ~ch,
    init = 1e6 ;    % look for minimum;
                    % big initial guesses for costs to go
else,
    init = 1e-6 ;   % look for maximum;
                    % small initial guesses for value to go
end;

% interconnect matrix:  row is the node (first is starting
```

```
% point) and column is the set of nodes pointed to.  Note
% that the ending node is not included because it points to
% nowhere.
I = [[2 6 4]        % node 1 (start) points to nodes 2,6,4
    [3 8 5]         % node 2 points to nodes 3,8,5.  And so on...
    [8 6 NaN]       % node 3
    [5 7 NaN]       % node 4
    [6 7 10]        % node 5
    [8 9 7]         % node 6
    [10 9 NaN]      % node 7
    [10 9 NaN]      % node 8
    [11 NaN NaN]    % node 9
    [11 NaN NaN]];  % node 10

% cost per link - these go with the interconnects in A.  Note
% that the entries with direct connection to the end node are NaN,
% because we will enforce the link cost in ctg (below) explicitly
C = [[3 7 5]        % The cost is 3 to move between nodes 1 and 2,
                    % and 7 to move between nodes 1 and 6, etc.
    [2 5 4]         % node 2
    [4 5 NaN]       % node 3
    [3 5 NaN]       % node 4
    [4 4 7]         % node 5
    [4 5 4]         % node 6
    [4 8 NaN]       % node 7
    [8 4 NaN]       % node 8
    [NaN NaN NaN]   % node 9
    [NaN NaN NaN]]; % node 10

% initial guess of cost-to-go (or value-to-go) at each node
tg = [[NaN]         % node 1
    [init]          % node 2
    [init]          % node 3
    [init]          % node 4
    [init]          % node 5
    [init]          % node 6
    [init]          % node 7
    [init]          % node 8
    [4]             % node 9 (points directly to end, node 11)
    [3]];           % node 10 (points directly to end, node 11)

w = size(I,2);  % width of interconnect matrix

disp(sprintf('%g ',tg));    % list the first cost-to-go or
```

```
                            % value-to-go

for k = 1:5,   % carry out a fixed number of iterations

    % cycle through the nodes one by one.  Note that we don't
    % need to recompute tg for nodes that point to the end
    for i = 1:sum(~isnan(C(:,1))),

        % We'll look for the minimum estimated cost-to-go
        % (or maximum estimated value-to-go) across
        % the possible nodes pointed to
        if ~ch,
            dummy = 1e6 ;  % initialize to be huge
        else,
            dummy = 1e-6 ;  % initialize to be tiny
        end;

        % look at all the nodes pointed to from node i
        for j = 1:w,
            if ~isnan(I(i,j)), % consider only true entries in I

                test = tg(I(i,j)) + C(i,j) ;

                if ~ch,     % look for minimum
                    if test < dummy,
                        dummy = test ;
                    end;
                else,       % look for maximum
                    if test > dummy,
                        dummy = test ;
                    end;
                end;

            end;    % "true entries"
        end;    % j: nodes pointed to
        tg(i) = dummy ;
    end;    % i: nodes
    disp(sprintf('%g ',tg));
end;    % k: iteration

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

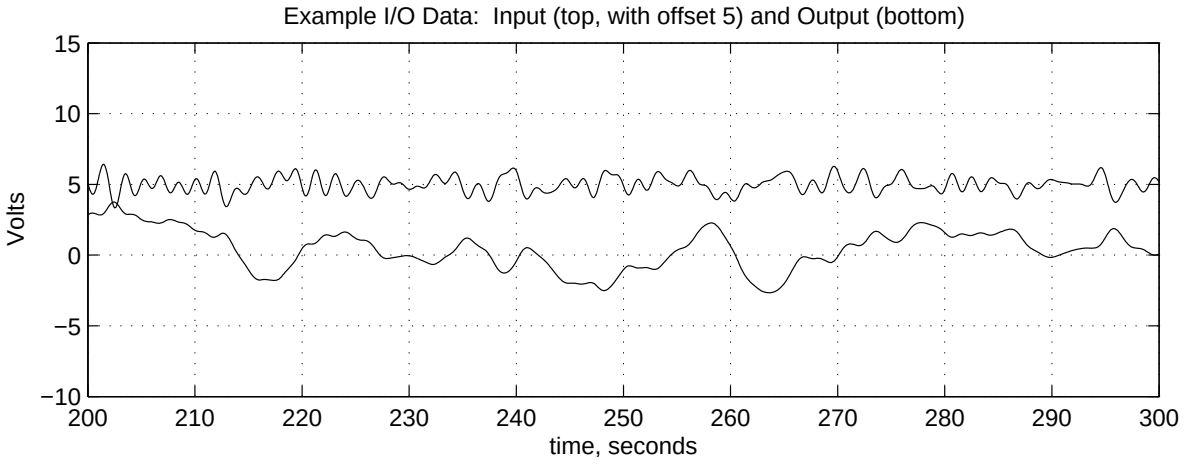
31 Identification of a Response Amplitude Operator from Data: Redux

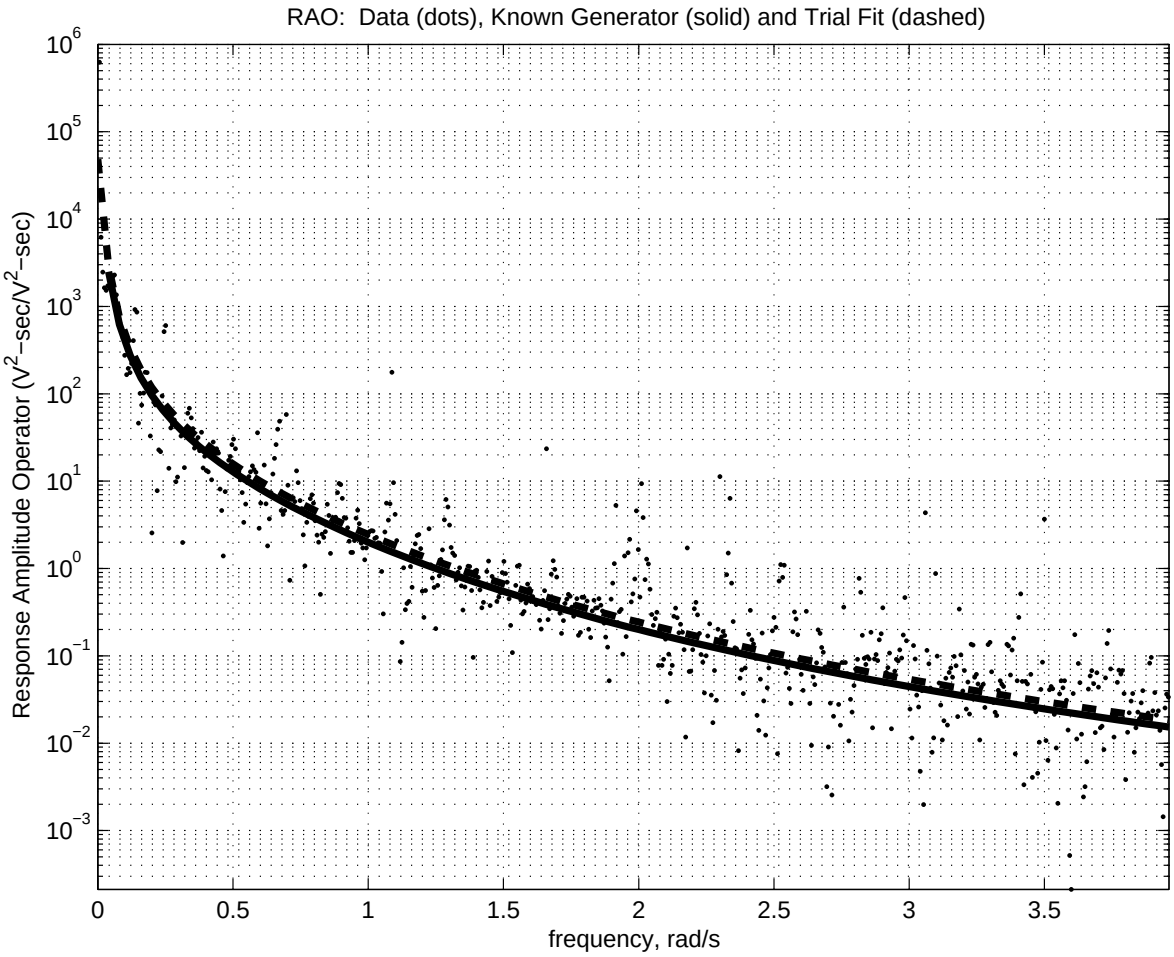
The problem here is almost exactly the same as in HW5, for which you have the full solution. The only differences are:

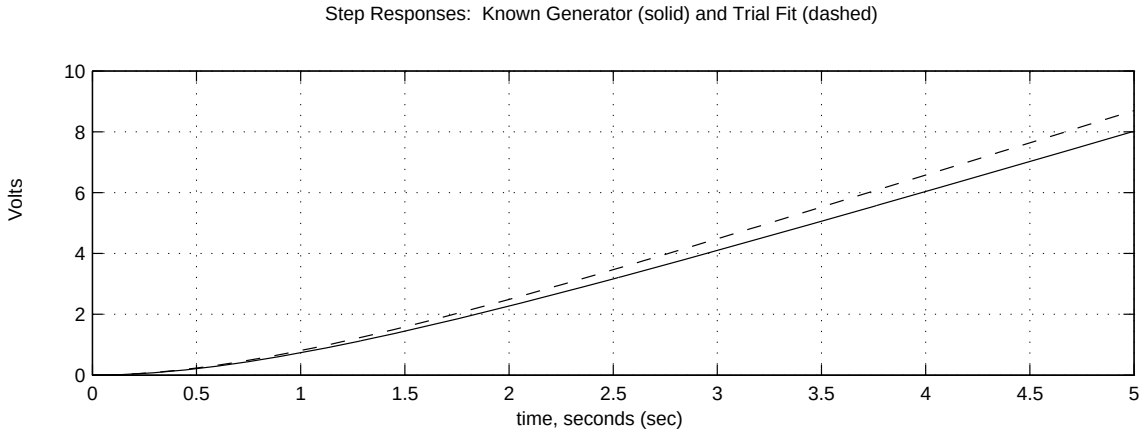
- Load the file `homework7.dat` from course.
- The frequency range of interest is now up to $4rad/s$, whereas before it was $2rad/s$.
- The noise levels on u and y are lower.
- Just looking at the input and output time plots, you can see that the dynamical system at work here is different. However, it is still second-order, and definitely does not have a zero: $y'' + by' + cy = ku$ describes it. Your task is (again) to find the best fits for $[b, c, k]$ based on the RAO plot, list the transfer function, and show the step response.

See the attached plots, made from virtually the same code as in HW5. The generating transfer function is $G(j\omega) = 2/(-\omega^2 + j\omega)$, $[b, c, k] = [1, 0, 2]$, *and the trial fit shown is* $G(j\omega) = 2.2/(-\omega^2 + j\omega + 0.01)$, $[b, c, k] = [1, 0.01, 2.2]$. *This is a low-pass system, in which any low-frequency input is greatly amplified. You can see this behavior directly the transfer function, where the denominator heads toward zero as* $\omega \to 0$, *and the transfer function gets arbitrarily large.*

Note that I did not use the same data as in the file `homework7.dat`, *for these solutions.*




RAO: Data (dots), Known Generator (solid) and Trial Fit (dashed)
Response Amplitude Operator (V2–sec/V2–sec)
frequency, rad/s

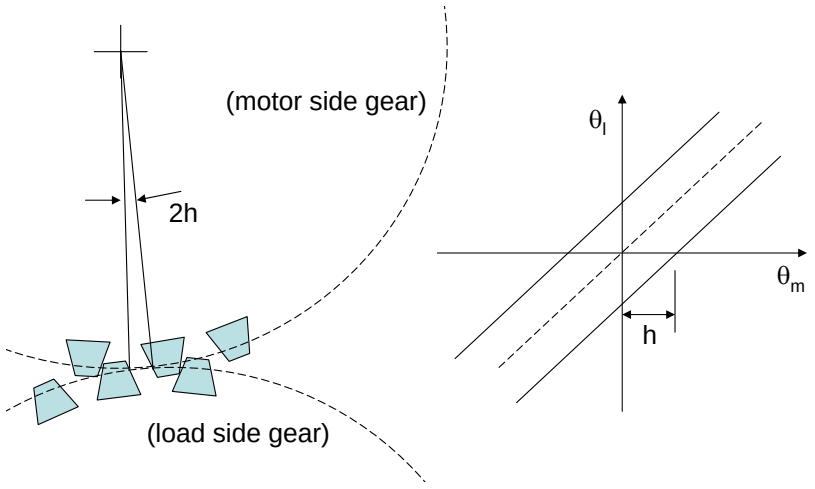

Step Responses: Known Generator (solid) and Trial Fit (dashed)
Volts
time, seconds (sec)

32 Motor Servo with Backlash

Consider the following: a motor is attached to a load through a gearbox, and the gearbox has a backlash behavior. Physically, backlash is a decoupling of gear teeth (due to imprecise meshing), so that the motor-side and the load-side teeth are not always in contact. Contacts are characterized by the condition $|\theta_m - \theta_l| > h$, where the θ's are the motor and load angles respectively, and h is the half-width of the backlash. During contact, a small deflection material occurs according to a linear spring model. In the forward direction, that is, when θ_m leads θ_l by a positive amount,

$$\tau_t = k(\theta_m - \theta_l - h) \text{ when } \theta_m - \theta_l > h,$$

where τ_t is the torque transmitted from the motor side to the load. A similar condition holds in the reverse direction. Conversely, whenever θ_m is within h of the angle θ_l, the teeth are out of contact and no torque is passed between the motor and the load.



The rotary mass on the motor side is J_m; on the load side, we have rotary mass J_l as well as a rotary damping b. It is desired to position the load via a servo, so a simple feedback controller is implemented of the form $\tau_m = -g\theta_l$, where τ_m is the motor torque. We assume that we can control the torque in the motor directly through a current amplifier.

All the physical properties in a particular system built by Company XYZ are well understood, *except* for the load-side damping b. This varies slowly over time because the bearings break in and then age, and because the load side is connected to an unpredictable environment. For the purposes of this problem, we assume that b is a random variable, described by a uniform probability density function, with minimum value zero and maximum value 0.02 Nm/(rad/s). The other parameters in the model are

```
Jm = 0.003 ;  % motor-side inertia, kg-m^2
Jl = 0.01 ;      % load-side effective inertia, kg-m^2
```

```
k = 10000 ;   % stiffness of tooth contact point, Nm/rad
h = 0.02 ;       % backlash half-width, rad
g = 6 ;          % position feedback gain, Nm/rad
```

The standard transient response that we are interested in has the initial condition of zero speeds, zero motor position, and a load position of h. We take as characteristic outputs of any particular model run the two-norms of the load-side position error, and the commanded torque. The two-norm of a signal over a time interval $[t_1, t_2]$ is defined as:

$$||z(t)||_2 = \sqrt{\int_{t_1}^{t_2} z^2(t)dt}$$

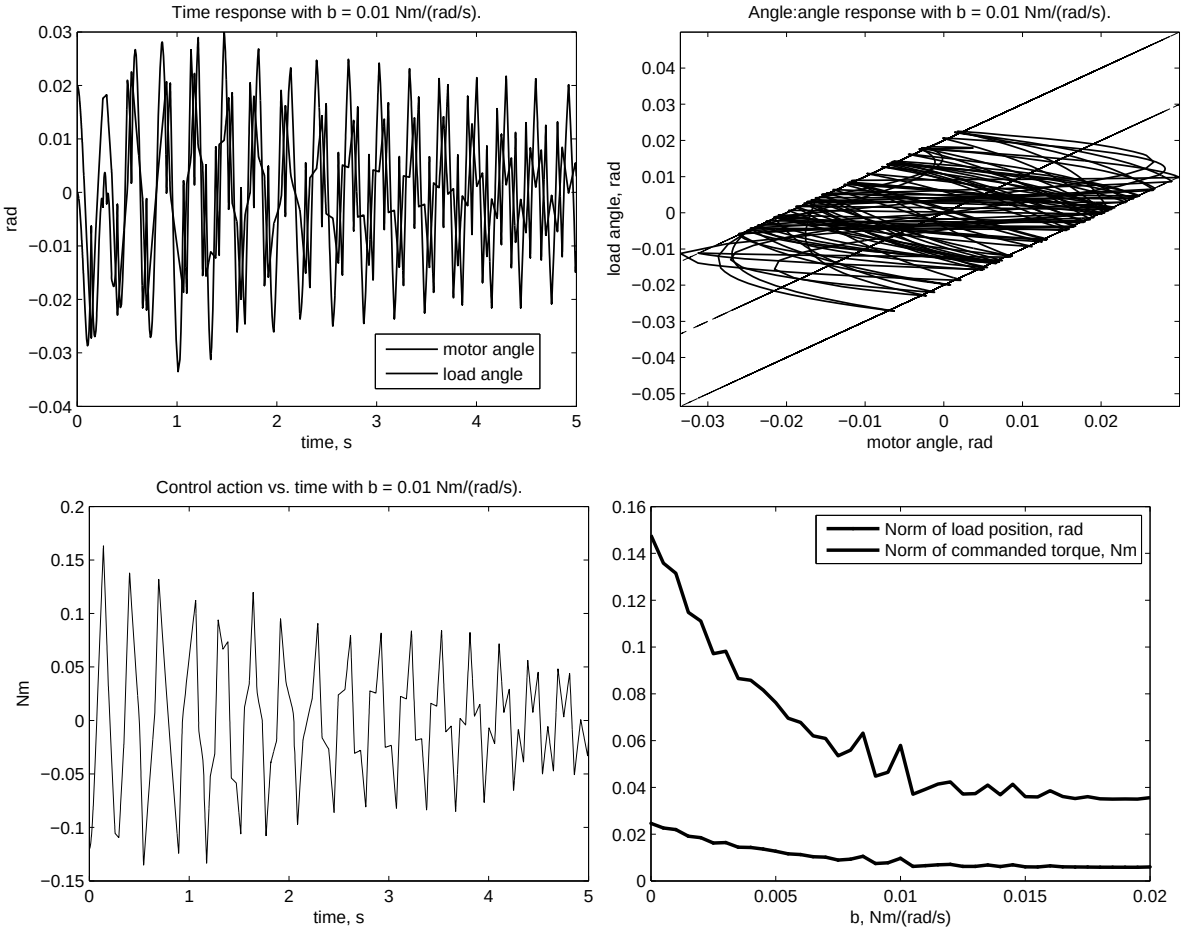
These norms are to be computed over the time range of 2.5-5.0 seconds after the feedback system turns on.

Question: What are the means and standard deviations of the two characteristic outputs defined above?

The attached code models the dynamic behavior, and runs a number of cases along the uniformly-divided range of damping b. I used a trapezoid rule for the integration of the mean, and just computed the straight standard deviation of the samples. It would also have been possible to make these calculations with a Gauss quadrature or a Monte Carlo technique.

A typical response is shown below versus time, as is the norm $||z(t)||_2$ versus b. The mean and standard deviation of the load-side position are 0.0101 and 0.00527 rad, respectively. The mean and standard deviation of the motor torque are 0.0609 and 0.0316 Nm, respectively.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Backlash sensitivity problem

% FSH MIT Mechanical Engineering April 2008

clear all;
global Jm Jl k h g b ;

Jm = 0.003 ; % motor-side inertia, kg-m^2
Jl = 0.01 ; % load-side inertia, kg-m^2
k = 10000 ;   % stiffness of tooth contact point, Nm/rad
h = 0.02 ;   % backlash half-width, rad
g = 6 ; % position feedback gain, Nm/rad

bvec = 0:.0005:.02 ;   % vector of dampings to apply,
                        % on load side, Nm/(rad/s) = kg*m^2/s
```



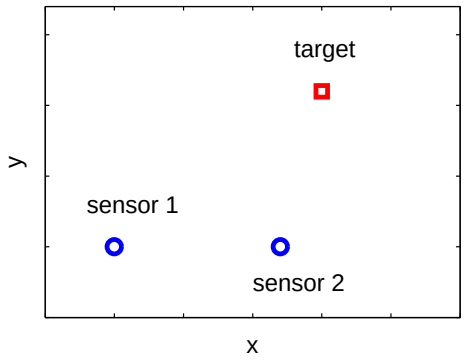

```
tfinal = 5 ; % final simulation time
tcut = 2.5; % time after which we will computing the signal norm
plotFlag = 0 ; % set to one to get all plots of trajectories

ct = 0 ; % counter
for b = bvec, % loop through all the b's

    disp(sprintf('Approx. closed-loop damping ratio:  %g', ...
        b/2/sqrt((Jm+Jl)*g) ));

    % run the simulation - note initial error in load position
    % equal to the backlash half-width (states are listed in deriv.)
    [t,y] = ode45('backlashderiv',tfinal,[0 0 0 h]);

    tauMotor = -g*y(:,4); % recreate the motor torque command

    if plotFlag | b == bvec(end)/2,  % show some intermediate results
```

```
        figure(1);clf;hold off;
        subplot('Position',[.2 .2 .5 .5]);
        plot(t,y(:,[2,4]),'LineWidth',1);
        xlabel('time, s');
        ylabel('rad');
        legend('motor angle', 'load angle',4);
        title(sprintf('Time response with b = %g Nm/(rad/s).', b));

        figure(2);clf;hold off;
        subplot('Position',[.2 .2 .5 .5]);
        plot(y(:,2),y(:,4),'r','LineWidth',1);
        hold on;
        plot(y(:,2),y(:,2),'k--',y(:,2),y(:,2)+h,'k--',...
            y(:,2),y(:,2)-h,'k--');
        axis('tight');
        xlabel('motor angle, rad');
        ylabel('load angle, rad');
        title(sprintf('Angle:angle response with b = %g Nm/(rad/s).', b));

        figure(3);clf;hold off;
        subplot('Position',[.2 .2 .5 .5]);
        plot(t,tauMotor);
        xlabel('time, s');
        ylabel('Nm');
        title(sprintf('Control action with b = %g Nm/(rad/s).', b));

    end;

    ct = ct+1 ;

    % calculate the norm of the load motion and the torque, after tcut
    [dum,i] = sort(abs(t-tcut)) ;
    energyNorm(ct) = sqrt(trapz(t(i(1):end),y(i(1):end,4).^2));
    tauMotorNorm(ct) = sqrt(trapz(t(i(1):end),tauMotor(i(1):end).^2));

    disp(sprintf('%d/%d done.', ct,length(bvec)));

end;

figure(4);clf;hold off;
subplot('Position',[.2 .2 .5 .5]);
plot(bvec,energyNorm,'.-',bvec,tauMotorNorm,'LineWidth',2);
legend('Norm of load position, rad','Norm of commanded torque, Nm');
```

```
xlabel('b, Nm/(rad/s)');

disp(sprintf('Mean and std of norm in load positon:  %g, %g rad' , ...
    mean(energyNorm),std(energyNorm)));
disp(sprintf('Mean and std of norm in command torque:  %g, %g Nm', ...
    mean(tauMotorNorm),std(tauMotorNorm)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Backlash state derivative

function [dxdt] = backlashderiv(t,x);
global Jm Jl k h g b ;

% state vector x is:
% [hub velocity, hub position, gearhead velocity, gearhead position]

tauMotor = -g*x(4) ; % hub torque defined by proportional feedback on
                     % the load position

% taut is torque transmitted by teeth - work this out when there
% is contact, according to the stiffness k

if abs(x(2) - x(4)) <= h, % no teeth torque inside the backlash band
    tauTeeth = 0 ;
elseif x(2) - x(4) > h, % motor driving load in the positive direction
    tauTeeth = k*(x(2) - x(4) - h) ;
elseif x(2) - x(4) < -h, % motor driving load in the negative direction
    tauTeeth = k*(x(2) - x(4) + h) ;
end;

dxdt(1,1) = ( tauMotor - tauTeeth ) / Jm ; % propagate the state vector
dxdt(2,1) = x(1);
dxdt(3,1) = ( tauTeeth - b*x(3) ) / Jl ;
dxdt(4,1) = x(3) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

33 Positioning Using Ranging: 2D Case

1. Two sensors, denoted 1 and 2 and located at different locations in the $x-y$ plane, make a range measurements to a target t. Let the range from Sensor 1 to the target be r_1 and let the range from Sensor 2 to the target be r_2; we call the sensor locations $[x_1, y_1]$ and $[x_2, y_2]$, and the target location $[x_t, y_t]$.



   What are the two equations describing the target's $[x_t, y_t]$ position in the horizontal plane, based on the range measurements from the two devices?

2. Sketch these constraints for the fixed sensor locations $x_1 = 0$, $y_1 = 0$, $x_2 = 1$, and $y_2 = 0$, for two different target locations a) $x_t = 2, y_t = 0$, and b) $x_t = 0.5, y_t = 0.5$. This is a total of four circles to draw.

3. Consider your two constraints for a given target location; there are two unknowns $[x_t, y_t]$, so we ought to be able to solve for them. First, solve for x_t by subtracting the two constraint equations, such that the x_t^2 and y_t^2 terms go away; be sure you take advantage of the fact that $x_1 = y_1 = y_2 = 0$! This will let you derive a clean expression for x_t. Then, put this value for x_t into the constraint equation for Sensor 1, and solve for y_t. There will be two solutions for y_t, because this array can't distinguish which side the target is on.

4. Now we model the whole thing in a program; consider the first target location. You are to calculate the two ranges that perfect sensors would give, perturb each of the perfect ranges by `0.1*randn` (`randn` is a Gaussian random number with zero mean and unity variance) to simulate some sensor noise in a measurement, and then solve the real-world location problem, i.e., estimate x_t and y_t, using the measured ranges.

   The idea is to understand how the noisy range measurements affect the estimate of where the target is. To see this, make a hundred noisy measurements, and plot all the target estimates on an $x-y$ plot that also shows the true target location. *Note that when the noisy measurement makes the square root negative in the equation for y_t, you have to explicitly set $y_t = 0$, because we don't want imaginary results.*

5. Repeat this procedure for the second target location.

6. I find that the second configuration is about four times better than the first, in terms of scatter in the estimated location. Explain in words what is happening with the geometry of your constraint circles and the sensor noise.

1. *The range constraints are equations of circles centered at the sensors:*

$$(x_t - x_1)^2 + (y_t - y_1)^2 = r_1^2$$
$$(x_t - x_2)^2 + (y_t - y_2)^2 = r_2^2$$

2. *The figure below shows all four circles for the constraints.*

3. *Making the suggested manipulations, and with* $x_1 = y_1 = y_2 = 0$*, we get*

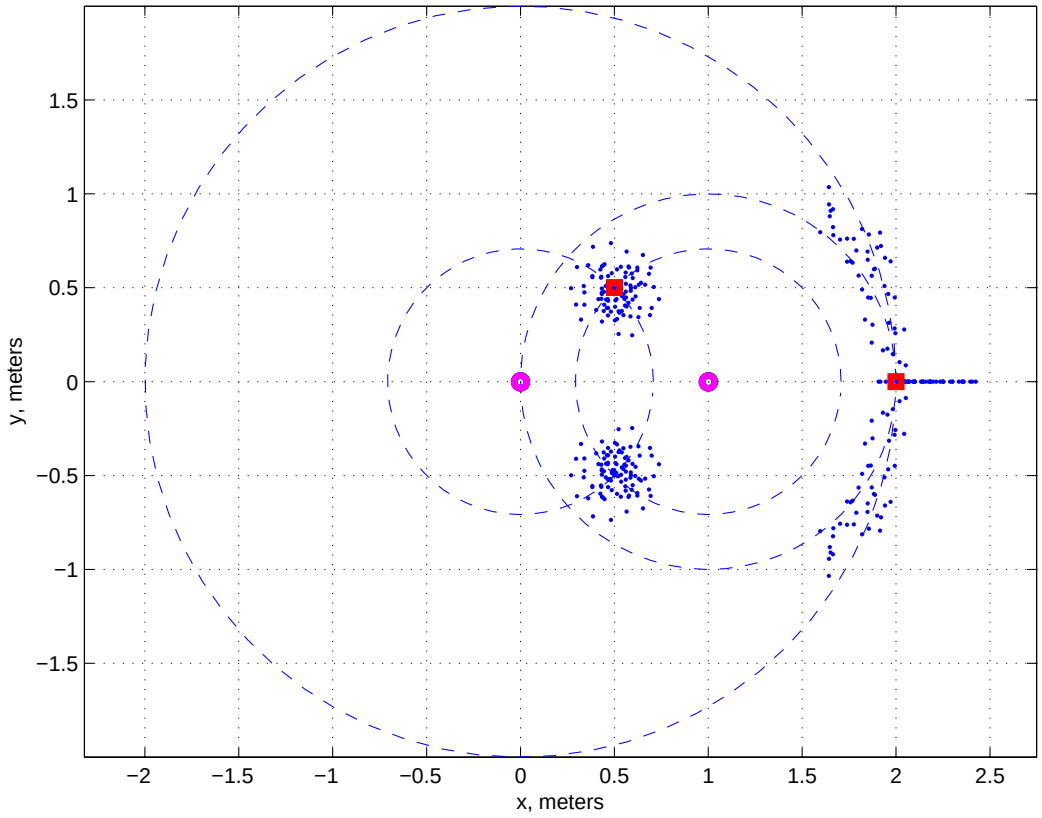
$$\begin{aligned} x_t &= \frac{r_1^2 + x_2^2 - r_2^2}{2x_2} \\ y_t &= \pm\sqrt{r_1^2 - x_t^2} \end{aligned}$$

4. *See the figure.*

5. *Ditto.*

6. *The effect of sensor noise is reduced when the range constraint circles are closer to perpendicular than to tangent. In the former case, the constraints span two directions, which are sufficient to locate the point in the plane; in the latter case, the tangent circles only constrain one direction, namely the radial. The lessons: don't operate near the straight lines that pass through several sensors - the baselines; if you have to, then additional sensors may be needed to get good geometry and a good tight solution. In the figure shown, the standard deviation of error is 0.30 meters for the poor configuration, and 0.064 meters for the good one.*

```
%-----------------------------------------------------------------------
% Analysis of a two-dimension range positioning system

% FSH MIT Mechanical Engineering April 2008

clear all; close all;

% set the positions of the receivers
x1 = 0 ;
y1 = 0 ;
x2 = 1 ;
```




```
y2 = 0 ;

% set the position of the target (both will be done in turn)
xt = [2 0.5] ;
yt = [0 0.5] ;

sigma = .1 ;    % standard deviation of noise to be applied
                % to range measurments

figure(1);clf;hold off;
for j = 1:length(xt),

    plot(x1,y1,'bo',x2,y2,'bo',xt(j),yt(j),'rs','LineWidth',3);
    axis('equal');
    grid on ;
    xlabel('x, meters');
    ylabel('y, meters');

    % make the calculation of ranges from each receiver to target
    r1 = sqrt((xt(j)-x1)^2 + (yt(j)-y1)^2) ;
```

```
    r2 = sqrt((xt(j)-x2)^2 + (yt(j)-y2)^2) ;

    % plot the range circles
    figure(1);hold on;
    cang = cos(0:.1:2*pi) ;
    sang = sin(0:.1:2*pi);
    plot(r1*cang+x1,r1*sang+y1,'--');
    plot(r2*cang+x2,r2*sang+y2,'--');

    % enter the loop to study sensitivity to noise and geometry
    for i = 1:100,

        % add random noise to make range measurements
        r1m = r1 + sigma*randn ;
        r2m = r2 + sigma*randn ;

        % do the estimations of xt and yt from these ranges
        xte = (r1m^2 + x2^2 - r2m^2) / 2 / x2 ;
        % (only do the square root if the argument is positive...
        if (r1m^2 - xte^2) > 0,
            yte(1) = sqrt(r1m^2 - xte^2) ;
            yte(2) = -sqrt(r1m^2 - xte^2) ;
        % ... otherwise we have to just guess
        else,
            yte(1) = 0 ;
            yte(2) = 0 ;
        end;

        figure(1); hold on;
        plot([xte xte],yte,'.');

        % calculate an error - the distance between true and estimated
        err(i) = sqrt((xte-xt(j))^2 + (yte(1)-yt(j))^2);
    end;

    % (make this plot again in case the scatter points cover up)
    plot(x1,y1,'mo',x2,y2,'mo',xt(j),yt(j),'rs','LineWidth',3);

    disp(sprintf('For %d samples, RMS error is:  %g meters.', i, ...
        std(err)));

end;

print -depsc acousticRangingIn2DSolution.eps
```

```
% make a simple diagram for instruction

figure(2);clf;hold off;
subplot('Position',[.2 .2 .3 .3]);
plot(0,0,'bo',1.2,0,'bo',1.5,1.1,'rs','LineWidth',2);
text(-.2,.3,'sensor 1');
text(1,-.25,'sensor 2');
text(1.3,1.4,'target');
set(gca,'XTickLabel',[]);
set(gca,'YTickLabel',[]);
axis([-.5 2.5 -.5 1.7]);
xlabel('x');
ylabel('y');

print -depsc acousticRangingIn2D.eps

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

34 Dead-Reckoning Error

An unmanned, untethered underwater vehicle operating near large metallic marine structures has to navigate without a magnetic compass, and in some cases without any acoustic ranging available. In these cases, a viable approach is dead-reckoning based on the yaw gyro and on the body-referenced velocity over ground, from a Doppler acoustic sensor. Specifically, if θ is yaw angle, U is forward speed, and $[x, y]$ is the position in a global frame of reference, dead-reckoning entails propagating

$$\dot{x} = U \cos\theta$$
$$\dot{y} = U \sin\theta.$$

Let us consider a yaw rate gyro with a noise variance of $0.0025 rad^2/s^2$; it is Gaussian noise. The speed sensor is also subject to Gaussian noise, but this has variance $0.0001 m^2/s^2$. The yaw rate sensor is measuring $r = \dot{\theta}$ and the speed sensor is measuring U.

The platform on which these sensors is mounted goes through a *known* circular trajectory, with $U = 1.5 m/s$ and $r = \pi/150\ rad/s$, and the sensors are sampled five times per second.

Question: Assuming that at time zero, there is no error in x, y, θ, what are the mean and standard deviation of the x, y, and range ($\sqrt{x^2 + y^2}$) errors at the end of one complete circle? Please also confirm that your errors are small with zero sensor noise - this should easily be several meters or less, even if you use a basic forward Euler integration rule.

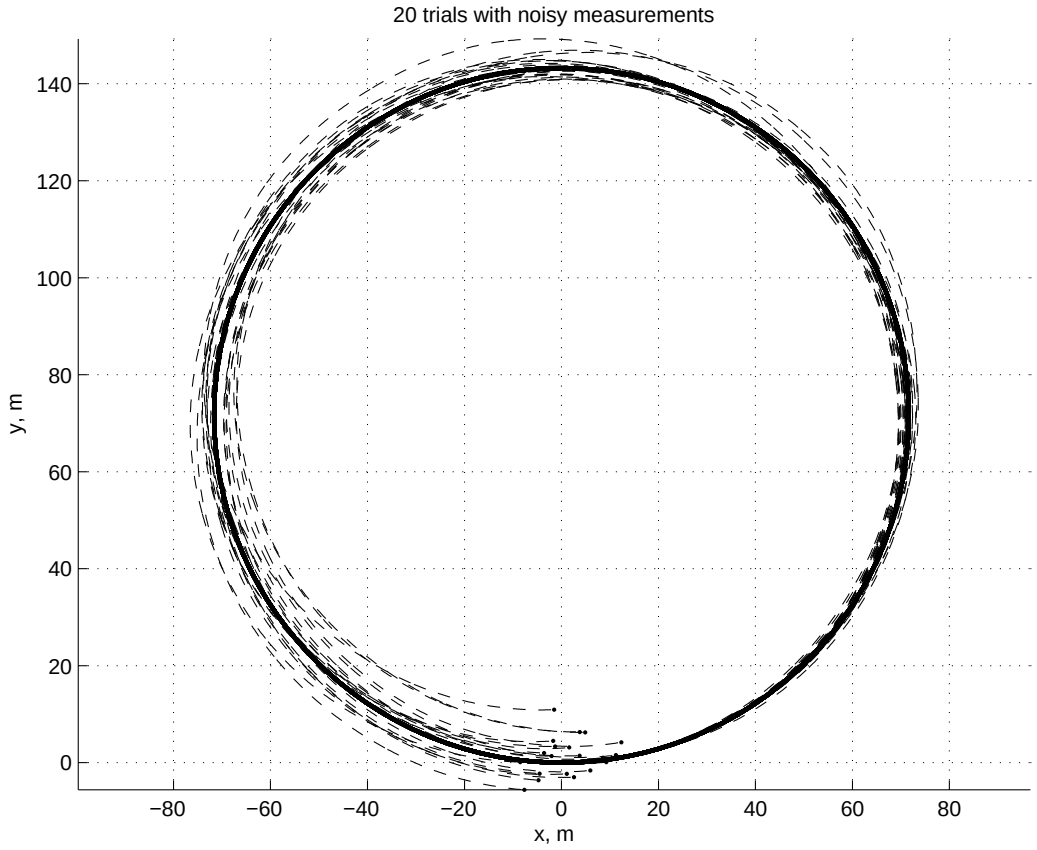
As indicated in the attached code, this is a Monte Carlo problem, where we inject a random number to go with with every noisy measurement. The figure below shows twenty realizations, and the scatter in the endpoints. You see that the paths get progressively worse as we move around the circle and more erroneous measurements are used in the integrations.

For 10,000 realizations, statistics computed are:

x-error:	*-0.03 m mean*	*6.81 m standard deviation*
y-error:	*0.15 m mean*	*3.88 m standard deviation*
range error:	*6.81 m mean*	*3.89 m standard deviation.*

The mean x and y errors should be close to zero, but there are not enough realizations to show this clearly.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dead reckoning.

% FSH MIT Mechanical Engineering April 2008

clear all;
clc;
```




```
dt = 0.2 ; % time step, s
U = 1.5 ; % speed, m/s
r = 2*pi/300 ; % yaw rate, rad/s
Uvar = 0.0025 ; % speed sensor variance, (m/s)^2
Rvar = 0.0001 ; % yaw rate variance (rad/s)^2
N = 20 ;    % number of trials (a small number if showing plots,
            % or a big number if you want good statistics)
plotFlag = 1 ; % set to one to show plots

T = 2*pi/r ; % time to complete an entire circle

figure(1);clf;hold off;
figure(2);clf;hold off;

tic ;
for ii = 1:N, % loop through the number of realizations

    % initialize the various states - first time step
```

```
    x(1) = 0 ;  % x-position
    y(1) = 0 ; % y-position
    th(1) = 0 ; % heading
    xp(1) = 0 ; % "pure" x-position : created without sensor noise
    yp(1) = 0 ; % "pure" y-position
    thp(1) = 0 ; % "pure" heading

    rMeasure(1) = sqrt(Rvar)*randn + r ; % first measurement of r
    UMeasure(1) = sqrt(Uvar)*randn + U ; % first measurement of U

    ct = 1 ;
    for tt = dt:dt:T,   % step through all the times needed to complete
        % a circle

        ct = ct + 1 ;

        UMeasure(ct) = sqrt(Uvar)*randn + U ; % get measurements
        rMeasure(ct) = sqrt(Rvar)*randn + r ;

        % propagate the states based on the noisy measurements
        % (using a one-shot modified Euler step)

%        th(ct) = th(ct-1) + dt*rMeasure(ct)/2 + dt*rMeasure(ct-1)/2 ;

%        x(ct) = x(ct-1) + dt*UMeasure(ct)*cos(th(ct))/2 + ...
%            dt*UMeasure(ct-1)*cos(th(ct-1))/2;

%        y(ct) = y(ct-1) + dt*UMeasure(ct)*sin(th(ct))/2 + ...
%            dt*UMeasure(ct-1)*sin(th(ct-1))/2;

        % a regular forward Euler - testing testing!!!*******
        % this is what most of the students did S08
        th(ct) = th(ct-1) + dt*rMeasure(ct-1) ;
        x(ct) = x(ct-1) + ...
            dt*UMeasure(ct-1)*cos(th(ct-1));
        y(ct) = y(ct-1) + ...
            dt*UMeasure(ct-1)*sin(th(ct-1));


        % for the first realization, compute also a "pure"
        % trajectory, i.e., without sensor noise - this will
        % confirm that our integration method is OK.
        if ii == 1,
            thp(ct) = thp(ct-1) + dt*r ;
```

```
            xp(ct) = xp(ct-1) + dt*U*cos(thp(ct))/2 + ...
                dt*U*cos(thp(ct-1))/2;

            yp(ct) = yp(ct-1) + dt*U*sin(thp(ct))/2 + ...
                dt*U*sin(thp(ct-1))/2;
        end;

    end;

    tvec = 0:dt:dt*(length(x)-1) ;

    % collect the errors for each realization
    xerr(ii) = x(end);
    yerr(ii) = y(end);
    rerr(ii) = sqrt(x(end)^2 + y(end)^2);

    % here are the errors with the "pure" trajectory
    xperr = xp(end);
    yperr = yp(end) ;
    rperr = sqrt(xperr(end)^2 + yperr(end)^2);

    if plotFlag,
        figure(1);hold on;
        plot(tvec,x,tvec,y,tvec,th*20) ;
        legend('x, m','y, m','\theta*20, rad/s',3);
        xlabel('sec');

        figure(2);hold on;
        plot(x,y,'--',x(end),y(end),'k.');
        plot(xp,yp,'r','LineWidth',2);
        axis('equal');
        grid on;
        xlabel('x, m');
        ylabel('y, m');
        title(sprintf('%d trials with noisy measurements',N));

        disp(sprintf('ii:  %d     Range Error:  %g m.', ii, rerr(ii)));

    end;

end;
disp(sprintf('Elapsed time: %g sec.', toc));
```

```
disp(sprintf('[xerr yerr rangeErr] with NO NOISE:  %g %g %g m.',...
    xperr,yperr,rperr));
disp(sprintf('Trials:  %d', length(xerr)));
disp(sprintf('Mean of [xerr yerr rangeErr]:  %g %g %g m.',mean(xerr),...
    mean(yerr), mean(rerr)));
disp(sprintf('Stddev of [xerr yerr rangeErr]:  %g %g %g m.',std(xerr),...
    std(yerr), std(rerr)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

35 Landing Vehicle Control

A controlled vehicle that is landing onto a target in the horizontal plane is equipped with a vision-based navigation system. The system gives very good resolution at low altitudes, but, unfortunately, poor resolution at high altitudes. This effect is modeled as $\sigma_\nu^2 = 0.05z^2$, where z is the altitude and σ_ν^2 is the variance of the noise ν in the measurement $y = x + \nu$; x is the true horizontal position of the craft. The sensor noise ν is considered to be of zero mean, and to have Gaussian distribution, at any particular altitude. There are no other effects, say due to roll, pitch, or yaw, in this sensor.

We consider the horizontal positioning problem in one direction only, e.g., North-South. The vehicle gets a measurement y once per second, and descends at a steady rate of $2m/s$. A controller applies a corrective thrust force according to $T = -k \times sign(y)$, and the vehicle physically responds to thrust with a velocity change: $\ddot{x} = T$. The vehicle physical behavior is also affected by a horizontal drift disturbance (due to current or winds). This is steady in time and constant over all altitudes, but its magnitude on any particular deployment can take a random value between -0.2 m/s and 0.2 m/s, uniformly distributed.

1. Is the sensor noise process either stationary or ergodic? Explain.

   Summary: The sensor noise is not stationary, because its statistics (variance in particular) are not constant through time. The sensor noise is worst at high altitudes and best at low altitudes. Ergodicity implies that the time statistics and the ensemble statistics are the same - in this case, clearly the ensemble variance at any given time cannot equal the variance over time of any particular realization. Hence the noise process is not ergodic either.

2. For an initial height of $z = 200m$, no initial error or horizontal velocity (i.e., $x(t = 0) = \dot{x}(t = 0) = 0$), and gain $k = 0.1$ what is the mean position error upon landing?

   Solution: The mean error is zero, as should be expected since everything is symmetric on the positive and negative x-axes. This is confirmed in Monte Carlo simulations; see the attached histogram with gain 0.1, and 10,000 trials.

3. Under these conditions, what is the standard deviation of the position error upon landing?

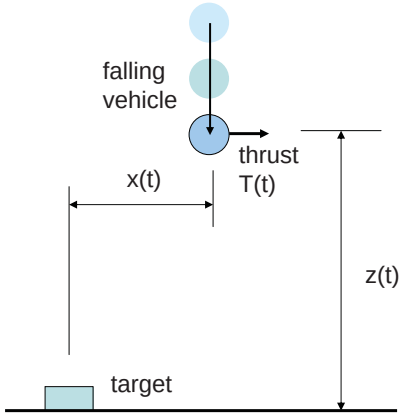
   Solution: On three "experiments" with 10,000 trials, I get [10.5, 10.7, 10.4]m as the answer - a good average is 10.5m.

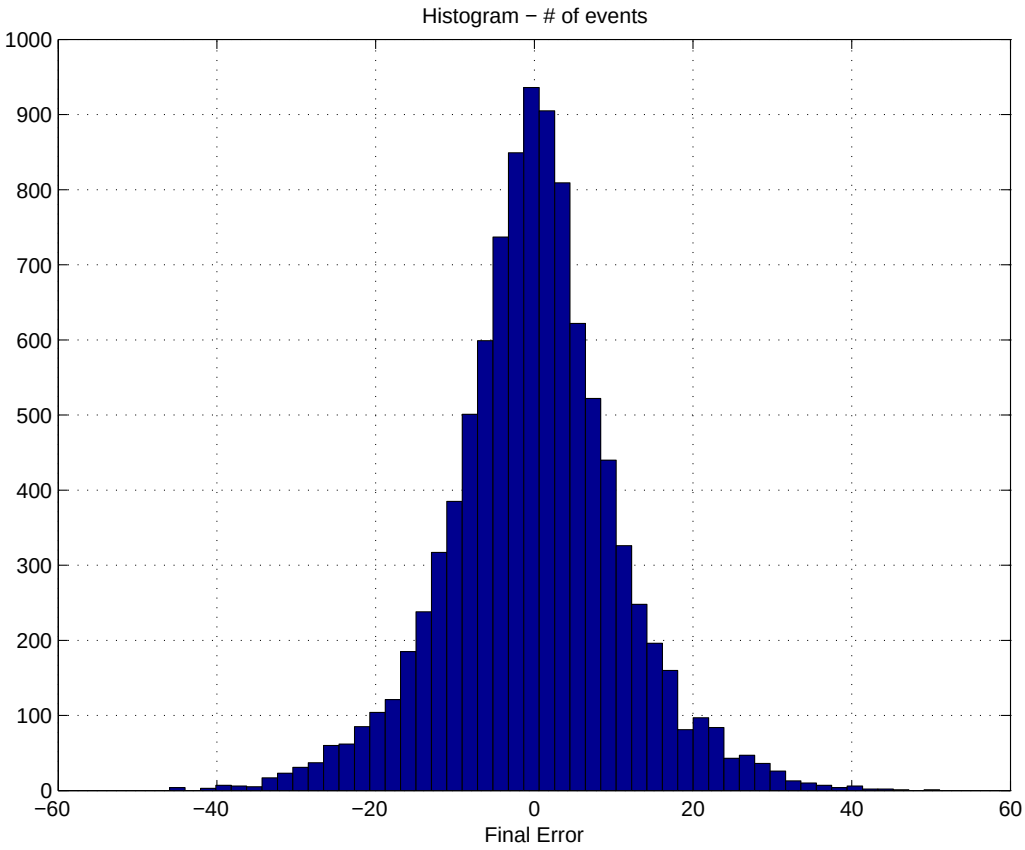
   The time traces below give the sensor noise (top subplot), the vehicle x-position (middle subplot), and the vehicle x-velocity (bottom subplot). We see that the vehicle spends a lot of effort responding to the very large sensor noise at altitude and does not really "home in" on the target effectively.

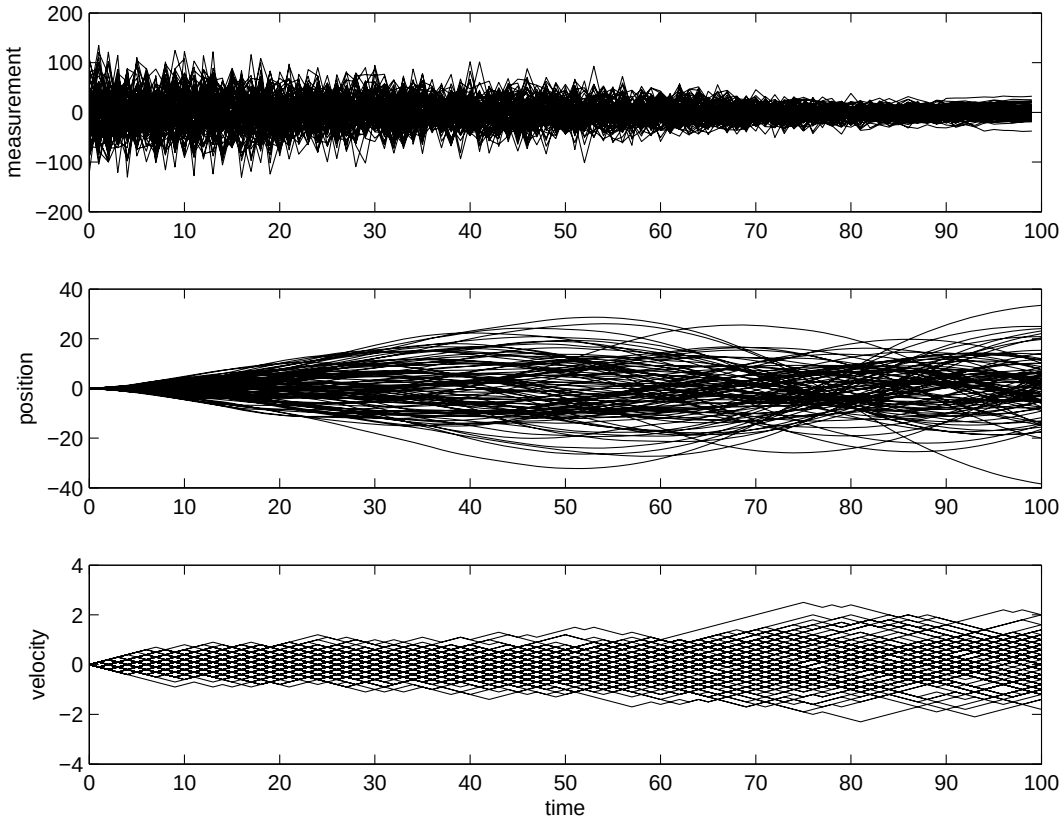
4. Can you pick another k that gives better performance?

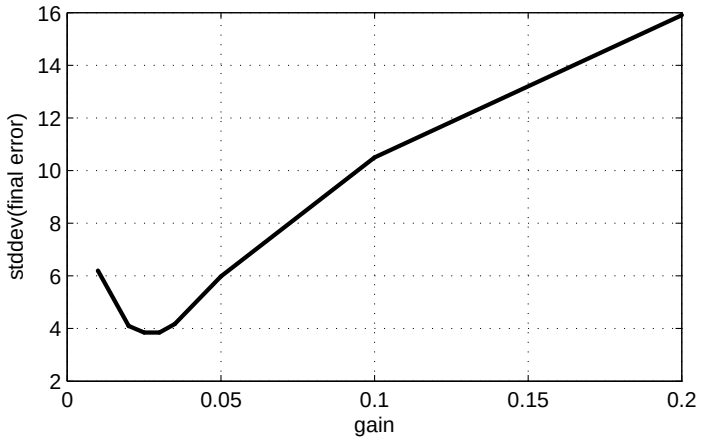
   Solution: The last plot gives the error standard deviation for some additional values of the gain. We see that a gain of 0.027 is quite good, with a standard deviation of

about 3.85m. Note that because the vehicle has zero initial error, it is the drift that creates the initial perturbation.










```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Falling Vehicle navigation & control simulation

clear all;
```

```
n = 100 ; % number of time steps
dt = 1 ; % time step
zdot = 2 ; % falling velocity
zInitial = 200 ; % initial elevation
N = 10000 ; % number of trials
gain = input('What is the gain? ');

figure(1);clf;hold off;
for i = 1:N,  % do many trials

    x(1) = 0 ; % set initial conditions
    xdot(1) = 0 ;

    wind = (rand-.5)*2*.2 ; % get the steady wind for this trial

    for j = 0:n-1,  % time index

        z = zInitial - zdot*j*dt ; % altitude at each time instant

        y(j+1) = x(j+1) + sqrt(.05)*z*randn ; % measurement

        % propagate the vehicle state
        xdot(j+2) = xdot(j+1) - gain*sign(y(j+1))*dt ;
        x(j+2) = x(j+1) + (xdot(j+1)+xdot(j+2))/2*dt + wind*dt ;

    end;

    if N <= 100, % make a few plots if a small # of trials
        subplot(311); plot([0:n-1],y); hold on;
        ylabel('measurement');
        subplot(312); plot([0:n]*dt,x); hold on ;
        ylabel('position');
        subplot(313); plot([0:n]*dt,xdot); hold on;
        ylabel('velocity');
        xlabel('time');
    end;

    finalError(i) = x(end) ; % store the final error
end;

figure(2);clf;hold off;
hist(finalError,50);
title('Histogram - # of events');
xlabel('Final Error');
```

```
disp(sprintf('gain:  %g.', gain)) ;
disp(sprintf('Mean error:  %g m.', mean(finalError)));
disp(sprintf('Error stddev:  %g m.', std(finalError)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

36 Control of a High-Speed Vehicle

An instability in certain aircraft and in some high-speed marine vehicles is characterized by a pair of complex right-half plane poles, and is due to inadequate aerodynamic stability. It is particularly pronounced in highly maneuverable craft, where open-loop stability in the physical design has been intentionally traded off against agility. In this problem, you will use the Nyquist criterion to bring such a vehicle under control and to achieve specific performance and robustness properties.

1. The transfer function taking the elevator control surface command into the pitch of the vehicle is given as

$$P(s) = \frac{y(s)}{u(s)} = \frac{2(s+2)}{s^2 - 0.2s + 16}$$

   Confirm that this is an unstable system by both stating the poles and by plotting the impulse response.

2. Plot the complex loci for this plant. This means plotting the real versus the imaginary parts of $P(s)$, $s = j\omega$, for ω going from zero to infinity. Then plot this same curve again, negating the imaginary part, and you then have covered the range of ω from $[-\infty, \infty]$. Use logarithmically-spaced frequencies.

3. Applying the Nyquist criterion to this plot, can the vehicle be stabilized with unity feedback gain, i.e., $C(s) = 1$? If so, comment on whether it is a useful feedback system.

4. We need to achieve through feedback a) integral action, b) a phase margin of 30-40 degrees, and c) a gain margin of at least two. To do this, design a controller $C(s)$ with the following properties:

   - a real pole just to the left of the origin, corresponding with an integrator (we want it to be definitely *not* in the RHP);
   - a stable complex zero pair that approximately mirrors the unstable plant poles across the imaginary axis, but possibly with a higher damping ratio;
   - two additional stable, real poles so that the compensator has a high-frequency roll-off behavior.

   State the complete transfer function of your designed compensator, show that it achieves the desired gain and phase margins on a plot of $P(s)C(s)$ loci, and show a closed-loop step response to prove that it is indeed well-behaved.

1. *The plant poles are at $s = 0.1 \pm 4j$. Since they are in the right-half plane, the system is unstable. The open-loop response shows an oscillation of growing amplitude.*

2. *See the figures with loci of $P(s)$ and magnitude $|P(s)|$.*

3. *$|P(s)|$ goes to zero at high frequencies, and takes a small but definitely nonzero value for low frequencies. Thus, sweeping s from negative to positive infinity and around the RHP leads to two counterclockwise encirclements of the critical point $1 + 0j$. This is matched by the two unstable poles in the plant, i.e., we do have $P = CCW$, and the system is stable with unity feedback!*

   It is not a very good design, however, for at least one reason: the magnitude of $P(s)C(s)$ is small at low frequencies, and hence the system will track a reference signal very poorly. A simulation quickly verifies this fact.

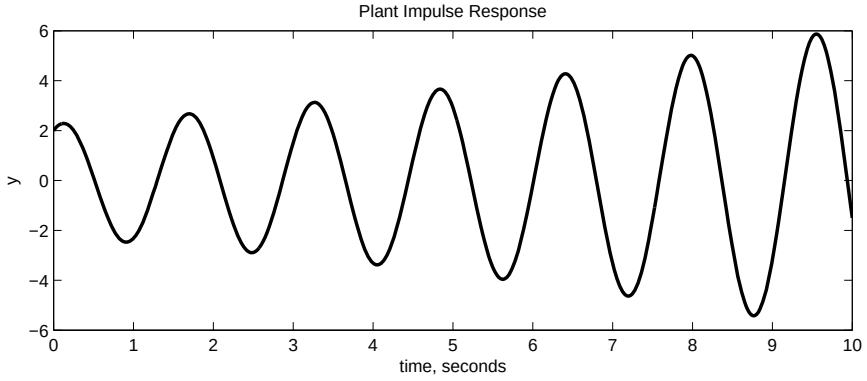
4. *A feasible compensator is:*

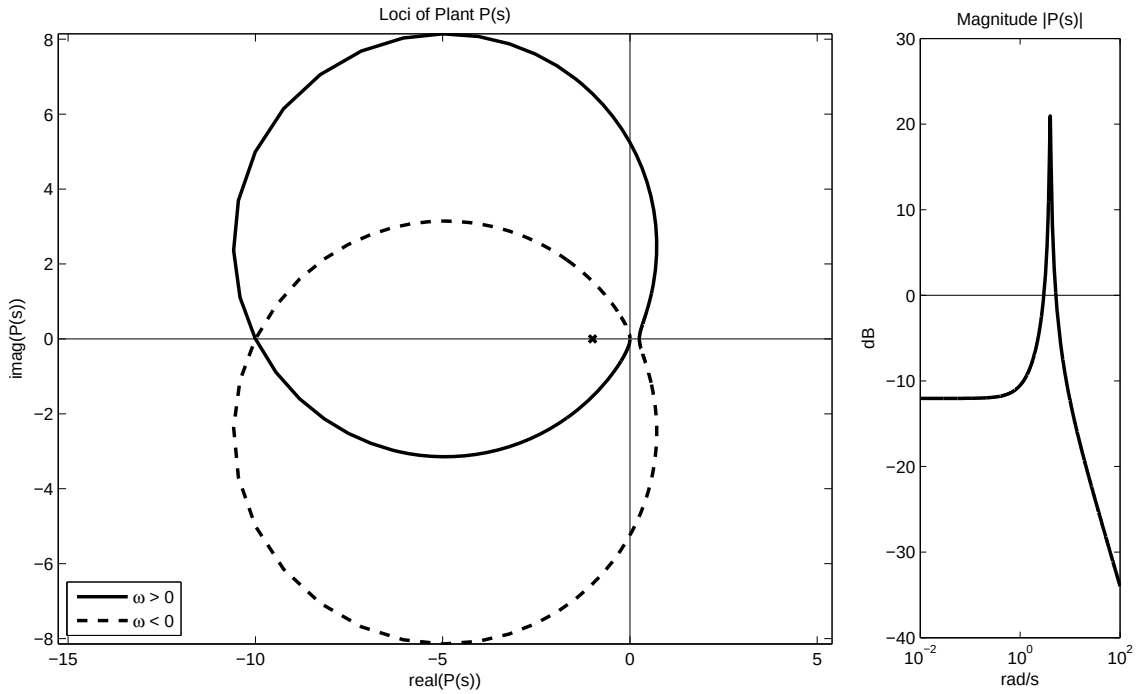
$$C(s) = \frac{u(s)}{e(s)} = \frac{30(s^2 + 0.4s + 16.04)}{(s + 0.0001)(s + 4)^2}.$$

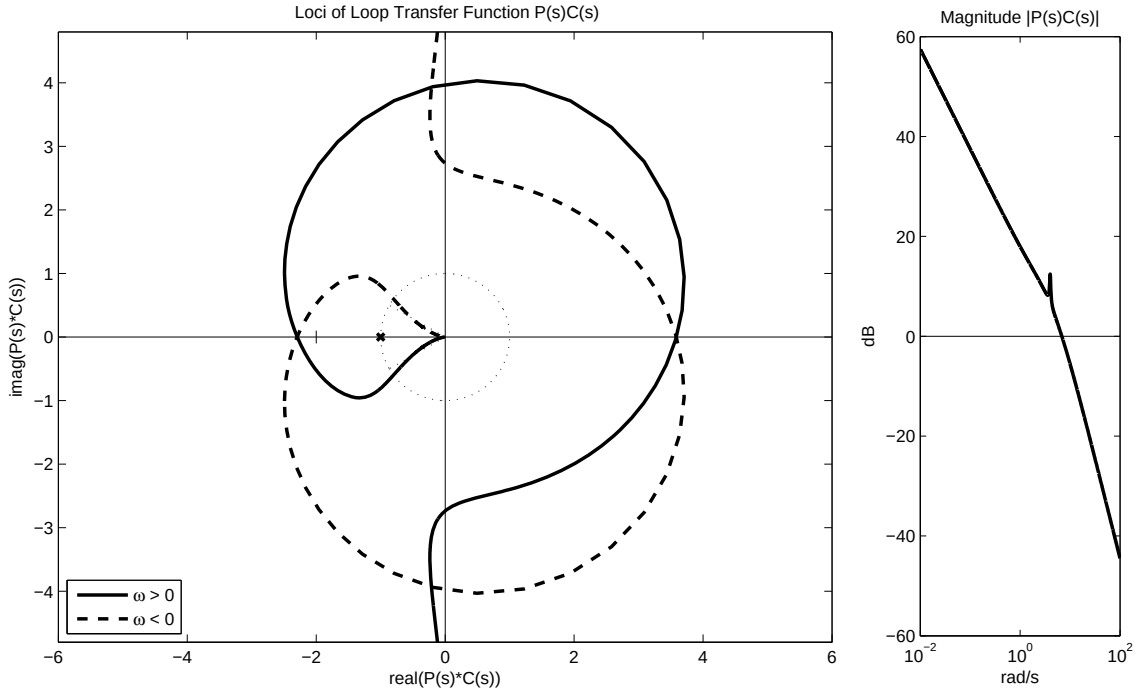
   As suggested in the problem statement, this solution has a pole very close to the origin but definitely stable, two stable zeros in a mirror configuration plus a little more damping, and two real poles. The mirror configuration zeros provide a clever way to deal with nasty unstable plant poles; the pair roughly cancels the $|P(s)|$ peak. This compensator has quite high gain below about four radians per second, but rolls off to zero at high frequencies.

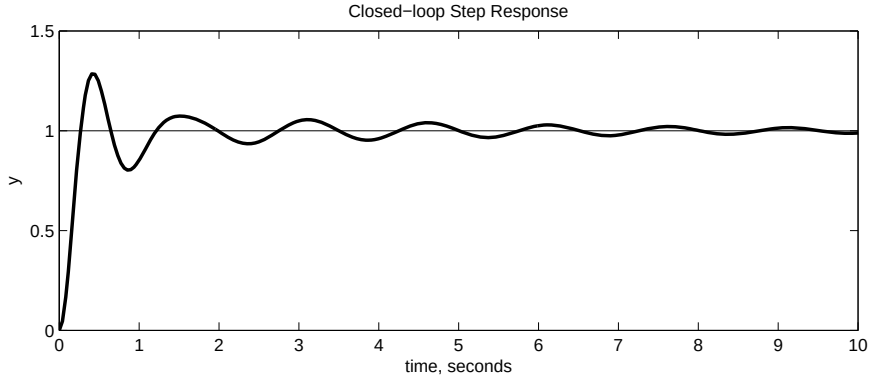
   Looking at the loci of the loop transfer function $|P(s)C(s)|$, we see immediately two CCW encirclements of the critical point, with finite magnitudes. The transition through zero frequency (from small negative to small positive), however, occurs at very large magnitude of $P(s)C(s)$ and is approached at ± 90 degrees. We have to ask which way do these lines connect, in the right- or left-half plane? Since the path of s encloses the entire RHP, this path must include a positive, real value - putting a positive real value for s into the loop transfer function shows that the completing portion of the $P(s)C(s)$ loci occurs in the RHP. We are then able to conclude that this loop does not encircle the critical point. The remaining two CCW encirclements match the unstable poles of the plant, $P = CCW$ as before, and the system is stable.

   The figures show that this design satisfies the gain and phase margin requirements, and that the closed-loop response is acceptable in terms of steady-state error and ringing.




Loci of Plant P(s)
imag(P(s))
real(P(s))
ω > 0
ω < 0
Magnitude |P(s)|
dB
rad/s


Loci of Loop Transfer Function P(s)C(s)
imag(P(s)*C(s))
real(P(s)*C(s))
ω > 0
ω < 0
Magnitude |P(s)C(s)|
dB
rad/s


Closed−loop Step Response
y
time, seconds

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Design of a flight controller using Nyquist plot.

% FSH MIT Mechanical Engineering April 2008

clear all;

zP = [-2] ; % zeros in plant
pP = [0.1+4*sqrt(-1), 0.1-4*sqrt(-1)] ; % poles in plant
kP = 2 ; % gain of plant

zC = [-0.2+4*sqrt(-1), -0.2-4*sqrt(-1)] ; % zeros in controller
pC = [-4 -4 -0.0001] ; % poles in controller
kC = 30 ; % gain of controller

w = logspace(-2,2,3000); % (a very dense) frequency vector

% compute and plot the plant impulse response
sysP = zpk(zP,pP,kP);
[y,t] = impulse(sysP,10);
        % (could also solve the ode explicitly, with a tall
        % rectangle or triangle approximation for the impulse)
figure(1);clf;hold off;
subplot('Position',[.1 .2 .8 .4]);
plot(t,y,'k','LineWidth',2);
xlabel('time, seconds');
ylabel('y');
title('Plant Impulse Response');
print -deps flightControl1.eps ;

% make up the loci of P(s)
% (could also do this in one line with freqresp() !)
for i = 1:length(w),
    P(i) = kP ;
    s = sqrt(-1)*w(i) ;
    for j = 1:length(zP),
        P(i) = P(i) * (s - zP(j)) ;
    end;
    for j = 1:length(pP),
        P(i) = P(i) / (s - pP(j)) ;
    end;
end;

% plot the loci of P(s)
```

```
figure(2);clf;hold off;
plot(real(P),imag(P),'k',real(P),-imag(P),'k--','LineWidth',2);
axis('equal');
a = axis ;
hold on;
title('Loci of Plant P(s)');
legend('\omega > 0','\omega < 0',3);
plot([0 0],[a(3) a(4)],'k',[a(1) a(2)],[0 0],'k');
plot(-1,0,'kx','LineWidth',2);
xlabel('real(P(s))');
ylabel('imag(P(s))');
print -deps flightControl2.eps ;

% make up the loci of C(s)
for i = 1:length(w),
    C(i) = kC ;
    s = sqrt(-1)*w(i) ;
    for j = 1:length(zC),
        C(i) = C(i) * (s - zC(j)) ;
    end;
    for j = 1:length(pC),
        C(i) = C(i) / (s - pC(j)) ;
    end;
end;

% plot the loci of P(s)*C(s)
figure(3); clf;hold off;
plot(real(P.*C),imag(P.*C),'k-',real(P.*C),-imag(P.*C),'k--',...
    -1,0,'bx','LineWidth',2);
hold on;
legend('\omega > 0','\omega < 0',3);
axis([-5 5 -4 4]*1.2);
a = axis ;
plot(sin(.0:.01:2*pi),cos(0:.01:2*pi),'k:');
plot([-2,-2],[-.1 .1],'k-');
plot([0 0],[a(3) a(4)],'k',[a(1) a(2)],[0 0],'k');
plot([0 -sqrt(3)/2]*1.,[0 1/2]*1.,'k:',...
        [0 -sqrt(3)/2]*1.,[0 -1/2]*1.,'k:');
xlabel('real(P(s)*C(s))');
ylabel('imag(P(s)*C(s))');
title('Loci of Loop Transfer Function P(s)C(s)');
print -deps flightControl3.eps ;

% plot the magnitude of PC
```

```
figure(4);clf;hold off;
subplot('Position',[.2 .1 .2 .8]);
semilogx(w,20*log10(abs(P.*C)),'k','LineWidth',2);
hold on;
semilogx([min(w),max(w)],[0 0],'k-');
xlabel('rad/s');
ylabel('dB');
title('Magnitude |P(s)C(s)|');
print -deps flightControl4.eps ;

% compute and plot the closed-loop step response
sysC = zpk(zC,pC,kC);
[y,t] = step(feedback(sysP*sysC,1),10);
    % could also do an ode to solve this

figure(5);clf;hold off;
subplot('Position',[.1 .2 .8 .4]);
plot(t,y,'k','LineWidth',2);
hold on;
plot([0 10],[1 1],'k-');
xlabel('time, seconds');
ylabel('y');
title('Closed-loop Step Response');
a = axis ; axis([0 10 a(3:4)]);
print -deps flightControl5.eps ;

% plot the magnitude of P - just for good measure
figure(6);clf;hold off;
subplot('Position',[.2 .1 .2 .8]);
semilogx(w,20*log10(abs(P)),'k','LineWidth',2);
hold on;
semilogx([min(w),max(w)],[0 0],'k-');
xlabel('rad/s');
ylabel('dB');
title('Magnitude |P(s)|');
print -deps flightControl6.eps ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

37 Nyquist Plot

Consider the attached images; here are a few notes. In the plant impulse response, the initial condition before the impulse is zero. The frequency scale in the transfer function magnitude plots is $10^{-3} - 10^4$ radians per second. In the plot of $P(s)$ loci, the paths taken approach the origin from $\pm 90°$, and do not come close to the critical point at $-1+0j$, which is shown with an `x`. In the plot of $P(s)C(s)$ loci, the unit circle and some thirty-degree lines are shown with dots. Also, the two paths in this plot connect off the page in the right-half plane.

Answer the following questions by circling the correct answer.

1. The overshoot evident in the open-loop plant is about
   (a) 120%
   (b) there is no overshoot since this is not a step response
   (c) **70**%
   (d) 40%
2. The natural frequency in the open-loop plant is about
   (a) one Hertz
   (b) **one radian per second** - To compute this, you need a whole cycle.
   (c) 1.2 radians per second
   (d) six radians per second
3. Based on the plant behavior, $P(s)$ probably has
   (a) no zeros and one pole
   (b) one zero and one pole
   (c) no zeros and two poles
   (d) **one zero and two poles** - This plant has a zero at $+1$ (yes, a right-half plane zero, also known as an unstable zero) and two poles at $-0.1 \pm j$. You can tell it has two complex,stable poles because of the ringing in this impulse response. You can tell it has a zero because the output instantaneously moves to a nonzero value during the impulse - this could only be caused by a differentiator.
4. Compare the abilities of the plant hooked up in a unity feedback loop (i.e., with $C(s) = 1$), and of the designed closed-loop system, to follow low-frequency commands:
   (a) The $P(s)C(s)$ case has a lot more magnitude above one radian per second, and so it has a better command-following
   (b) $P(s)$ is nice and flat at low frequencies, so it is better at command-following

(c) **P(s)C(s) has increasing values at lower frequencies and this makes it better** - Setting $C(s) = 1$ will achieve about 10% tracking accuracy at low frequencies. The designed $P(s)C(s)$ has a pole at or near the origin and hence is an integrator; this gives us no tracking error in the steady state.

The plant is stable, but lightly damped and it has an unstable zero. As you can guess from a quick check with a root locus, this is a difficult control problem, intuitively because the plant always moves in the wrong direction first. A PID cannot stabilize this system! I ended up using the loopshaping method in MATLAB's LTI design tool; this gave a third-order controller with two zeros.

(d) The peak in $P(s)$ is not shared by the other plot and this makes $P(s)$ better at command-following.

5. Is the unity feedback loop stable, based on the loci of $P(s)$?

   (a) **No: The path encircles the critical point once in the clockwise direction and that is all it takes, because the poles of $P(s)C(s)$ are in the left-half plane** - Note that the unstable zero in the plant is immaterial by itself. Nyquist's rule is that stability is achieved if and only if $p = ccw$, where p is the number of unstable poles in $P(s)C(s)$, and ccw is the number of counter-clockwise encirclements of the critical point.

   (b) Yes: The path encircles the critical point once in the counter-clockwise direction

   (c) Yes: The path encircles the critical point once clockwise and this is matched by a plant zero in the right-half plane

   (d) No: The path encircles the critical point twice whereas it should only circle it once.

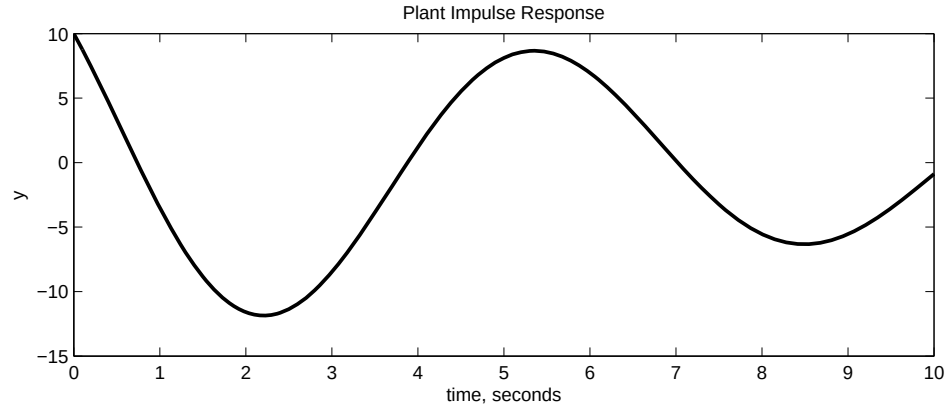
6. The designed compensator creates a stable closed-loop system, as is seen in the step response plot. The gain and phase margins achieved are approximately:

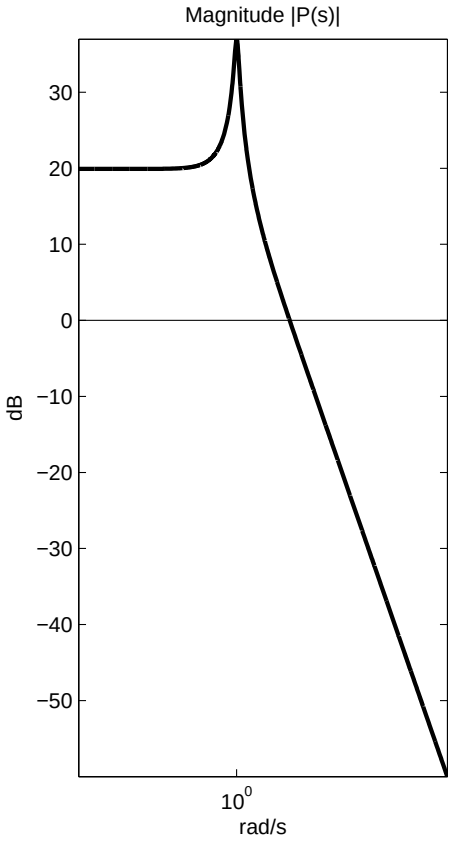
   (a) 0.2 upward gain margin, 1.2 downward gain margin, and $\pm 40^\circ$ phase margin

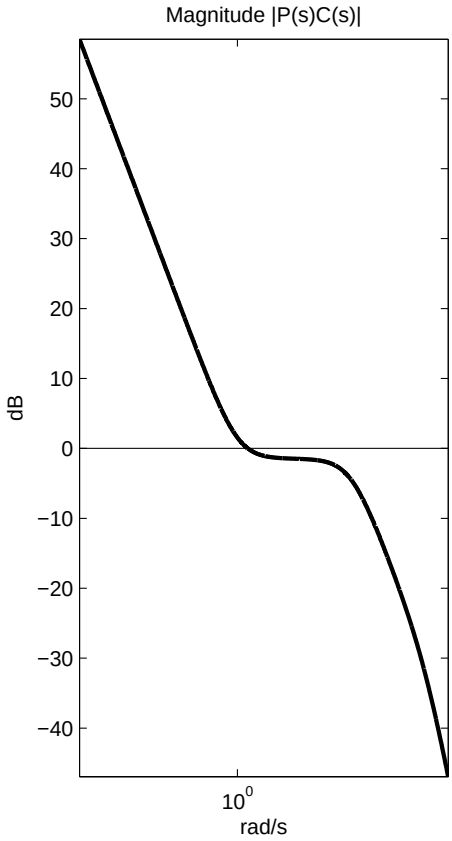
   (b) **1.2 upward gain margin, infinite downward gain margin, and $\pm 30^\circ$ phase margin**

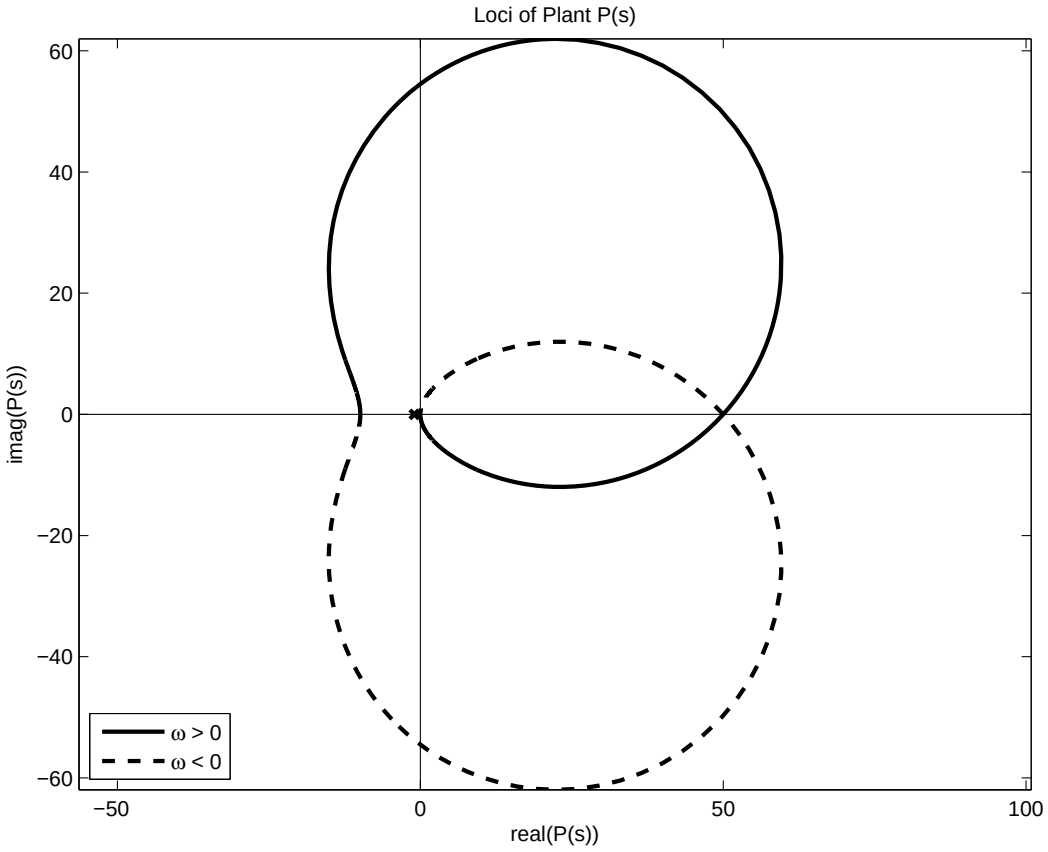
   (c) infinite upward gain margin, 1.2 downward gain margin, and $\pm 30^\circ$ phase margin

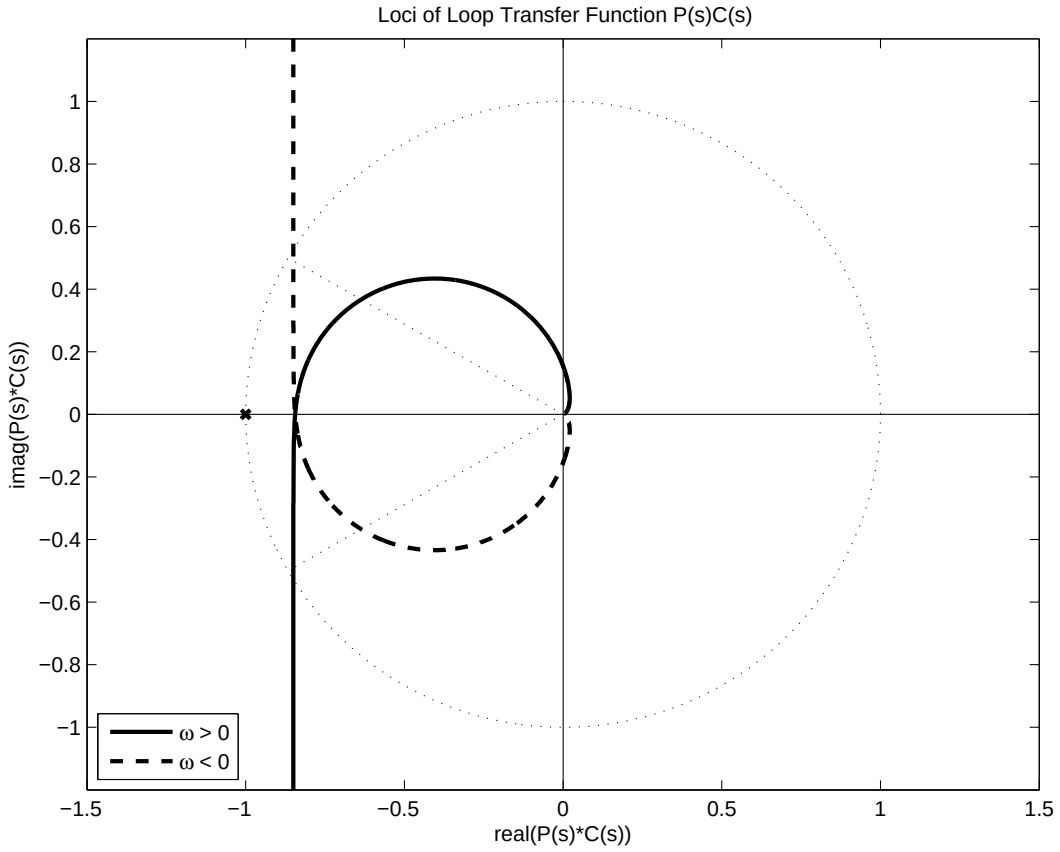
   (d) 1.2 upward gain margin, infinite downward gain margin, and $\pm 60^\circ$ phase margin

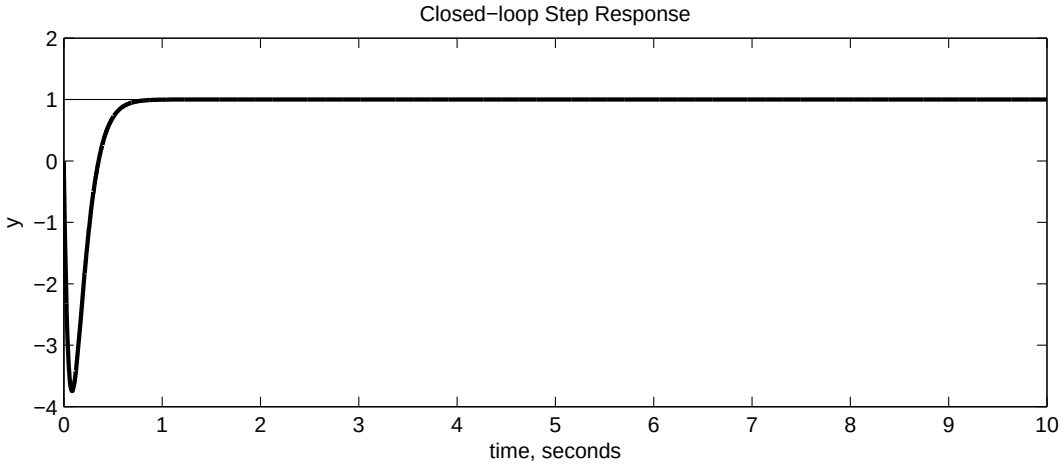

Plant Impulse Response
y
time, seconds


Magnitude |P(s)|
dB
rad/s


Magnitude |P(s)C(s)|
dB
rad/s


Loci of Plant P(s)
imag(P(s))
real(P(s))
60
40
20
0
−20
−40
−60
−50
0
50
100
ω > 0
ω < 0


Loci of Loop Transfer Function P(s)C(s)
imag(P(s)*C(s))
real(P(s)*C(s))
ω > 0
ω < 0
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
−1.5
−1
−0.5
0
0.5
1
1.5


Closed−loop Step Response
y
time, seconds
2
1
0
−1
−2
−3
−4
0
1
2
3
4
5
6
7
8
9
10

38 Monte Carlo and Grid-Based Techniques for Stochastic Simulation

In this problem you will compare the performance of random vs. regular sampling on a specific stochastic dynamics problem.

The system we are considering is a simple rotary mass, controlled by a motor:

$$J\ddot{\phi} = \tau = k_t i,$$

where J is the mass moment of inertia, ϕ is its angular position, τ is the control torque, k_t is the torque constant of the motor, and i is the electrical current applied. While this is a simple control design problem for given values of J and k_t, the situation we study here is when these are each only known within a *range* of values. In particular, J is described as a uniform random variable in the range $[5, 15]kg \cdot m^2$, and k_t is a uniform random variable in the range $[4, 6]Nm/A$. The basic question we ask is: if the control system is designed for a nominal condition, say $J = 10kg \cdot m^2$ and $k_t = 5Nm/A$, how will the closed-loop system vary in its response, for all the possible J and k_t?

This is a question of stochastic simulation, that is, finding the statistics of a function output, given the statistics of its input. The code fragment provided below applies Monte Carlo and grid-based approaches to find the mean and variance of the function $\cos(y)$, when y is uniformly distributed in the range $[2, 5]$. Try running this a few times and notice the effects of changing N. The grid-based approach is clearly giving a good result with far less work than MC - for this example with only one random dimension. In general, the grid-based methods suffer greatly as the d dimension increases; for trapezoidal integration, the error goes as $1/N^{2/d}$, whereas for Monte Carlo it is simply $1/N^{1/2}$ for any d!

1. For the nominal system model (as above) design a proportional-derivative controller so that the closed-loop step response reaches the commanded angle for the first time in about one second and the maximum overshoot is twenty percent. The closed-loop system equation is

$$\begin{aligned} J\ddot{\phi} &= k_t(-k_p(\phi - \phi_{desired}) - k_d\dot{\phi}) \longrightarrow \\ J\ddot{\phi} + k_t k_d\dot{\phi} + k_t k_p\phi &= k_t k_p \phi_{desired}. \end{aligned}$$

   Remember that if you write the left-hand side of the equation as $\ddot{\phi}+2\zeta\omega_n+\omega_n^2$, you can tune this up quite easily because the overshoot scales directly with damping ratio ζ, and you can then adjust ω_n to get the right rise time. Show a plot of the step response and list your two gains k_p and k_d.

   The step response for the nominal system is shown, along with the "four corners" of the parameter space, that is, at the max and min combinations of J and k_t. The gains I used are derived from $\zeta = 0.455$ and $\omega_n = 2.3rad/s$; they are $k_p = 10.58$ and $k_d = 4.19$.

2. Keeping your controller for the nominal system, use the Monte Carlo technique to calculate the mean and the variance of the overshoot z, over the random domain that

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
N = 1000; % how many trials to run

%%%%%%%%%%%%%
% Monte Carlo
%%%%%%%%%%%%%
for i = 1:N,
    q = 2 + 3*rand ;       % random sample from the random domain
    z(i) = cos(q);         % evaluate the function
end;
meanzMC = sum(z)/N ;              % calculate mean
varzMC = sum((z-meanzMC).^2)/N ;  % calculate variance

%%%%%%%
% Grid
%%%%%%%
for i = 1:N,
    q = 2 + 3/N/2 + (i-1)*3/N ;  % regular sample from the random domain
    z(i) = cos(q) ;
end;
meanzGrid = sum(z)/N ;
varzGrid = sum((z-meanzGrid).^2)/N ;

disp(sprintf('Means       MC: %7.4g   Grid: %7.4g   EXACT:  %7.4g', ...
    meanzMC, meanzGrid, (sin(5)-sin(2))/3 ));
disp(sprintf('Variances   MC: %7.4g   Grid: %7.4g', varzMC, varzGrid));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

covers all the possible J and k_t values. Show a plot of mean $\bar{z}$ vs. N, and a plot of variance σ_z^2 vs. N, for $N = [1, 2, 5, 10, 20, 50, 100, 200, ...]$. About how high does N have to be to give two significant digits?

The MC version is pretty noisy, and you'd need at least some hundreds of trials to say with confidence that $\bar{z}$ is between 0.19 and 0.20; ditto for the variance. Clearly a thousand or more trials is preferable.

3. Keeping your controller for the nominal system, use the trapezoidal rule to calculate the mean and variance of z. Let n_1 and n_2 be the number of points in the J and the k_t dimensions, and set $n_1 = n_2$, so that $N = n_1 n_2$. Show plots of $\bar{z}$ and σ_z^2 vs. N, to achieve at least two significant digits.

   We see the grid-based calculation is much cleaner, evidently reaching very stable values

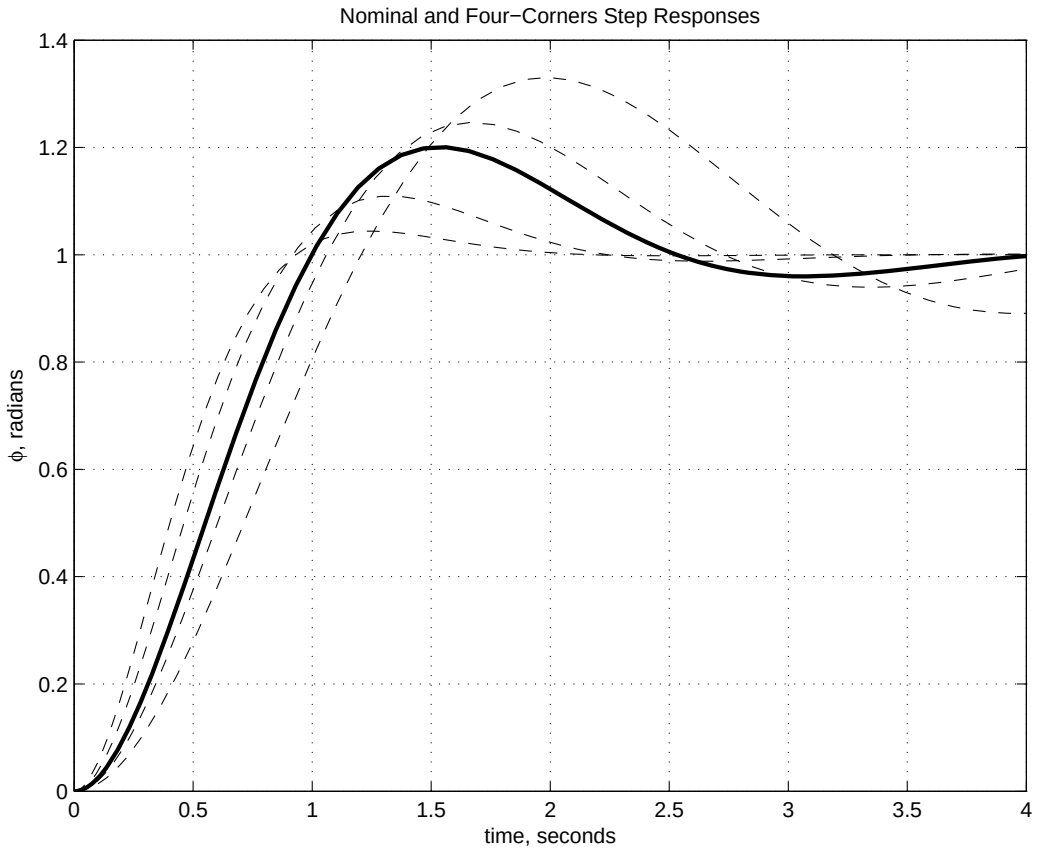
of $\bar{z}$ and $var(z)$ in only a hundred or so trials!

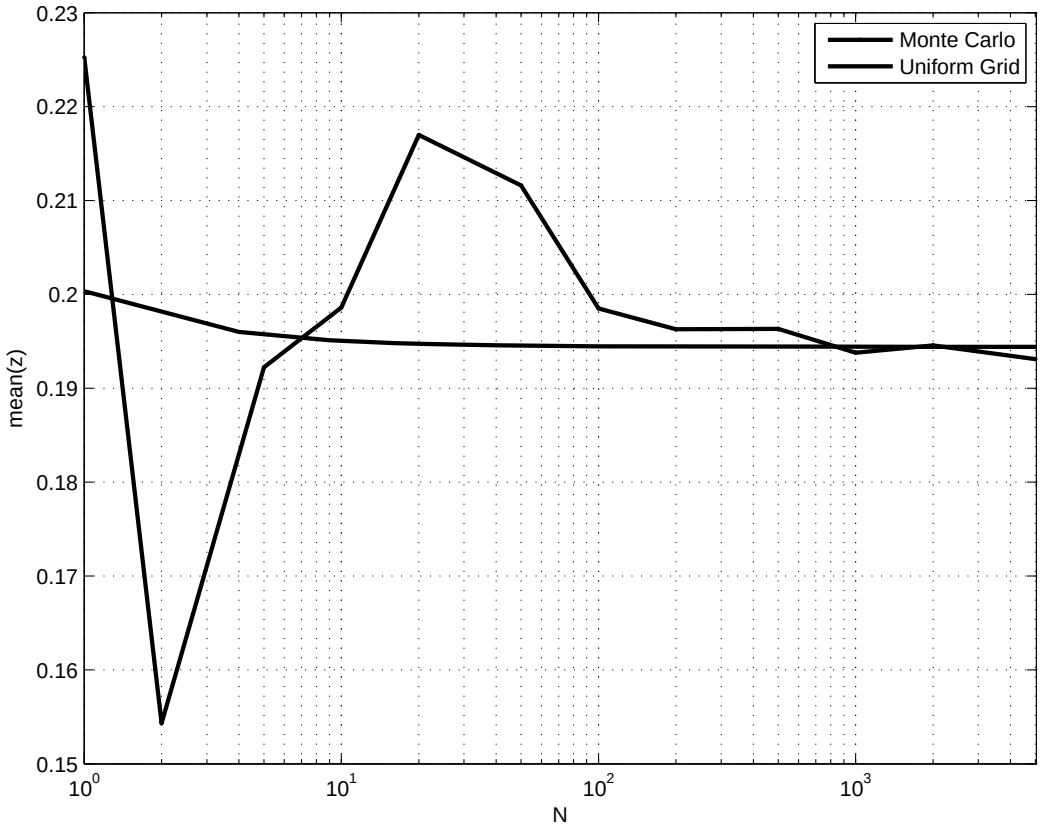
4. Comparing the curves you obtained, which is the superior technique for this problem, and how can you tell?

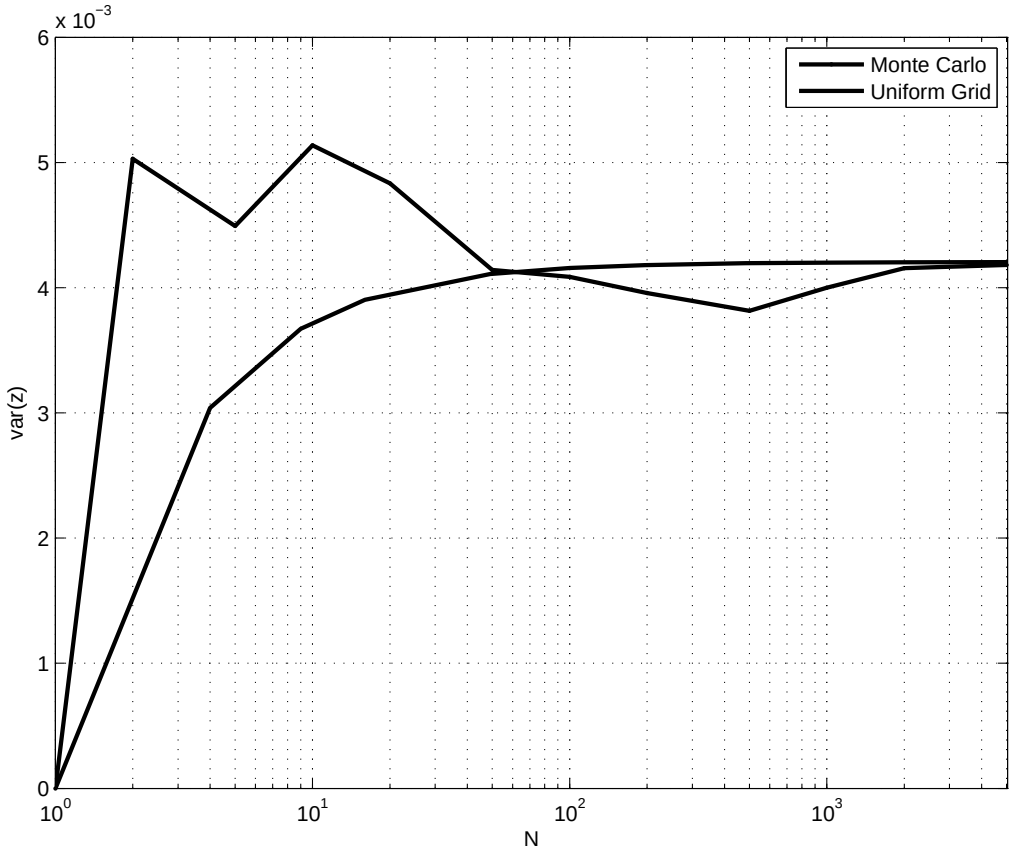
   The grid!

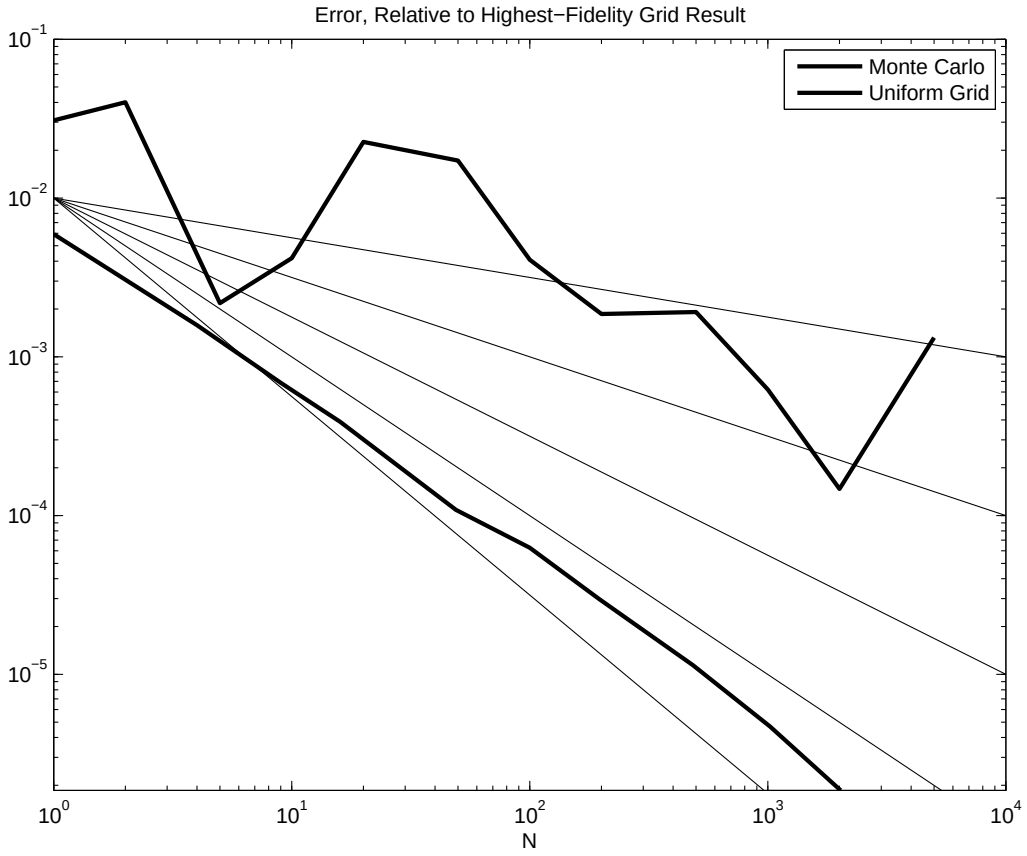
5. Taking your highest-fidelity result for $\bar{z}$ (probably the grid-based calculation with high N) as *truth*, you can calculate the apparent errors in $\bar{z}$ for each method, as a function of N. Making a log-log plot of the absolute values of these errors, can you argue that the error scaling laws $1/N^{1/2}$ (MC) and $1/N^{2/d} = 1/N$ (grid) hold?

   See the last plot. The thin lines indicate trends for $N^{-1/4}, N^{-1/2}, N^{-3/4}, N^{-1}, N^{-5/4}$. The MC points are scattered but generally fit the $N^{-1/2}$ line. The grid data fit the N^{-1} line, and since the dimension is two, it all works out.




Monte Carlo
Uniform Grid
mean(z)
N
0.23
0.22
0.21
0.2
0.19
0.18
0.17
0.16
0.15
10^0
10^1
10^2
10^3


x 10^-3
6
5
4
3
2
1
0
var(z)
10^0
10^1
10^2
10^3
N
Monte Carlo
Uniform Grid


Error, Relative to Highest−Fidelity Grid Result
Monte Carlo
Uniform Grid
10^−1
10^−2
10^−3
10^−4
10^−5
10^0
10^1
10^2
10^3
10^4
N

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Study MC vs. grid-based sensitivity
% FSH MIT 2.017 November 2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

global kp kd J kt ;

Jl = 5 ; Ju = 15 ;   % lower and upper values of the MMOI
ktl = 4 ; ktu = 6 ;   % lower and upper values of torque constant

zeta = .455 ; % set the CL damping ratio and natural frequency
wn = 2.3 ;

tfinal = 4 ;  % final time for all simulations

odeset('AbsTol',1e-4, 'RelTol',1e-2); % lower the accuracy a bit = faster

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% first, show that the gains achieve the desired step response with
% the nominal system

J = (Jl + Ju)/2 ;       % nominal values = midpoints
kt = (ktl + ktu)/2 ;

kp = J*wn^2/kt ;        % control gains - work these out for the nominal
kd = 2*zeta*wn*J/kt ;   % case and then leave them alone

[t,s] = ode45('MCvsGridDeriv',[0 tfinal],[0 0]);

figure(1);clf;hold off;
plot(t,s(:,2),'LineWidth',2);
grid;
xlabel('time, seconds');
ylabel('\phi, radians');

% also run the four corners to make sure the time scale is about right

J4corners = [Jl Jl Ju Ju];
kt4corners = [ktu ktl ktl ktu] ;
figure(1);hold on;
for i = 1:4,
```

```
    J = J4corners(i);
    kt = kt4corners(i);
    [t,s] = ode45('MCvsGridDeriv',[0 tfinal],[0 0]);
    plot(t,s(:,2),'--');
end;
title('Nominal and Four-Corners Step Responses');
pause ;

%%%%%%%%%%%%%%%%
% do the MC runs
%%%%%%%%%%%%%%%%

% Nvec carries the sizes of the ensembles for which we will do statistics
Nvec = [1,2,5,10,20,50,100,200,500,1000,2000,5000] ;

% Note that as written, we do just the largest ensemble, and then
% use portions of it for the statistics

tic;
for i = 1:max(Nvec),
    J = (Ju-Jl)*rand + Jl ; % generate random J in the domain
    kt = (ktu-ktl)*rand + ktl ; % generate random kt in the domain
    [t,s] = ode45('MCvsGridDeriv',[0 tfinal],[0 0]);
    z(i) = max(s(:,2)-1); % get the overshoot
    if rem(i,100) == 0,
        disp(sprintf('Done with %d/%d', i,max(Nvec)));
    end;
end;
toc ;

% calculate the mean and variance for subsets given by Nvec
for k = 1:length(Nvec);
    meanzMC(k) = mean(z(1:Nvec(k))) ;
    varzMC(k) = var(z(1:Nvec(k)),1) ;
end;

figure(2);clf;hold off;
semilogx(Nvec,meanzMC,'.-','LineWidth',2) ;
a=axis ; axis([min(Nvec) max(Nvec) a(3) a(4)]);
grid;

figure(3);clf;hold off;
semilogx(Nvec,varzMC,'.-','LineWidth',2) ;
a=axis ; axis([min(Nvec) max(Nvec) a(3) a(4)]);
```

```
grid;

pause(.1);

%%%%%%%%%%%%%%%%%%
% do the grid runs
%%%%%%%%%%%%%%%%%%

% N1vec is the set of (one-diminsion) ensemble sizes for which we will
% compute statistics.  Note we will use N1 = N2 so that the total number
% of evaluations is N = N1 * N2
N1vec = [1 2 3 4 7 10 14 22 32 45 71];

% Most of the grids don't overlap, so we just use the brute force - do
% all the ensembles and their statistics independently.  It's more
% expensive than what we did for MC

tic;
for k = 1:length(N1vec),
    clear z ;
    for i = 1:N1vec(k),
        for j = 1:N1vec(k),
            J = Jl + (Ju-Jl)/N1vec(k)/2 + (i-1)*(Ju-Jl)/N1vec(k) ;
            kt = ktl + (ktu-ktl)/N1vec(k)/2 + (j-1)*(ktu-ktl)/N1vec(k);
            [t,s] = ode45('MCvsGridDeriv',[0 tfinal],[0 0]);
            z(i,j) = max(s(:,2)-1);
        end;
    end;

    meanzGrid(k) = mean(mean(z)); % the mean is easy...
    % but the variance calculation takes a little more attention
    sumsq = 0 ;
    for i = 1:N1vec(k),
        for j = 1:N1vec(k),
            sumsq = sumsq + (z(i,j) - meanzGrid(k))^2 ;
        end;
    end;
    varzGrid(k) = sumsq / N1vec(k)^2 ;

    disp(sprintf('Done with %d/%d', sum(N1vec(1:k).^2),sum(N1vec.^2)))
end;
toc;

figure(2);hold on;
```

```
semilogx(N1vec.^2,meanzGrid,'r','LineWidth',2);
axis('auto');a=axis ; axis([min([Nvec,N1vec.^2]) max([Nvec,N1vec.^2]) a(3) a(4)]);
legend('Monte Carlo', 'Uniform Grid');
xlabel('N');ylabel('mean(z)');

figure(3);hold on;
semilogx(N1vec.^2,varzGrid,'r','LineWidth',2);
axis('auto');a=axis ; axis([min([Nvec,N1vec.^2]) max([Nvec,N1vec.^2]) a(3) a(4)]);
legend('Monte Carlo', 'Uniform Grid');
xlabel('N');ylabel('var(z)');

figure(4);clf;hold off;
surf(z);
title('Values of z Seen Over the Random Domain');

figure(5);clf;hold off;
loglog(Nvec,abs(meanzMC - meanzGrid(end)),'LineWidth',2);
hold on;
loglog(N1vec.^2,abs(meanzGrid - meanzGrid(end)),'r','LineWidth',2);
for i = 3:7,
    loglog( [1e0 1e4], [.01 10^(-i)]) ;
end;
title('Error, Relative to Highest-Fidelity Grid Result');
legend('Monte Carlo', 'Uniform Grid');
a = axis ; axis([a(1) a(2) abs(meanzGrid(end-1)-meanzGrid(end)), a(4)]);
xlabel('N');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [sdot] = MCvsGridDeriv(t,s) ;
global kp kd J kt ;

phidot = s(1);
phi = s(2);

torque = kt*(-kp*(phi-1) - kd*phidot); % control action

phidotdot = torque/J ; % equation of motion

sdot(1,1) = phidotdot;
sdot(2,1) = phidot ;
```

%%

39 Hurricane Ida Wind Record

The remnants of Hurricane Ida swept through Norfolk, Virginia during the period 10-15 November 2009, with 25cm of rain and high winds.The data file `homework9.txt` contains wind data recorded at Weather Station "Larchmont-Cornicks," in Norfolk. There are three columns, with average wind direction $\bar{\theta}$ in degrees (CW positive from north; zero is wind coming from the north), average wind speed $\bar{u}$ in miles per hour, and gust wind speed $\tilde{u}$, also in miles per hour. $\bar{\theta}$ and $\bar{u}$ are average values taken over five-minute intervals, and $\tilde{u}$ denotes the peak speeds taken over these same intervals. Each row corresponds with one five-minute period.

1. Make some nice annotated plots of the data, and write a few paragraphs to describe what features you can see in the three data channels. Notice that there are approximately six days' worth of data here - it was a long storm!

   Between calm wind conditions, this weather event lasted about 7000 minutes, or 4.8 days.

   The wind started from the northeast and then gradually came around the the northwest. Interestingly, there was a short period of no wind direction data near time 1700 minutes (lack of noise is a dead giveaway), and also some odd five-degree quantization effects near the zero direction.

   Looking at the speeds now, we see immediately a one-mph quantization in the gust speed. In the intensifying phase there was a temporary drop in wind speed at time 2000 minutes, but winds quickly built up again by time 3000 minutes. The mean wind speed reached a maximum value of fifty miles per hour, near time 4000 minutes, with a gust in that bin of 57 mph. In the first half of the storm event, gust speed beyond the mean speed seems to be about ten mph, whereas in the second half of the event, it is less; this may indicate that turbulence was highest at the beginning of the storm and lower at the end. Also, we notice that starting at about 4000 minutes, there were several dramatic, one-hour-scale changes in the wind speed (but not the direction). These correspond with various fronts moving through the area.

   More broadly, wind data has some subtleties not shared directly by ocean wave data. A major distinction is that ocean waves are taken with respect to a zero level, whereas wind events are typified by a mean wind and a varying, additive component.[1] *We could debate whether the mean wind or the absolute gusts, or some combination of them, should be used in design. The point of view in our present question is that the gusts relative to the mean winds are most important. The interested reader can look at a histogram of the absolute gusts, but will find that the statistics are ruined by the non-stationarity of the weather event.*

2. For the purpose of assessing gust size relative to the mean wind speed, we first need to filter (or smooth) $\bar{u}$ while also allowing for some low-frequency content, e.g., on the

[1] Spectral content of some other hurricane data is given in another of the worked problems, **Hurricane Winds**.

one-hour scale. You can make a simple filter as follows: Suppose the raw, bin-indexed input data is $[\bar{u}_1, \cdots, \bar{u}_n]$. We'll call the filtered version of this $[y_1, \cdots, y_n]$. Initialize $y_1 = \bar{u}_1$, and then recursively apply $y_k = (1-\alpha)y_{k-1} + \alpha\bar{u}_k$, for $k = 2, \cdots, n$, and some fixed α, $0 \leq \alpha \leq 1$. You see that with $\alpha = 0$, y_k gets only information from y_{k-1}; the input $\bar{u}_k$ is never used, and so y is a flat line for all time, at x_1. With $\alpha = 1$, y simply matches the input. The first case allows NO frequency content of the input signal $\bar{u}$ to come through, whereas the second case allows ALL of its frequency content through. As α takes intermediate values, you will see that you can control how much filtering is occurring.

Select a specific α for the purpose of smoothing the average wind speed data $\bar{u}$. Show a time plot overlaying the raw data and the smoothed version, and list your chosen value for α.

The plot shows smoothed mean wind data for a choice of $\alpha = 0.25$. This setting takes off a good amount of the noise, but doesn't eliminate too many of the details. Other values of α will give quite different results in the fit below.

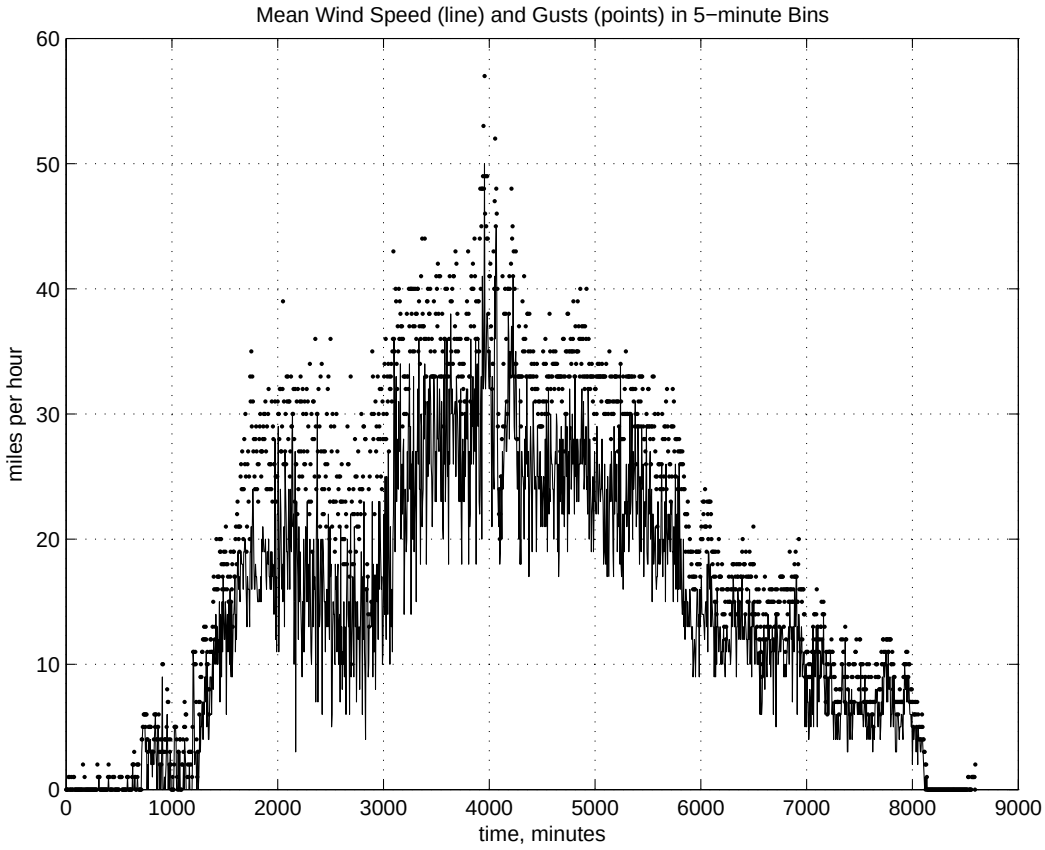
3. With $\bar{u}$ suitably filtered, we can now estimate the sizes of the gusts more accurately, namely as $g_k = \tilde{u}_k - y_k$. We are subtracting off the smoothed mean wind data from the recorded gusts.

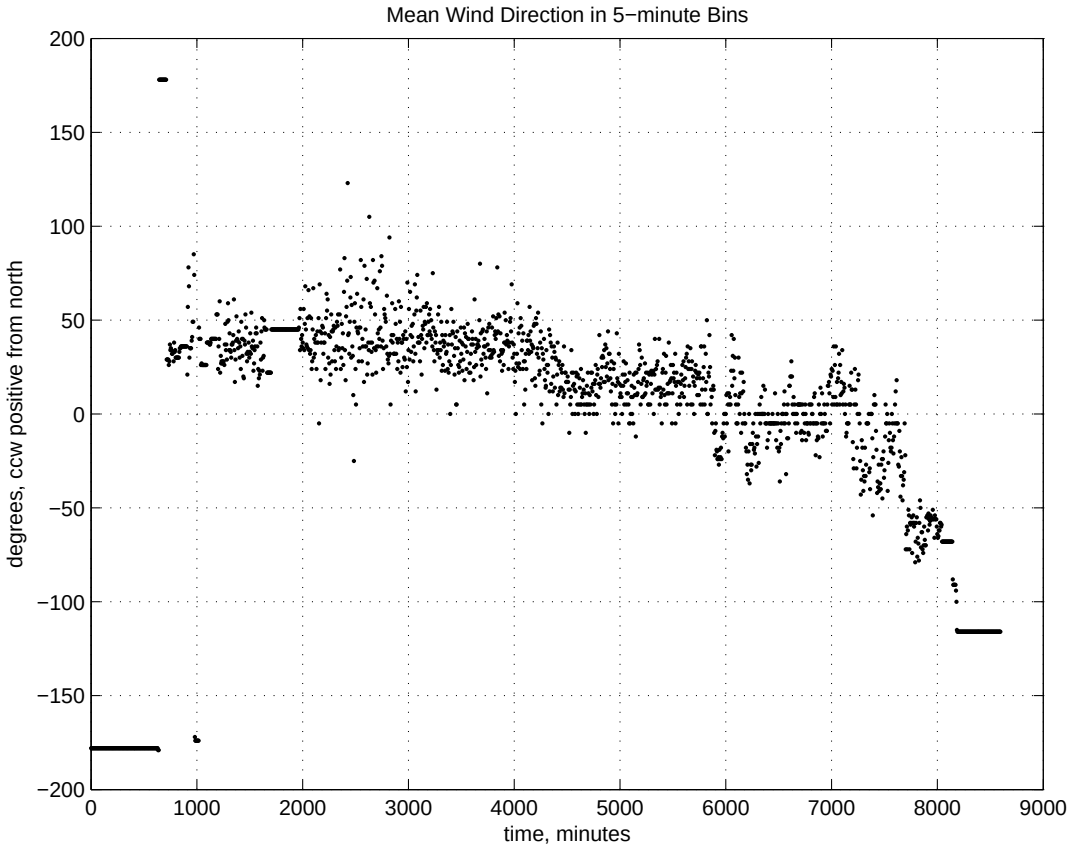
   Show a histogram of g, and discuss how it does or does not look like the Gaussian and Rayleigh distributions. Notice that this data set is clearly non-stationary, so there is no reason up front that either should fit. (Because of the smoothing, there may occur a few instances where $\tilde{u}$ is lower than y, and hence $g < 0$; you can ignore these.)

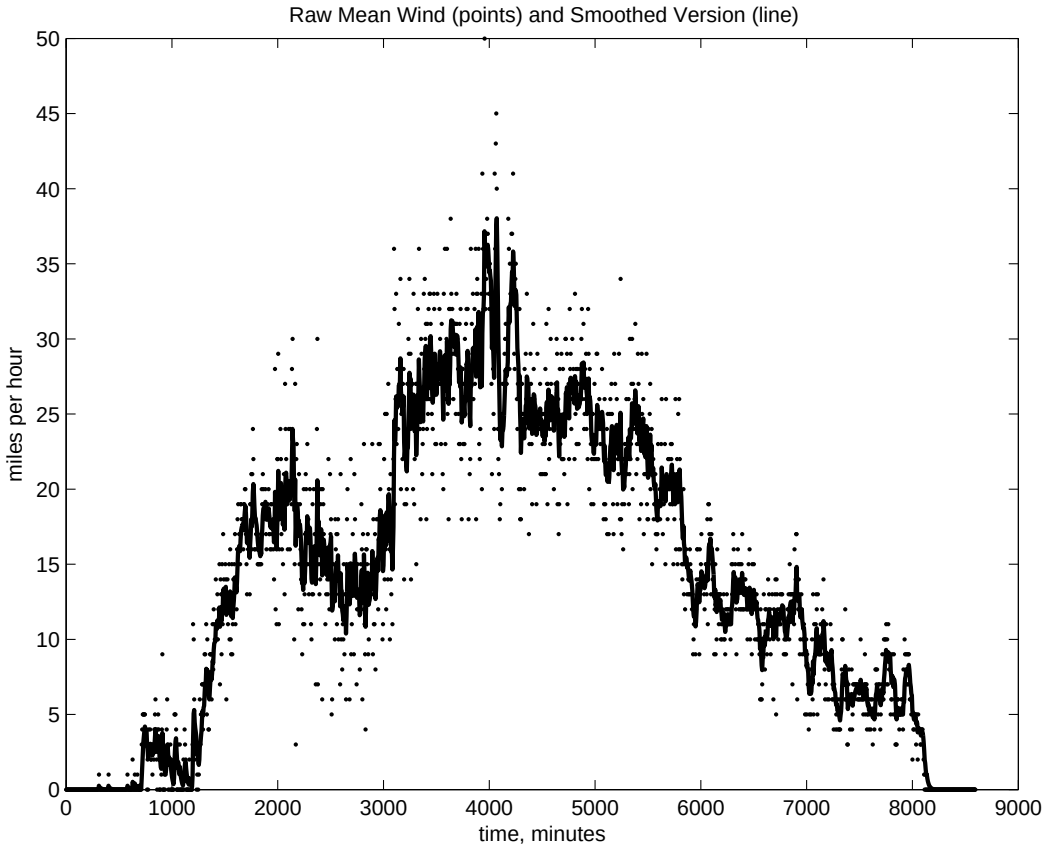
   The histogram is Rayleigh-like at first look. The extra area near zero and below zero speed is due to our smoothing of the mean wind data, which puts some mean values above the associated gust values.

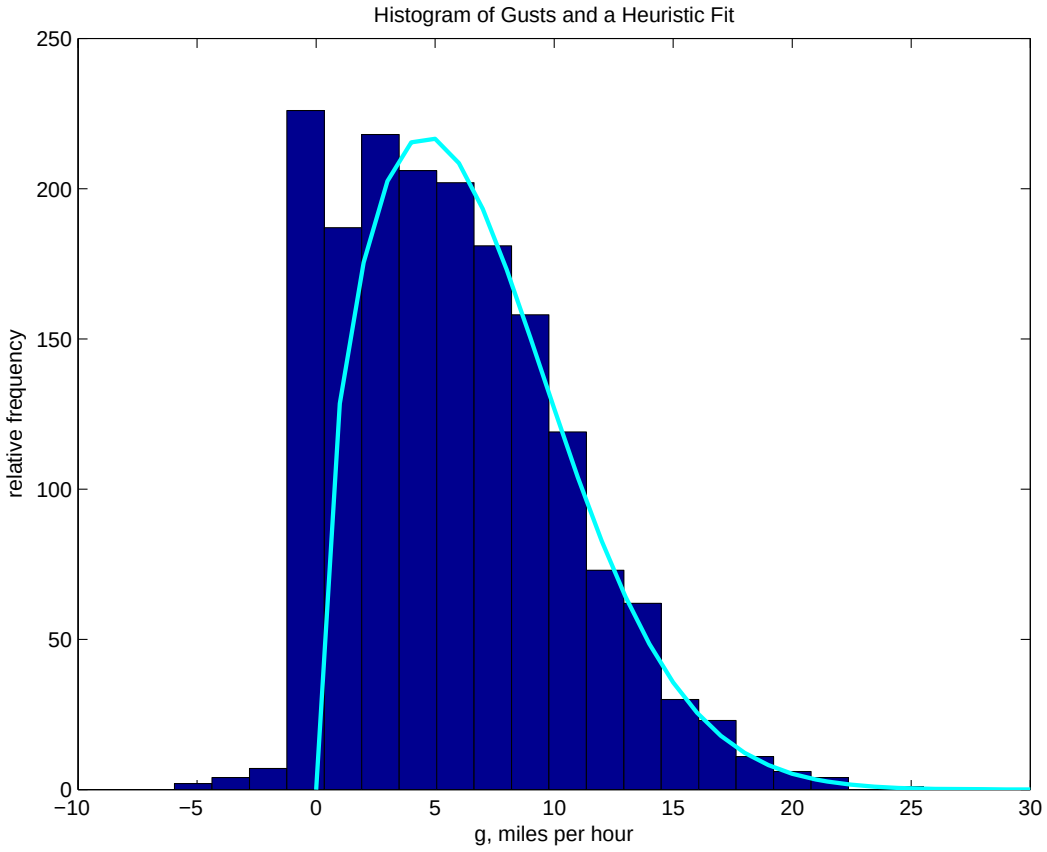
4. Make a plot showing a reasonable fit to the histogram (or pdf), using one of the distributions we have encountered in this class or perhaps a new one. State whether you think this fitted curve could effectively capture extreme events in a storm like this one, and why. You can answer this question based entirely on the graph showing the curve fit.

   The best fit I found is of the form $\sqrt{g}e^{-g^2}$, which is neither Weibull nor Rayleigh. It does a good job matching the histogram, at least away from $g = 0$, and up to the highest gust of about 25 mph. Certainly more data would be desirable for making this assessment of fit.


Mean Wind Speed (line) and Gusts (points) in 5−minute Bins
miles per hour
time, minutes
0
10
20
30
40
50
60
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Mean Wind Direction in 5−minute Bins
degrees, ccw positive from north
time, minutes
200
150
100
50
0
−50
−100
−150
−200
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Raw Mean Wind (points) and Smoothed Version (line)
miles per hour
time, minutes
0
5
10
15
20
25
30
35
40
45
50
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Histogram of Gusts and a Heuristic Fit
relative frequency
g, miles per hour
250
200
150
100
50
0
−10
−5
0
5
10
15
20
25
30

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Study Nov 2009 Hurricane Ida wind data.
% FSH MIT ME 2.017 Nov 2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
load homework9.txt;

n = length(homework9);
t = 0:5:5*(n-1); % time vector

dir = homework9(:,1); % direction
for i =1:n, % unwrap the direction data for a good plot
    if dir(i) > 180,
        dir(i) = dir(i) - 360 ;
    end;
end;

ubar = homework9(:,2); % mean wind in the bin
utilde = homework9(:,3); % peak gust in the bin

figure(1);clf;hold off;
plot(t,ubar,t,utilde,'.');
xlabel('time, minutes');
ylabel('miles per hour');
title('Mean Wind Speed (line) and Gusts (points) in 5-minute Bins');
grid;

figure(2);clf;hold off;
plot(t,dir,'.');
ylabel('degrees, ccw positive from north');
xlabel('time, minutes');
title('Mean Wind Direction in 5-minute Bins');
grid;

alpha = .25; % set the low-pass filter parameter ...
y(1,1) = ubar(1);
for i = 2:n, % ... and filter
    y(i,1) = (1-alpha)*y(i-1,1) + alpha*ubar(i);
end;

% plot the raw mean wind and the filtered values
figure(3);clf;hold off;
plot(t,ubar,'m.',t,y,'LineWidth',2);
```

```
xlabel('time, minutes');
title('Raw Mean Wind (points) and Smoothed Version (line)');
ylabel('miles per hour');

g = utilde-y; % gusts relative to the smoothed mean values

% make histogram (pdf) of the gusts
figure(4);clf;hold off;
hist(g,20);

% make up a fitting line and overlay it
gsize = 0:30; % gust levels to evaluate the fitting function
fitCurve = 130*gsize.^(.5).*exp(-gsize.^2/85);

figure(4);hold on;
plot(gsize,fitCurve,'c','LineWidth',2);
xlabel('g, miles per hour');
ylabel('relative frequency');
title('Histogram of Gusts and a Heuristic Fit');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

40 Metacentric Height of a Catamaran

Consider the roll stability of a catamaran described as follows: total beam is $6b$, where b is the beam of each hull, the draft is T, and each hull is rectangular in cross-section.

1. What is the distance KM for this section, at small roll angles? You may approximate the area of the "buoyancy wedges" as the mean height times the width.

   The differential moment is as follows:

   $$dM = 2 \times \frac{5}{2}\, b^2\theta \times \frac{5}{2}\, b \times \rho g\, dx,$$

   where the terms separated by the $\times$ symbol are for: two wedges, the area enclosed in each wedge (approximation), the moment arm of each wedge, density and gravity and the differential length. We know that $\rho g d\nabla (KM)\theta = dM$, which leads directly to

   $$KM = \frac{25}{4}\frac{b^2}{T}.$$

   There is an additional term $T/2$ (as in the case of monohulls) but this is very small by comparison for a catamaran.

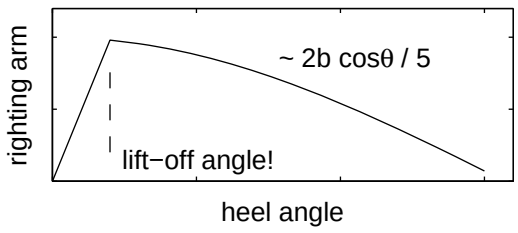
2. What is KM for a single block of draft T and beam $2b$?

   We find for the block $KM = b^2/3T + T/2$. The "wedge" term ($b^2/3T$) is about one-twentieth of the value from part a).

3. Make a sketch of the righting moment as the angle increases from zero to the point where one hull lifts out of the water, and then beyond. Describe what has happened in this situation and hence one of the inherent problems with catamarans.

   See the figure: The righting moment essentially goes flat once a hull is out of the water, because the there is no more loss of buoyancy on one side and gain of buoyancy on the other, as roll increases. You have left only the center of mass against buoyancy at the submerged hull. As the angle increases further, the righting moment arm decreases with the cosine of the roll angle.

   A proa is an unusual sailing boat with one large hull, and a very light pontoon on a long arm. Proas are double-ended - you keep the pontoon on the leeward side!



41 Floating Structure Heave and Roll

Consider the heave and roll response of a two-hull structure with the parameters below:

mass (metric tons)	3000
body rotary inertia (kg-m^2)	3×10^8
beam of each hull b (m)	5
open space between hulls B (m)	20
draft T (m)	10
vessel length L (m)	30
each hull effective added mass A_{33} (tons)	750
each hull effective damping coefficient B_{33} (Ns/m)	4×10^5

The hull is assumed to be uniform in cross-section over the entire length. We are going to study the behavior of this structure in beam waves - that is, moving from port to starboard across the two hulls. We will use the Bretschneider spectrum as in Homework 2, for sea states 2-6.

1. About what are the undamped natural frequencies in heave and in roll? Assume in this problem that the vessel's center of mass is near the waterline.

   The square root of waterplane stiffness divided by mass and added mass gives about 0.700 rad/s in heave and 0.875 rad/s in roll.

2. About what are the damping ratios in heave and in roll?

   We have $\zeta \simeq 0.095$ in heave and 0.12 in roll; not much damping!

3. Up to what wave frequency is the long-wavelength approximation valid for this problem? Make a plot to verify that you have a ratio of at least three or so, up to the highest frequency you will use in the following calculations.

   The figure below shows that we are OK up to about two radians per second, considering each hull alone. If we take the whole vessel beam, then we could only do the higher sea states, e.g., $\omega < 0.9$.

4. Taking $y = 0$ at the port side of the port hull, compute for a range of wave frequencies the phase angle in the incident wave heave force (F_{3I} in the notes) seen at the middle of the port hull and then the middle of the starboard hull. Make and annotate a plot.

   This is shown in the figure; formulas for each term are given below. The angle for the far hull changes very quickly as the wavelengths grow to and then exceed B.

5. Write out the differential equation that governs the heave motions, under wave disturbances, and including the incident wave, diffraction, and radiation terms.

We have from the formulas in the notes

$$\begin{aligned} mz'' + (B_{33p} + B_{33s})z' + (C_{33p} + C_{33s})z &= (F_{3Ip} + F_{3Is})\eta(y=0) + \\ & (F_{3Dp} + F_{3Ds})\eta(y=0) + (F_{3Rp} + F_{3Rs})z \end{aligned}$$

where the p and s subscripts refer to port and starboard hulls, respectively. The coefficients are:

$$\begin{aligned} F_{3Ip} &= -\rho g \frac{e^{-kT}}{k}[\sin\omega t(\cos kb - \cos 0) - \cos\omega t(\sin kb - \sin 0)] \\ F_{3Is} &= -\rho g \frac{e^{-kT}}{k}[\sin\omega t(\cos k(2b+B) - \cos k(b+B)) - \cos\omega t(\sin k(2b+B) - \\ &\quad \sin k(b+B))] \\ F_{3Dp} &= -e^{-kT/2}A_{33p}\omega^2[\cos\omega t\cos(kb/2) + \sin\omega t\sin(kb/2)] \\ F_{3Ds} &= -e^{-kT/2}A_{33s}\omega^2[\cos\omega t\cos(k(B+b+b/2)) + \sin\omega t\sin(k(B+b+b/2))] \\ F_{3Rp} &= A_{33p}\omega^2 \\ F_{3Rs} &= A_{33s}\omega^2 \end{aligned}$$

6. Write the transfer function for heave, relating the input (wave elevation at $y = 0$) to the output (heave motion). You have to separate the time and space components, because the two hulls are in different locations: you should see terms like $\sin(\omega t)$, $\sin(k(b+B+b/2))$ and so on. Note that k here is related to ω through the dispersion relation. In the frequency domain, $\cos\omega t$ will become simply one, and $\sin\omega t$ will become $-j = -\sqrt{-1}$.

   Collecting the terms in z to the left-hand side of the above equation, and η on the right, we get

$$\frac{z(\omega)}{\eta(x=0,\omega)} = \frac{F_{3Ip} + F_{3Is} + F_{3Dp} + F_{3Ds}}{-F_{3Rp} - F_{3Rs} - m\omega^2 + j\omega(B_{33s} + B_{33p}) + C_{33p} + C_{33s}}$$

7. Write out the differential equation that governs the roll motions, under wave disturbances, and including the incident wave, diffraction, and radiation terms.

   Using the coefficients above, we have

$$\begin{aligned} J\theta'' + (B_{33p} + B_{33s})r^2\theta' + (C_{33p} + C_{33s})r^2\theta &= (F_{3Ip} - F_{3Is} + F_{3Dp} - F_{3Ds})r\eta(x=0) + \\ & (F_{3Rp} - F_{3Rs})r^2\theta \end{aligned}$$

   Here r is the moment arm from the centerline to the middle of each hull, i.e., $r = B/2 + b/2$.

8. Write the transfer function for roll, relating the input (wave elevation at $y = 0$) to the output (roll motion).

Collecting terms again, the roll transfer function is:

$$\frac{\theta(\omega)}{\eta(x=0,\omega)} = \frac{(F_{3Ip} - F_{3Is} + F_{3Dp} - F_{3Ds})r}{-(F_{3Rp} - F_{3Rs})r^2 - \omega^2 J + j\omega(B_{33p} + B_{33s})r^2 + (C_{33p} + C_{33s})r^2}$$

9. Make a plot of the two transfer function magnitudes, showing the heave transfer function in units of (meters/meter) and the roll in (degrees/meter).

   See attached plot.

10. How do the major features in the transfer function relate to the phase plot you made? How do they relate to the natural frequencies and damping ratios you estimated?

    First consider the undamped natural frequencies in heave and roll, which we found to be about 0.7 and 0.87 rad/s respectively. There is a peak in the heave transfer function near 0.75, and a peak in roll around 1.0; these are in reasonable agreement. More interestingly, referring to the phase plot, we know that the incident forcing on the hulls is out of phase at about 1.1 rad/s and this is a point where the transfer function in heave is near zero, while in roll we have a big response. Note also how close this point is to the roll resonance. On the other hand, at about 1.6 rad/s the incident forcing is in phase (360 degrees) across the two hulls; at this point, the heave transfer function is reasonably large, while the roll transfer function goes to zero.

11. For each of the sea states [2-6], compute the "significant height" in of the vessel motion in heave (meters) and roll (meters).

    Square the transfer function and multiply it by the spectrum to get the spectrum of the response. See the figure, which shows results for all the sea states. I get significant heights in heave of [0.076, 0.32, 0.75, 1.26,1.52] meters, and in roll of [4.3, 18, 43, 72, 87] degrees. One should question the linear assumptions for such large roll angles as this!

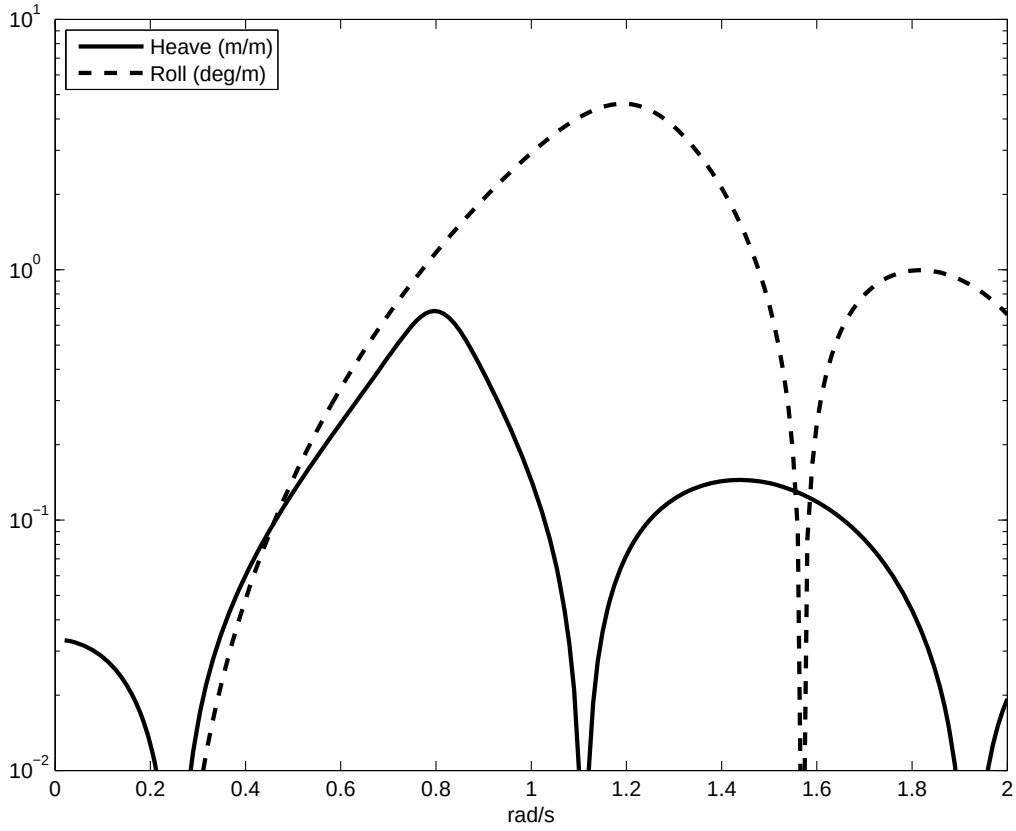
```
%--------------------------------------------------------------------
% heave and pitch analysis of a pair of hulls in waves
%

clear all;

g = 9.81 ; % gravity
rho = 1000 ; % water density

Tu = 5 ; % draft of each hull, upper
bu = 5 ; % beam of each hull, uppper
Tl = 10 ; % draft of lower hull
bl = 5 ; % beam of lower hull
B = 20 ; % distance between the two hulls' inner faces
```

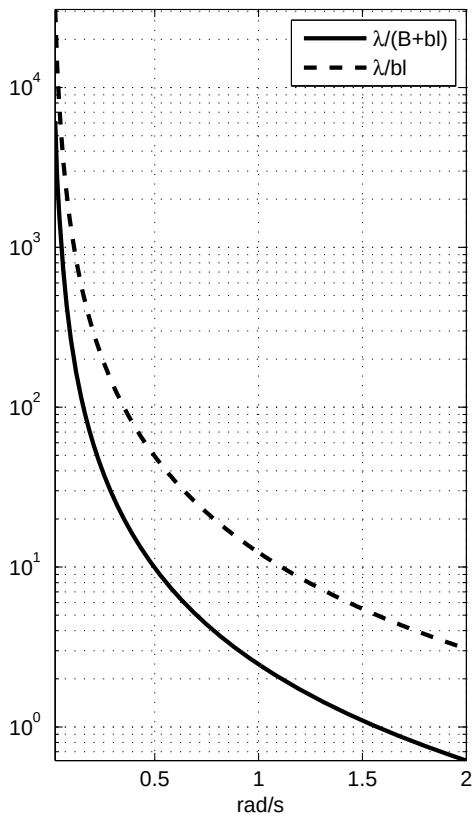
```
L = 30 ; % "length" of the pair of hulls

% show the data from the table in the problem
SSvec = [2 3 4 5 6] ; % sea states
wmvec = 2*pi*ones(size(SSvec)) ./ [6.3 7.5 8.8 9.7 12.4] ;
    % modal frequencies
Hsigvec = [0.3 0.9 1.9 3.3 5.0] ; % significant wave heights

arm = (B/2 + bl/2) ; % moment arm for stiffness and damping terms

Del = 2*L*( Tu*bu + (Tl-Tu)*bl ) ; % volume
m = Del*rho ; % mass
J = m*(B/2)^2 ; % rotary moment of inertia of the body, approx.

A33s = rho*bl^2*L ; % added mass, approx.
A33p = A33s ;

B33s = 1/2*rho*L*bl*8/3/pi*(2*pi/10)*10 ;
B33p = B33s ;

C33s = rho*g*bu*L ; % waterplane area stiffness, stbd hull
C33p = C33s ;       % port hull

[m J bu B Tl L A33s B33s]' % show us the matrix of parameters

disp('------------------------------------------------------------');
disp(sprintf('Approximate heave nat. freq:  %g rad/s', ...
    sqrt(C33s/(m)))); % with added mass
disp(sprintf('Approximate roll nat. freq:  %g rad/s', ...
    sqrt(C33s*arm^2/J))); % with added mass
disp(sprintf('Approximate damping ratio in heave:  %g', ...
    B33s / 2 / sqrt(m*C33s))); % with added mass
disp(sprintf('Approximate damping ratio in roll:  %g', ...
    B33s*arm^2 / 2 / sqrt(J*C33s*arm^2))); % with added mass

sinwt = -sqrt(-1) ; % frequency domain equivalents for sin and cos
coswt = 1 ;

wvec = .02:.01:2 ;

for i=1:length(wvec),
    w = wvec(i) ;

    k  = w^2 / g ; % dispersion relation
```

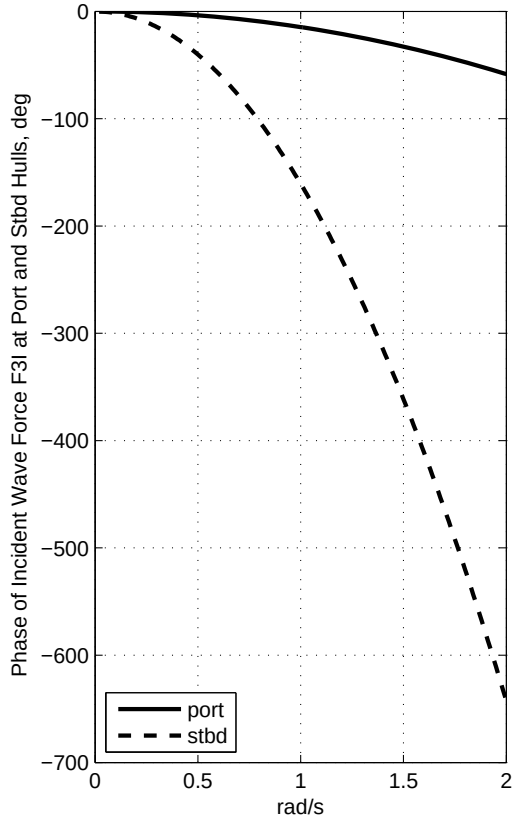
```
    lam(i) = 2*pi/k ; % wavelength

    % force due to incident waves, port and starboard.  We
    % take y=0 at the port side of the port hull
    F3Ip(i) = -rho*g*exp(-k*Tl)/k*[
            sinwt*(cos(k*bl)-cos(0)) - ...
            coswt*(sin(k*bl)-sin(0))] ;
            % multiplies wave elevation at y=0
    F3Is(i) = -rho*g*exp(-k*Tl)/k*[
            sinwt*(cos(k*(2*bl+B))-cos(k*(bl+B))) - ...
            coswt*(sin(k*(2*bl+B))-sin(k*(bl+B)))] ;
            % multiplies wave elevation at y=0

    % forces due to diffraction
    F3Dp(i) = exp(-k*Tl/2)*A33p*(-w^2)*[coswt*cos(k*bl/2) + ...
        sinwt*sin(k*bl/2)] ;
            % multiplies wave elevation at y=0
    F3Ds(i) = exp(-k*Tl/2)*A33s*(-w^2)*[coswt*cos(k*(B+bl+bl/2)) + ...
            sinwt*sin(k*(B+bl+bl/2))] ;
            % multiplies wave elevation at y=0

    % forces due to radiation
    F3Rp(i) = -A33p*(-w^2) ; % multiplies body heave at port hull
    F3Rs(i) = -A33s*(-w^2) ; % multiplies body heave at stbd hull

    tfHeave(i) = (F3Ip(i) + F3Is(i) + F3Dp(i) + F3Ds(i)) / ...
        (-F3Rp(i) - F3Rs(i) - m*w^2 + ...
        sqrt(-1)*w*(B33s + B33p) + (C33s + C33p)) ;

    tfRoll(i) = ...
        ( F3Ip(i)*arm - F3Is(i)*arm + F3Dp(i)*arm - F3Ds(i)*arm) / ...
        (-F3Rp(i)*arm^2 + F3Rs(i)*arm^2 - J*w^2 + ...
        sqrt(-1)*w*(B33p*arm^2 + B33s*arm^2) + ...
        (C33p*arm^2 + C33s*arm^2)) ;

end;

figure(1);clf;hold off;
semilogy(wvec,abs(tfHeave),wvec,(abs(tfRoll)*180/pi),'--','LineWidth',2);
axis([0 2 .01 10]);
legend('Heave (m/m)','Roll (deg/m)',2);
xlabel('rad/s');

figure(2);clf;hold off;
```

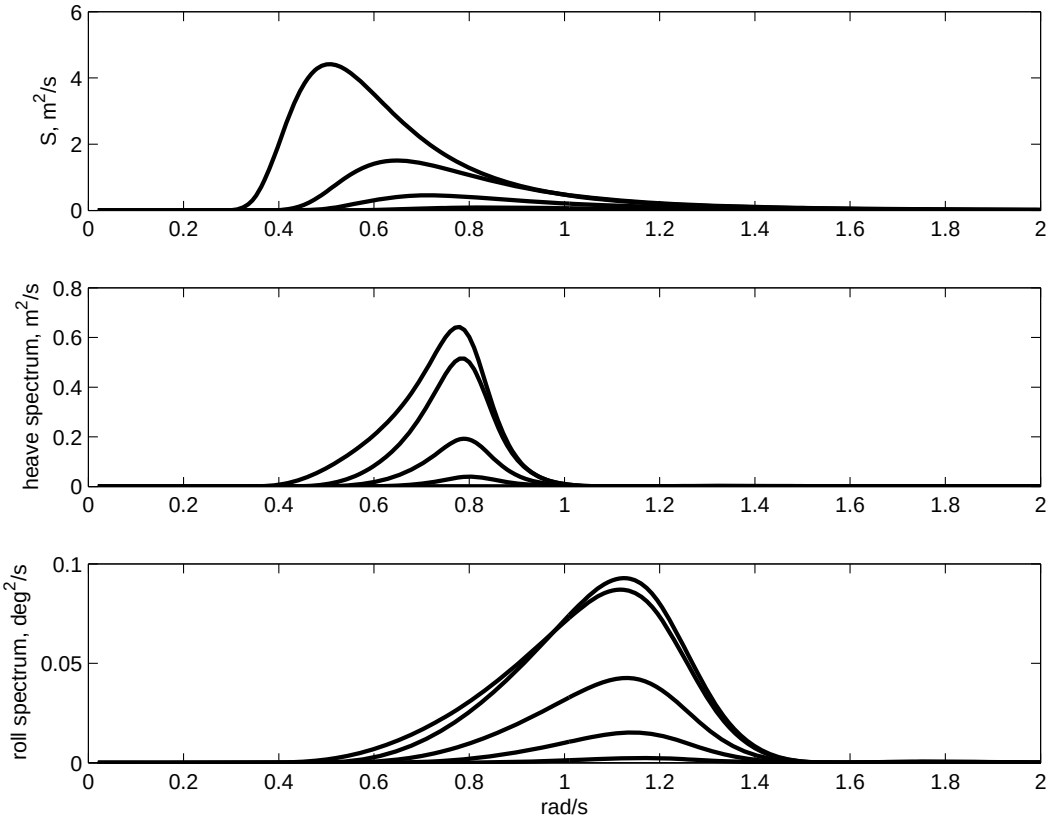
```
subplot(122)
plot(wvec,unwrap(angle(F3Ip))*180/pi,wvec,...
    unwrap(angle(F3Is))*180/pi,'--','LineWidth',2);
ylabel('Phase of Incident Wave Force F3I at Port and Stbd Hulls, deg');
xlabel('rad/s');
grid;
legend('port','stbd',3);
subplot(121);
semilogy(wvec,lam/(B+bl),wvec,lam/bl,'--','LineWidth',2) ;
grid;legend('\lambda/(B+bl)','\lambda/bl');
axis('tight');
xlabel('rad/s');

% MAKE UP THE BRETSCHNEIDER SPECTRUM

figure(4);clf;hold on;
% step through the different seastates
for i = 1:length(wmvec),

wm = wmvec(i) ;
Hsig = Hsigvec(i) ;
SS = SSvec(i) ;

for j = 1:length(wvec),
w = wvec(j) ;
S(j) = 5/16 * wm^4 / w^5 * Hsig^2 * exp(-5 * wm^4 / 4 / w^4) ;
end;

% check that we got the right formula!
disp(sprintf(...
        '[Square Root of Integral of Area of S:  %g;  Hsig/4:  %g]', ...
sqrt(sum(S)*mean(diff(wvec))), 1/4*Hsig));

    Heave = S.*conj(tfHeave).*tfHeave ;
    Roll = S.*conj(tfRoll).*tfRoll ;

    subplot(311);
    plot(wvec,(S),'LineWidth',2);
    ylabel('S, m^2/s');
    hold on;
%    a=axis ; axis([a(1:2) .001 3]);
    subplot(312);
    plot(wvec,(abs(Heave)),'LineWidth',2);
    hold on;
```

```
%    a=axis ; axis([a(1:2) .001 10]);
    ylabel('heave spectrum, m^2/s');
    subplot(313);
    plot(wvec,(abs(Roll))*180/pi,'LineWidth',2);
    hold on;
%    a=axis ; axis([a(1:2) .001 .5]);
    xlabel('rad/s');
    ylabel('roll spectrum, deg^2/s');

    sigHeaveHeight = sqrt(sum(Heave)*mean(diff(wvec)))*4 ;
    sigRollHeight = sqrt(sum(Roll)*mean(diff(wvec)))*4 ;

    disp(sprintf('For SeaState %d:',SS));
    disp(sprintf('The significant height in heave is %g m.',...
        sigHeaveHeight));
    disp(sprintf('The significant height in  roll is %g deg.',...
        sigHeaveHeight*180/pi));

end;
%-------------------------------------------------------------------
```


Heave (m/m)
Roll (deg/m)
rad/s


λ/(B+bl)
λ/bl
rad/s


Phase of Incident Wave Force F3l at Port and Stbd Hulls, deg
port
stbd
rad/s


S, m2/s
heave spectrum, m2/s
roll spectrum, deg2/s
rad/s

42 Submerged Body in Waves

A fixed, submerged body has a circular cross-section of constant diameter one meter, with length twenty meters. For the calculations below, use the Morison formulation for wave-induced force, with $C_d = 1.2$. Fix the wave amplitude at one meter, and consider the frequency range of 0.1 to two radians per second.

For each of the first four items below, in addition to making a plot please answer: What is the maximum magnitude and at what frequency does it occur? What are the physical mechanisms that create the curve you see?

1. Consider the body oriented horizontally and pointing into waves, at a fixed depth of four meters. Calculate and show in a graph the total heave force magnitude as a function of wave frequency ω.

   See Figure 1. The maximum magnitude of vertical force is 18.1kN, at about 1.1 rad/s. Since the wave amplitude is taken as constant at all frequencies, the Morison forces are increasing rapidly as the frequency goes up. At about 1.75 rad/s, however, there is a zero net force because the wavelength is the same as the body length.

   The fact that this plot is made for a unity wave amplitude means that the square of this function could be multiplied by the wave spectrum, to get the spectrum of the force. Beware of the deep-water assumption however, as indicated below.

2. Under the same conditions, calculate and show in a graph the total pitch moment magnitude as a function of wave frequency. Take the moment around the center of the body ($L/2$).

   The maximum pitch moment magnitude is 122 kNm at 1.45 rad/s. Explanation is similar to the case of vertical force, except that the zero frequency is now quite high; it is where the wavelength is equal to about sixty percent of the body length (not half!).

3. Now let the body be vertically oriented - e.g., it is a fixed piling. Using the **deep-water** wave formulas, calculate and show what is the magnitude of the total surge force on the piling, as a function of wave frequency.

   The maximum surge force magnitude seen is 16.7kN, at the maximum considered frequency of two radians per second. The forces get larger and larger because the velocities and accelerations at all depths both grow with frequency. This effect is stronger than the effect of the depth envelope getting narrower, over most of the frequency range. As above, also note that the wave amplitude is taken to be one meter for all frequencies.

4. Under the same conditions, calculate and show in a graph the pitch moment magnitude as a function of wave frequency. Take the moment around the lower end of the body.

   The maximum pitch moment magnitude seen is 292kNm, at the maximum considered frequency of two radians per second.

5. If the water depth is just twenty meters, for what range of frequencies is the analysis of the vertical body loading reasonable? In deep water, we approximate the wave velocity as $\sqrt{g/k}$, whereas in the general case it is $\sqrt{g \tanh(kh)/k}$, where h is the water depth.

   Figure 2 shows that $\sqrt{\tanh(kh)}$ *is close to one for all frequencies above one radian per second. Hence all of the maxima reported above are valid because they occur above this value, typical to SS 3. In practice, we have to design for higher sea states, which have lower frequency content as well as larger amplitudes; in these cases, we have to take into account the shallow-water formulae.*

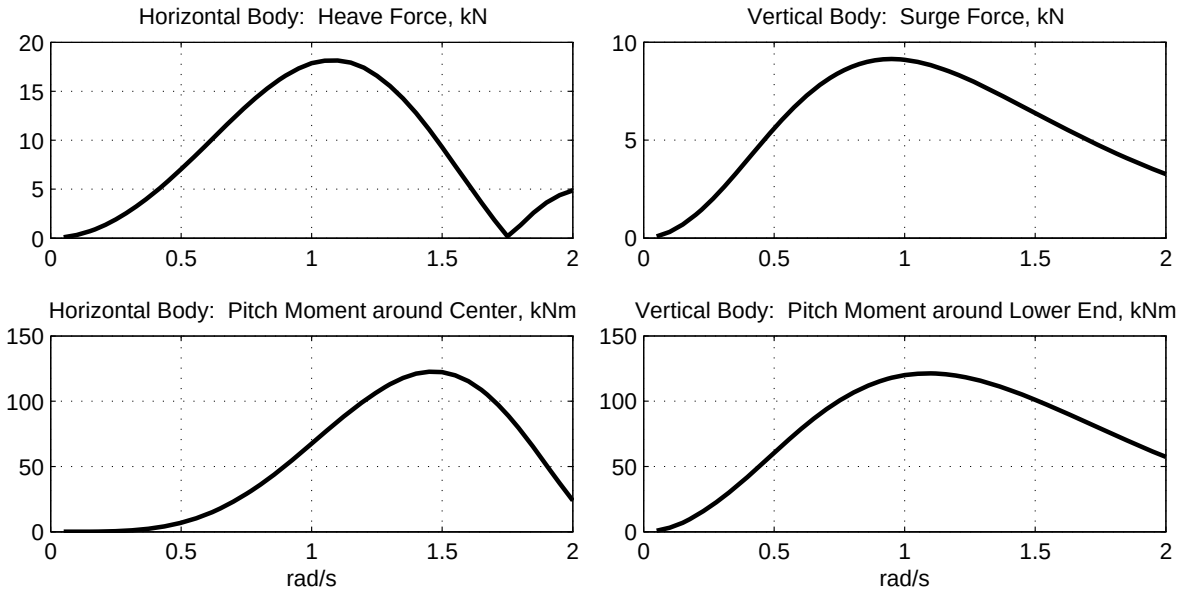


Figure 9: Force and moment on the horizontally-oriented body (left), and the vertically-oriented body (right).

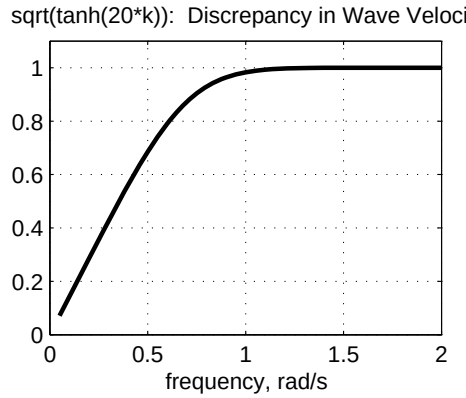


Figure 10: The deep-water approximation is reasonable when this function is near one.

```
%-----------------------------------------------------------
% Forces on a submerged body in waves

clear all;

d = 1 ; % cylinder diameter, m
L = 20.0 ; % cylinder length, m
Cd = 1.2 ; % Morison drag coefficient
T = 4 ; % depth of centerline, when the body is horizontal, m
dx = L/50 ; % differential length for strip theory calculations
g = 9.81 ; % gravity, m/s^2
rho = 1000 ; % water density, kg/m^3
A = 1 ; % wave amplitude, m

ntimes = 41 ; % number of time steps per cycle to study
wvec = .05:.05:2 ; % frequency vector, rad/s
xvec = dx/2:dx:L-dx/2 ; % body longitudinal location vector; centers

for i = 1:length(wvec) ; % loop through the frequencies of interest
    w = wvec(i) ;
    k = w^2/g ; % wave number
    tvec = 0:2*pi/w/(ntimes-1):2*pi/w ; % make up a one-cycle time
                                       % vector for this freq.

    for jj = 1:length(tvec), % loop through time
        t = tvec(jj) ;

        for j = 1:length(xvec) ; % loop through longitudinal locations
            x = dx*j ;
```

```
            % first get the vertical velocity, acceleration, and
            % Morison force
            % on the horizontal body; note the Archimedes force!
            e1 = exp(-k*T);
            v1 = A*e1*(-w)*sin(w*t-k*x) ; % local vertical velocity
            vv1 = A*e1*(-w^2)*cos(w*t-k*x) ; % local vertical acc.
            dFz(j) = 1/2*rho*Cd*d*dx*v1*abs(v1) + ...
                rho*(pi/4*d^2)*2*dx*vv1 ; % differential force

            % now get the horizontal velocity, acceleration, and
            % Morison force on the vertical body; note the
            % Archimedes force!
            e2 = exp(-k*(L-x+4)) ;
                % (move from the seafloor to the surface)
            v2 = A*e2*w*cos(w*t) ;
                % local horiz. veolcity (horiz. position is zero,
                % so kx=0)
            vv2 = A*e2*(-w^2)*sin(w*t) ; % local horiz. acceleration
            dFx(j) = 1/2*rho*Cd*d*dx*v2*abs(v2) + ...
                rho*(pi/4*d^2)*2*dx*vv2 ;

        end;
        Fz(jj) = sum(dFz); % net vertical force on horiz. body
        Momz(jj) = sum(dFz.*(xvec-L/2));
            % pitch moment around horiz. body mid-point
        Fx(jj) = sum(dFx) ; % net horizontal force on vertical body
        Momx(jj) = sum(dFx.*xvec);
            % pitch moment around lower end of vert. body
    end;

    Fzf(i) = max(abs(Fz)) ;
        % get the force and moment magnitudes over the whole cycle
    Momzf(i) = max(abs(Momz));
    Fxf(i) = max(abs(Fx)) ;
    Momxf(i) = max(abs(Momx));

end;

figure(1);clf;hold off;
subplot('Position',[.2 .5 .4 .2]);
plot(wvec,Fzf/1000,'LineWidth',2);
title('Horizontal Body:  Heave Force, kN');
grid;
```

```
subplot('Position',[.2 .2 .4 .2]);
plot(wvec,Momzf/1000,'LineWidth',2);
xlabel('rad/s');
title('Horizontal Body:  Pitch Moment around Center, kNm');
grid;

figure(2);clf;hold off;
subplot('Position',[.2 .5 .4 .2]);
plot(wvec,Fxf/1000,'LineWidth',2);
title('Vertical Body:  Surge Force, kN');
grid;
subplot('Position',[.2 .2 .4 .2]);
plot(wvec,Momxf/1000,'LineWidth',2);
xlabel('rad/s');
title('Vertical Body:  Pitch Moment around Lower End, kNm');
grid;

% display the specific maxima
disp('------------------------------------------------------------------');
disp('Horizontal body:');
[junk,ind] = sort(Fzf) ;
disp(sprintf('Maximum heave force magnitude:  %g N at %g rad/s.',...
    Fzf(ind(end)),wvec(ind(end))));
[junk,ind] = sort(Momzf) ;
disp(sprintf('Maximum pitch moment magnitude:  %g Nm at %g rad/s.',...
    Momzf(ind(end)),wvec(ind(end))));
disp('Vertical body:');
[junk,ind] = sort(Fxf) ;
disp(sprintf('Maximum surge force magnitude:  %g N at %g rad/s.',...
    Fxf(ind(end)),wvec(ind(end))));
[junk,ind] = sort(Momxf) ;
disp(sprintf('Maximum pitch moment magnitude:  %g Nm at %g rad/s.',...
    Momxf(ind(end)),wvec(ind(end))));

% compute and plot the test function for deep-water wave approximation
kvec = wvec.^2/g ;
figure(3);clf;hold off;
subplot('Position',[.2 .2 .3 .3]);
plot(wvec,sqrt(tanh(kvec*L)),'LineWidth',2);
xlabel('frequency, rad/s');
title('sqrt(tanh(20*k)):  Discrepancy in Wave Velocity');
axis([0 2 0 1.1]);
grid;
```

```
%-----------------------------------------------------------------------
```

43 Spectral Analysis to Find a Hidden Message

As you know, the fast Fourier transform (`fft()` in MATLAB), turns time-domain signals x into frequency-domain equivalents X. Here you will analyze a given signal from the course site and apply several of the common treatments so as to get clean power spectral densities.

1. First, download the x data file from the course site; it is called `computeSpectra.dat`. This signal x looks like a mess at first because of the noise, but we will tease out some specific information about X. Make a plot of the data versus time, assuming the time step is 0.0001 seconds. What features do you see if you zoom in on x, if any?

   It's an evidently narrowband signal with content around 1.6kHz (10krad/s), but probably with a lot of noise. Overall it is very hard to see anything beyond this.

2. Use the FFT to make the transformed signal X. Plot the magnitude of X versus frequency; recall that the frequency vector corresponding to X goes from zero to the roughly the sampling rate (use the instructions from your previous homework). In your plot, only show the frequency range near 10000 rad/s where there is significant energy, and remember the $2/n$ scaling that is needed to put FFT peaks into the same units as x. What do you see?

   See the top plot of the psd's. There are about fifteen peaks at frequencies slightly above 10 krad/s.

3. Treatment 1: Windowing. The Fourier transform operation on a finite-length signal x assumes periodicity, and so any discontinuity between the last and the first points in x is a feature that will be accounted for in X. In general we don't want this effect, because it muddies up the water and makes it harder to see the details in X that we are interested in. For this and some other reasons, it is common to multiply x by a windowing function, before taking the transform. There are many such windows, developed according to different metrics. If you do no windowing, this is called the boxcar window! An easy one we can consider here is the half-sine window, $w = \pi \sin(\pi[0:n-1]/(n-1))/2$, where n is the length of the signal. Multiply pointwise this window by x, and then plot the magnitude of the FFT, i.e., `abs(fft(x.*w))` in MATLAB. How does it compare with your psd from above?

   The window and the windowed data are plotted, the transform of this is the middle of the fft's. The fifteen peaks here might be better formed than for the raw data.

4. Treatment 2: Zero-padding. The frequency scale to go with X is defined by the sampling rate and the number of samples. To get really fine resolution in frequency, we would want to sample much faster or sample a lot longer. We can't fix either of these once the data is obtained of course, but we can lengthen the signal artificially, either by stacking it end to end against itself, or more commonly by adding a lot of zeros to the end of x. Do this for the windowed signal xw, with a block of $7n$ zeros added on the end. Again plot the magnitude of the resulting transform. Note that you will have to make up a new frequency vector for this case since you now have eight

times as many points, and ditto for your amplitude scaling of the transform. What do you see?

See the bottom plot of the fft's. The peaks now appear much more clearly because of the improved frequency resolution, and in fact they take amplitudes of about [1,2,3], and at very evenly spaced frequencies.

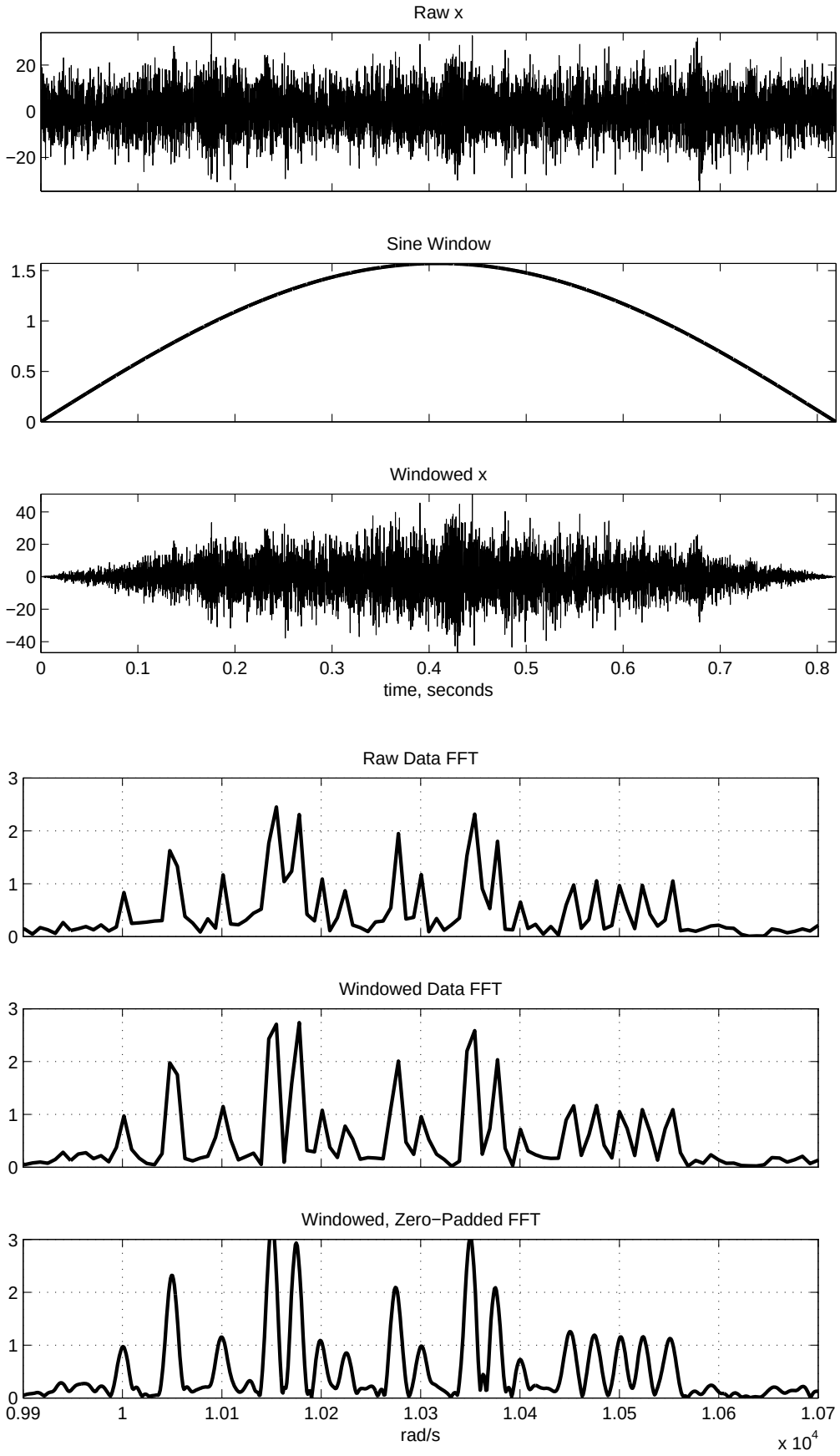
5. BONUS. The psd peaks you see encode a six-character message which you can only discover reliably if the processing above is done correctly. To **crack the code**, consider that each peak's magnitude indicates two bits of information, and thus that each set of four peaks encodes a full byte. The characters in the message are read left to right on the frequency axis; the left-most peak in each set of four gives the most significant bits so that, for example, a first peak height of three means the most significant bits of the character are [1,1] and the decimal equivalent is $128 \times 1 + 64 \times 1 = 192$. The first of 24 such two-bit peaks occurs right at $10000 rad/s$. Use the standard ASCII conversion table where [A-Z] corresponds with the decimal numbers 65-90. What is the message? All-caps, it is a single word you know!

   We read off the bit pairs that go with each peak starting at 10krad/s: [1020 1033 1102 1032 1011 1110]. These transform to the bytes [01001000 01001111 01010010 01001110 01000101 01010100] or in decimal form [72 79 82 78 69 84] or in characters [HORNET].

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simple Spectral Analysis

clear all;

message = 'HORNET'; % the message to be encoded in the psd
% get the most significant four bits and least significant four
% bits (decimal form)
for i = 1:length(message);
    word = dec2bin(message(i));
    llsb(i) = str2num(word(7)) + 2*str2num(word(6)) ;
    lsb(i) = str2num(word(5)) + 2*str2num(word(4)) ;
    msb(i) = str2num(word(3)) + 2*str2num(word(2)) ;
    mmsb(i) = str2num(word(1))  ;
end;
dec2bin(message)
[mmsb' msb' lsb' llsb']

n = 8193 ; % length of time series
basewn = 10000 ; % base frequency, rad/s
dwn = 25 ; % delta frequency between peaks
dt = .0001 ; % time step
nz = 7 ; % number zero padding blocks (of size n)
noise = 7 ; % noise standard deviation
```

```
npeaks = 4*length(message); % number of peaks - two per character of
    % message
wn = basewn + [0:dwn:(npeaks-1)*dwn] ; % frequencies of the peaks

wsamp = 2*pi/dt ; % sampling rate
t = 0:dt:(n-1)*dt ; % time vector

if 1, % load the data
    load computeSpectra.dat ;
    x = computeSpectra ;
    clear computeSpectra ;
else,   % or make up the signal:  noise plus all the elements for msb's
    % and lsb's
    x = randn(1,n)*noise ;
    for i = 1:length(wn)/4,
        x = x + mmsb(i)*sin(wn(4*i-3)*t+rand*2*pi) + ...
            msb(i)*sin(wn(4*i-2)*t+rand*2*pi) + ...
            lsb(i)*sin(wn(4*i-1)*t+rand*2*pi) + ...
            llsb(i)*sin(wn(4*i)*t+rand*2*pi) ;
    end;

    save computeSpectra.dat x -ascii
end;
figure(1);clf;hold off;
subplot(311);
plot(t,x) ;
title('Raw x');
axis('tight');
set(gca,'XTickLabel',[]);

fx = fft(x)*2/n; % here is the scaled fft
w = [0:n-1]/(n-1)*wsamp ; % and its corresponding frequency
figure(2);clf;hold off;
subplot(311);
plot(w,abs(fx),'LineWidth',2);
axis([basewn-4*dwn basewn+dwn*(4+npeaks) 0 max(max([msb lsb]))]);
grid;
title('Raw Data FFT');
set(gca,'XTickLabel',[]);

win = sin([0:n-1]/(n-1)*pi)*pi/2;
% the sine window, with scaling to preserve the energy in x
figure(1);
```

```
subplot(312);
plot(t,win,'LineWidth',2);
title('Sine Window');
set(gca,'XTickLabel',[]);
axis('tight');

xs = x.*win ; % apply the window to x
figure(1);
subplot(313);
plot(t,xs,'r');
axis('tight');
title('Windowed x');
xlabel('time, seconds');

fxs = fft(xs)*2/n; % scaled transform of the windowed data
figure(2);
subplot(312);
plot(w,abs(fxs),'r','LineWidth',2);
axis([basewn-4*dwn basewn+dwn*(npeaks+4) 0 max(max([msb lsb]))]);
grid;
set(gca,'XTickLabel',[]);
title('Windowed Data FFT');

z = zeros(1,nz*n); % the vector of zeros
xsw = [xs z]; % add zeros on to the windowed signal
ww = [0:((nz+1)*n)-1]/(((nz+1)*n)-1)*wsamp;
    % the new corresponding frequency vector
fxsw = fft(xsw)*2/n ; % scaled fft of the windowed, zero-padded signal
figure(2);
subplot(313);
plot(ww,abs(fxsw),'m','LineWidth',2);
grid;
axis([basewn-dwn*4 basewn+dwn*(4+npeaks) 0 max(max([msb lsb]))]);
title('Windowed, Zero-Padded FFT');
xlabel('rad/s');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```


Raw x
20
0
−20
Sine Window
1.5
1
0.5
0
Windowed x
40
20
0
−20
−40
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
time, seconds
Raw Data FFT
3
2
1
0
Windowed Data FFT
3
2
1
0
Windowed, Zero−Padded FFT
3
2
1
0
0.99
1
1.01
1.02
1.03
1.04
1.05
1.06
1.07
rad/s
x 10^4

44 Feedback on a Highly Maneuverable Vessel

The directional behavior of a highly maneuverable vessel is modeled by

$$\frac{\theta}{\delta} = \frac{0.1s + 1}{s^2 + 0.6s - 0.05}$$

where the rudder deflection is δ and the heading is θ. You note that because the transfer function denominator has a negative coefficient, the system is unstable without a control system. Also, the numerator shows that the hydrodynamic lift on the rudder has a component that scales rudder position as well as one that scales rudder rotation *rate*.

1. Find the *poles* (the roots of the denominator polynomial) of this system. Are the roots complex (indicating an oscillatory behavior), or real (indicating two exponential growth/decay modes)? At what rate does the system go unstable - that is, over what time does the response grow by a factor of e? (You can confirm this with a simulation of the system behavior from some nonzero initial condition.)

   The poles are given by the roots of the denominator polynomial equation in s, i.e., the roots of $s^2 + 0.6s - 0.05 = 0$. These are -0.674 and 0.074, both real. The negative one pertains to a stable mode whereas the positive one indicates an unstable mode. The time scale (or time constant) of the unstable mode is 1/0.074 or about 13.5 seconds. This is the amount of time it takes for the output to increase by a factor of $e = 2.718$. It is confirmed in an impulse response, for example, after the transient response of the first, stable mode has died away.

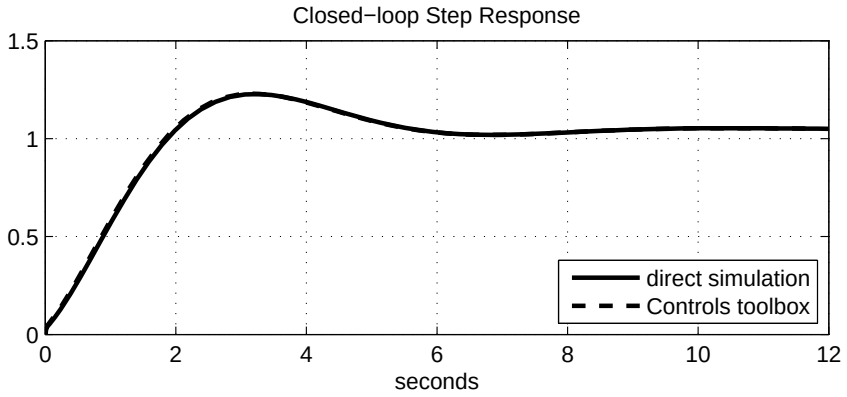
2. Design a proportional-derivative controller for the boat (PD), so as to achieve a closed-loop bandwidth of $\omega_c = 1$ rad/s and a damping ratio of $\zeta = 0.5$. The idea here is to choose k_p and k_d so as to put the closed-loop poles - that is, the roots of $1+P(s)C(s) = 0$ - at the specific locations $\omega_c\zeta \pm j\omega_c\sqrt{1-\zeta^2}$. The easiest way to carry this out is to write out the polynomial with k_d and k_p as free variables, and recognize that this same polynomial can be written as $s^2 + 2\zeta\omega_c s + \omega_c^2 = 0$. Some algebra will give you a set of two equations in two unknowns.

   The poles of the closed-loop system are to be the same as those of $s^2 + 2\zeta\omega_c s + \omega_c^2 = 0$; the closed-loop poles satisfy $1+P(s)C(s) = 0$, where $P(s)$ is the transfer function of the plant, and $C(s)$ is that of the controller. If $P(s) = n_p(s)/d_p(s)$ and $C(s) = n_c(s)/d_c(s)$ (numerators and denominators), then the condition $1 + P(s)C(s) = 0$ is the same as $d_P(s)d_C(s) + n_P(s)n_C(s) = 0$. Working this out, we obtain:

$$\begin{aligned} 0 &= s^2 + 0.6s - 0.05 + (k_p + k_d s)(0.1s + 1) \rightarrow \\ 2\zeta\omega_c &= \frac{0.6 + 0.1k_p + k_d}{1 + 0.1k_d} \quad \textit{and} \\ \omega_c^2 &= \frac{k_p - 0.05}{1 + 0.1k_d}. \end{aligned}$$

Setting ω_c and ζ to the desired values and solving this for the unknown variables we get $k_p = 1.082$ and $k_d = 0.324$.

3. Simulate the plant and controller, combined as a negative feedback system, in a step response, and show a plot. Note the step is applied to the commanded position; so you have to combine the plant and the controller, using $\delta = (k_p + sk_d)(\theta_{command} - \theta)$. Approximate the step function input as a ramp that rises over 0.02 seconds or longer (this will avoid artifacts of MATLAB's adaptive step-size algorithm). Does the response have the time scale and damping properties that you designed for?



The step response can be computed using calls to the MATLAB Control System Toolbox, but it is instructive to perform the calculations as a continuous-time simulation. After all, what is MATLAB doing?

The closed-loop dynamics are given as

$$\begin{aligned} (s^2 + 0.6s - 0.05)\theta(s) &= (0.1s + 1)\ \delta \\ \ddot{\theta} + 0.6\dot{\theta} - 0.05\theta &= \left(0.1\frac{d}{dt} + 1\right)\left(k_d\frac{d}{dt} + k_p\right)(r - \theta), \end{aligned}$$

where r is the reference heading. Collecting terms, we find

$$(1 + 0.1k_d)\ddot{\theta} + (0.6 + 0.1k_p + k_d)\dot{\theta} + (k_p - 0.05)\theta = 0.1k_d\ddot{r} + (0.1k_p + k_d)\dot{r} + k_p r.$$

The left-hand side looks like a regular mass-spring-damper, which will certainly have positive coefficients if the gains have been picked correctly. The right-hand side is more interesting because it involves the first and second derivatives of the commanded heading; for a step input in r, both of these derivatives are infinite! If r is a unit step, then $\dot{r}$ is a unit impulse, and $\ddot{r}$ is a (scaled) positive, then negative impulse.

A control action generated in the case of a step input would be impulsive here, and this is generally to be avoided in practice - unless some saturation occurs elsewhere in the system, or you know that the actuator can handle this kind of input. One solution is to never make r a step function, or even a ramp, so that $e = r - \theta$ always has two finite derivatives. A curve that is quadratic in time, for example, would suffice; so would the output of a separate second-order stable system, without any differentiation of the input (i.e., no factors of s in its transfer function numerator).

After all this, we still have to simulate the system one way or another. We extend a trick used in a prior question - the construction of an approximate impulse, of duration ϵ, and area one. This will be $\dot{r}$; its derivative will be $\ddot{r}$, and its integral will be r. Consider a very tall symmetric triangle with a peak at time $\epsilon/2$, and height $2/\epsilon$. The area under it is clearly one, and its derivative is $\pm 4/\epsilon^2$. With this technique, the plot shows that the step response computed via simulation matches that created by the Control System Toolbox.

Getting back to the question now, we expect a damped natural frequency in the closed loop of $\omega_d = \sqrt{1 - 0.5^2} = 0.866$, or a damped period of about 7.25 seconds. This is what we see in the figure. The damping ratio of 0.5 corresponds with an overshoot of around fifteen percent, and this is also confirmed by the step response, when the final value - which is not one - is taken into account.

4. After the transients of the step response have died out, what is the steady-state error?

   The final value can be computed by setting $s = 0$ in the closed-loop transfer function; this captures the effects at zero frequency, i.e., the steady-state after transients have died out. We have

$$\textit{final value} = \frac{k_p}{-0.05 + k_p} = 1.048.$$

   Clearly when k_p is large, this final value will be closer to one, as desired. As you know, however, gains cannot be turned up indiscriminately because the closed loop natural frequency will follow.

```
%-------------------------------------------------------------
% Feedback Control for Maneuvering

clear all;

global xepsilon kp kd n1 n0 d2 d1 d0;

% epsilon for computing step function; if we make this really small, the
% integrator will be forced to take tiny steps at the beginning; this
% conflicts with the defaults in MATLAB's adaptive stepsize routine.
```

```
xepsilon = .02 ;

% P numerator and denominator coefficients
n1 = .1 ;
n0 = 1 ;
d2 = 1 ;
d1 = .6 ;
d0 = -.05 ;

disp(sprintf('Open-loop poles: %g %g',roots([d2 d1 d0])));

% find the two gains for the specified crossover frequency and damping
% ratio
wc = 1 ;
z = .5 ;
mat = [2*z*wc*n1-n0 , -n1 ; wc^2*n1 , -n0] ;
vec = [d1 - 2*z*wc*d2 ; d0 - d2*wc^2] ;
vec = inv(mat)*vec ;
kd = vec(1) ;
kp = vec(2) ;
disp(sprintf('The gains needed are [kp kd] = [%g %g].', kp,kd));

% A.  here's the analysis using control system toolbox calls

% construct the plant and look at the unstable impulse response
sysP = tf([0 n1 n0],[d2 d1 d0]);
figure(1);clf;hold off;
impulse(sysP);

% close the loop and look at step response
sysC = tf([kd kp],[0 1]);
sysPC = sysC*sysP ;
sysCL = feedback(sysPC,1);
[y1,t1] = step(sysCL);

% B.  here's the simulation using regular integration

[t,y] = ode45('maneuveringFeedback_dxdt',[0 12],[0 0]) ;

figure(2);clf;hold off;
subplot('Position',[.2 .2 .6 .3]);
plot(t,y(:,2),t1,y1,'--','LineWidth',2);
grid;
title('Closed-loop Step Response');
```

```
xlabel('seconds');
legend('direct simulation','Controls toolbox',4);

%-----------------------------------------------------------------


%-----------------------------------------------------------------------
function [xdot] = maneuveringFeedback_dxdt(t,x) ;

global xepsilon kp kd n1 n0 d2 d1 d0 ;

% construct the artificial step input, with its two derivatives
if t < xepsilon/2,
    rddot = 4/xepsilon^2 ;
    rdot = 4/xepsilon^2*t ;
    r = 2/xepsilon^2*t^2 ;
elseif t < xepsilon,
    rddot = -4/xepsilon^2 ;
    rdot = 2/xepsilon - 4/xepsilon^2*(t-xepsilon/2) ;
    r = 0.5 + + 2/xepsilon*(t-xepsilon/2) - ...
        2/xepsilon^2*(t-xepsilon/2)^2;
else,
    r = 1 ;
    rdot = 0 ;
    rddot = 0 ;
end;

% (an option to the above is just to ignore the derivatives in the
% numerator!  For this, one would use r = 1, rdot = 0, rddot = 0 in
% the line below.  )

% compute the derivatives
xdot(1,1) = -(d1 + n1*kp + n0*kd)*x(1,1) - (n0*kp+d0)*x(2,1) + ...
        n1*kd*rddot + (n1*kp + n0*kd)*rdot + n0*kp*r ;
xdot(1,1) = xdot(1,1)/(d2 + n1*kd) ;

xdot(2,1) = x(1,1) ;

%-----------------------------------------------------------------------
```

2.017J Design of Electromechanical Robotic Systems
Fall 2009