Unified Engineering II **Spring 2005**

Problem S20 (Signals and Systems)

Do problems 7.1–7.4 in Oppenheim and Willsky. Note: The solutions are in the back of the book.

7.1. A real-valued signal $x(t)$ is known to be uniquely determined by its samples when the sampling frequency is $\omega_s = 10{,}000\pi$. For what values of ω is $X(j\omega)$ guaranteed to be zero?

7.2. A continuous-time signal $x(t)$ is obtained at the output of an ideal lowpass filter with cutoff frequency $\omega_c = 1{,}000\pi$. If impulse-train sampling is performed on $x(t)$, which of the following sampling periods would guarantee that $x(t)$ can be recovered from its sampled version using an appropriate lowpass filter?

(a) $T = 0.5 \times 10^{-3}$

(b) $T = 2 \times 10^{-3}$

(c) $T = 10^{-4}$

7.3. The frequency which, under the sampling theorem, must be exceeded by the sampling frequency is called the *Nyquist rate*. Determine the Nyquist rate corresponding to each of the following signals:

(a) $x(t) = 1 + \cos(2{,}000\pi t) + \sin(4{,}000\pi t)$

(b) $x(t) = \frac{\sin(4{,}000\pi t)}{\pi t}$

(c) $x(t) = \left(\frac{\sin(4{,}000\pi t)}{\pi t}\right)^2$

7.4. Let $x(t)$ be a signal with Nyquist rate ω_0. Determine the Nyquist rate for each of the following signals:

(a) $x(t) + x(t-1)$

(b) $\frac{dx(t)}{dt}$

(c) $x^2(t)$

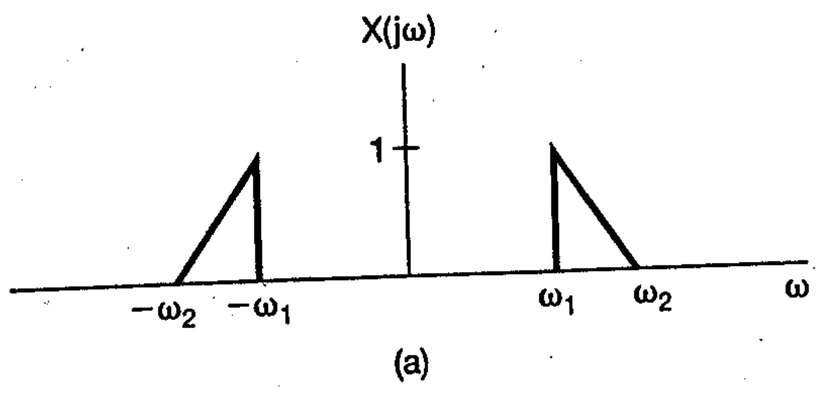
(d) $x(t)\cos\omega_0 t$

Unified Engineering II **Spring 2005**

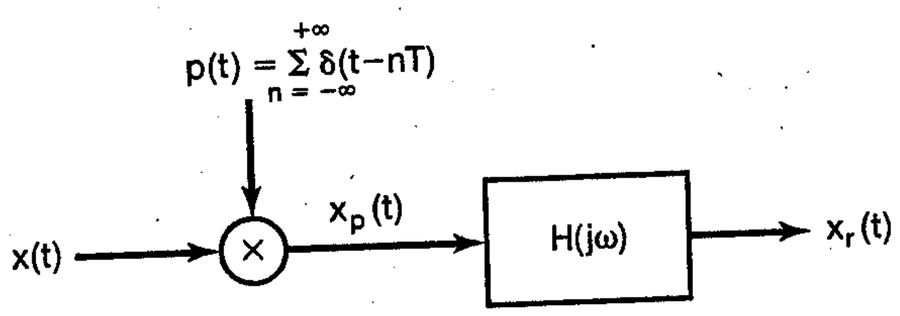
Problem S2 (Signals and Systems)

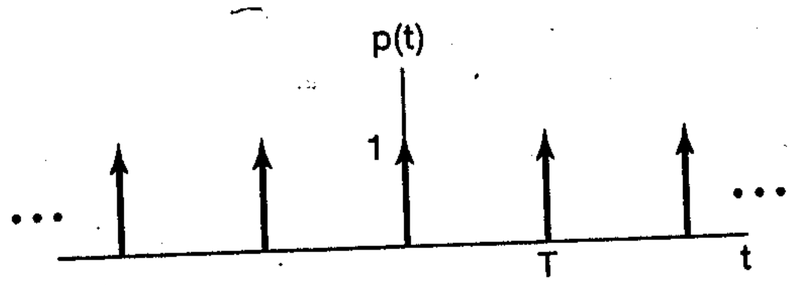
Do problem 7.26 in Oppenheim and Willsky.

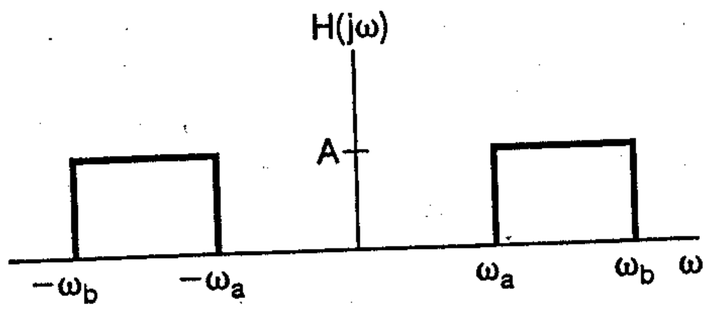
7.26. The sampling theorem, as we have derived it, states that a signal $x(t)$ must be sampled at a rate greater than its bandwidth (or equivalently, a rate greater than twice its highest frequency). This implies that if $x(t)$ has a spectrum as indicated in Figure P7.26(a) then $x(t)$ must be sampled at a rate greater than $2\omega_2$. However, since the signal has most of its energy concentrated in a narrow band, it would seem reasonable to expect that a sampling rate lower than twice the highest frequency could be used. A signal whose energy is concentrated in a frequency band is often referred to as a *bandpass signal*. There are a variety of techniques for sampling such signals, generally referred to as *bandpass-sampling* techniques.



(a)







(b)

Figure P7.26

To examine the possibility of sampling a bandpass signal as a rate less than the total bandwidth, consider the system shown in Figure P7.26(b). Assuming that $\omega_1 > \omega_2 - \omega_1$, find the maximum value of T and the values of the constants A, ω_a, and ω_b such that $x_r(t) = x(t)$.

Unified Engineering II **Spring 2005**

Problem S22 (Signals and Systems)

Consider the signal

$$g(t) = (1 + |t|)e^{-|t|}$$

1. Plot the signal. Do you expect the signal to have a "good" duration-bandwidth product, meaning that the product is close to the lower bound?

2. Find the duration of the signal, Δt.

3. Find the bandwidth of the signal, $\Delta\omega$. You may want to use the time domain formula for the bandwidth.

4. How close is the answer to the theoretical lower bound? Explain why the answer is or is not close to the bound.

Unified Engineering II **Spring 2005**

S23

Problem [redacted] (Signals and Systems)

Consider a pulse similar to the Loran-C pulse, given by

$$h(t) = t^3 e^{-t/\tau}\sigma(t)\sin(2\pi f t) = g(t)w(t)$$

where

$$g(t) = te^{-t/\tau}\sigma(t)$$
$$w(t) = \sin(2\pi f t)$$

(a) Find the *centroid* of the pulse envelope, given by

$$\bar{t} = \frac{\int t g^2(t)\,dt}{\int g^2(t)\,dt}$$

(b) Find the duration of the envelope, given by

$$\Delta t = 2\left(\frac{\int (t-\bar{t})^2 g^2(t)\,dt}{\int g^2(t)\,dt}\right)^{\frac{1}{2}}$$

(c)

$$\Delta\omega = 2\left(\frac{\int \dot{g}^2(t)\,dt}{\int g^2(t)\,dt}\right)^{\frac{1}{2}}$$

(d) How does the duration-bandwidth product compare to the theoretical minimum?

The problems in this problem set cover lectures C21, and C22

Download and unzip the following file from the C&P class webpage:

`C&P_PSet_Last.zip`

Some answers will need to have their answers zipped and uploaded to:

https://spacestation.mit.edu/unified/

Just kidding!! =)

Problem C21. (Hello World)[3]

This problem will help you explore the basics of tasking

With the `tasking_hello.adb` program:

a. Run the program several times. Is there a specific order in which the tasks start?

b. A delay statement is supposed to force a task to give up processor time for another task to execute. What happens when you delete the 'delay' statement?

c. We would like the tasks to be started in a specific order. This is possible by using the keyword "accept." Make the tasks start such that the first three outputs are:

"Hello from Task A"
"Hello from Task B"
"Hello from Task C"
…

d. Now we want to make sure that the tasks all execute A followed by B followed by C, continuously in a loop (This can be done with semaphores – but there is a simpler implementation. If you would like to play with semaphores, take a look at Lecture 21).

Problem C22. The Lego Printer

Tasking is VERY important in real-time applications, but can be difficult to understand because it is like trying to coordinate multiple people, dependent on each other, yet all doing their own things. In the Mars Rover assignment, tasking would have been very useful to allow one task to measure the spectral quality of the floor while another takes care of navigation. Check out the Lego Printer in the Unified Lounge (starting Wednesday afternoon through finals week). It uses several tasks to accomplish some very basic printing tasks.

Follow the instructions next to the printer to watch it print some basic repeating output. Below is a simplified excerpt of code that is running on the printer.

a. Task `Spool_Motor` currently starts even before the sensors are initialized. This is a problem because the rotation sensor will not be accurately keeping track of where the paper is. We would like `Spool_Motor` to start after the 'printer does some initialization of sensors.' Implement this delayed start by modifying the task. (Hint: do not use 'accept,' use the keyword 'new' in the main procedure)

b. When does the `Stop_Over_Spool` task start?

c. Task `Spool_Motor` implements a Floyd-Steinberg control of the motor, faking a PWM (a method for controlling motor RPM by turning it on and off quickly). Modify the body of this task so it will spin the motor in the opposite direction when variable `Spool_Motor_Speed` is negative.

Some useful subprograms to use are:

Integer=**abs**(Integer)
OnFwd(Motor_Name)

```
with Lego;
use Lego;

procedure Main is
   -- TASK 1: This task is designed to stop the printer from spooling
   -- paper continuously after it has spooled the paper through the printer

   task Stop_Over_Spool; -- Specification
   task body Stop_Over_Spool is -- Body
   begin
      while True loop
         if Spool_Rot_Sensor<-59
            -- -59 is the number of rotations needed to clear a sheet of paper
               Stopalltasks(); -- stop ALL tasks including the main program (only in Lego.ads)
         end if;
      end loop;
   end Stop_Over_Spool;


   -- TASK 2: This task is designed to fake a PWM - pulse width modulated
   -- motor signal. While Lego does implement a high frequency PWM, it is insufficient.

   -- Once the task has started, this variable can be changed to adjust motor speed.
   Spool_Motor_Speed : Integer := 0;

   --DO SOMETHING FOR (A) HERE

   -- Turns on the motor that spools the paper through the printer
   -- DO SOMETHING FOR (B) IN THIS TASK
   task body Spool_Motor is
   begin
      while True loop
           OnRev(S_Motor); -- turns the motor on in reverse
           Wait(Spool_Motor_Speed);
           Float(S_Motor);
           Wait(32 - Spool_Motor_Speed);
      end loop;
   end Spool_Motor;


   -- More code to take care of motor speed control follows. . .

begin
   -- Printer does some initialization of sensors

   --We start writing!
   --DO SOMETHING FOR (A) HERE
   Spool_Motor_Speed := 20;
   Wait(20);
   Spool_Motor_Speed := 30;
   Wait(20);
   -- more code is used to vary the speed of the spool motor and the
   -- motor that controls the print head.

end Main;
```