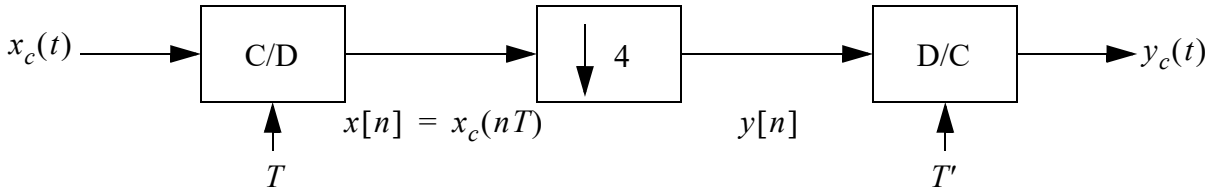
ECE 5650/4650 Exam II November 25, 2025 Name:

Take-Home **Exam Honor Code**

This being a take-home exam a **strict honor code** is assumed. Each person is to do his/her own work. Bring any questions you have about the exam to me. Please be clear and concise in your answers. You may use Python where appropriate. The exam is due via *Slack DM* no later than 5:00 PM Tuesday November 25, 2025 as a PDF on Slack.

1.) In the multirate system shown below, suppose $x_c(t)$ has Fourier transform such that $X_c(j\Omega) = 0, |\Omega| > 2\pi \times 2000$.



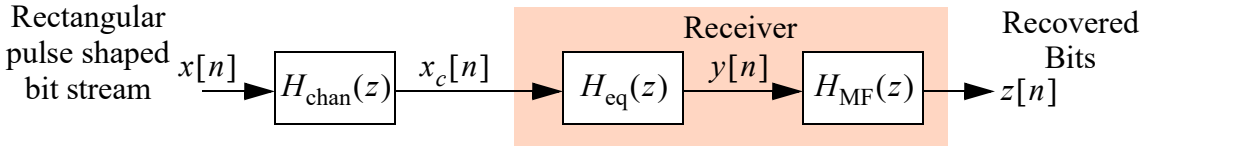
5 pts. a.) What value of T is required so that $X(e^{j\omega}) = 0, \pi/4 < |\omega| < \pi$.

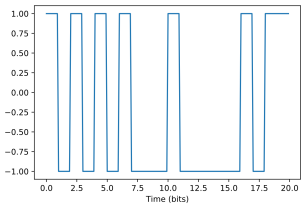
5 pts. b.) How should T' be chosen so that $y_c(t) = x_c(t)$?

15 pts. 2.) The transfer function of a multipath radio channel is FIR with system function

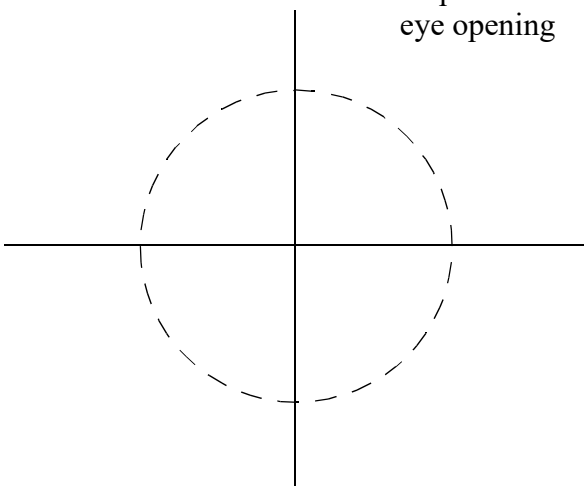
$$H_{\text{chan}}(z) = 1 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} = (1 + 1.2z^{-1})(1 + 0.3z^{-2}).$$

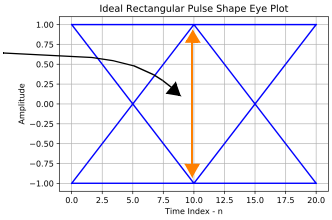
In order to correct for the magnitude distortion introduced by the channel, we wish to connect a **stable and causal** digital filter, characterized by a system function $H_{\text{eq}}(z)$, at the receiving end. Determine $H_{\text{eq}}(z)$ to remove the amplitude distortion. Note normally additive noise is present at the receiver input, but not considered here. A good starting point would be to plot the pole-zero pattern of $H_{\text{chan}}(z)$ then see how to design a stable and causal inverse.





The figure above uses Ns = 10 for illustration only





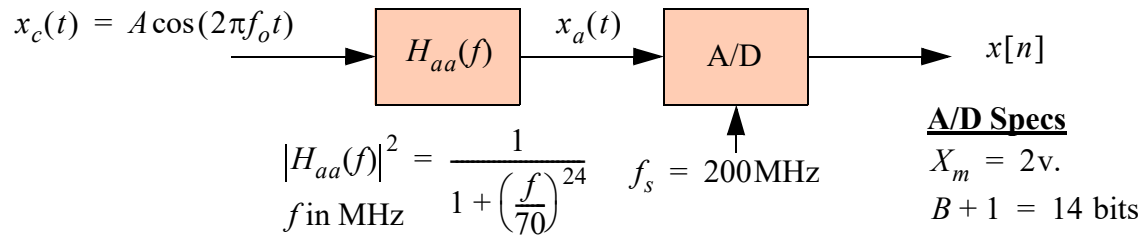
The figure above uses Ns = 10 for illustration only

To validate your design run the Python code below to see that without $H_{\text{eq}}(z)$ the eyeplot is only partially open, where as with the equalizer it again is fully open, as in the sample plot above. Three plots will be created: (1) ideal, no distortion channel, (2) with distortion, no equalizer, (3) the full system, distortion channel model, and equalizer. The matched filter (MF) is part of the signal processing to prepare the waveform for conversion back to recovered data bits, by sampling at the maximum eye opening (you learn about this in Comm 2).

```
h_chan = signal.convolve([1,1.2],[1,0,0.3])

Ns = 8 # Samples per bit used to create x[n]
# Clean signal
x,b,data = ss.nrz_bits(100,Ns,pulse='rect')
# Distorted signal
x_c = signal.lfilter(h_chan,1,x)
b_eq = ?
a_eq = ?
# Equalized signal
y = signal.lfilter(b_eq,a_eq,x_c)
# Before plotting apply the matched filter in b
ss.eye_plot(signal.lfilter(b,1,x),2*Ns,Ns-1)
title(r'Ideal Rectangular Pulse Shape Eye Plot')
ss.eye_plot(signal.lfilter(b,1,x_c),2*Ns,Ns)
title(r'Distorted Rectangular Pulse Shape Eye Plot')
ss.eye_plot(signal.lfilter(b,1,y),2*Ns,Ns)
title(r'Equalized Rectangular Pulse Shape Eye Plot')
```

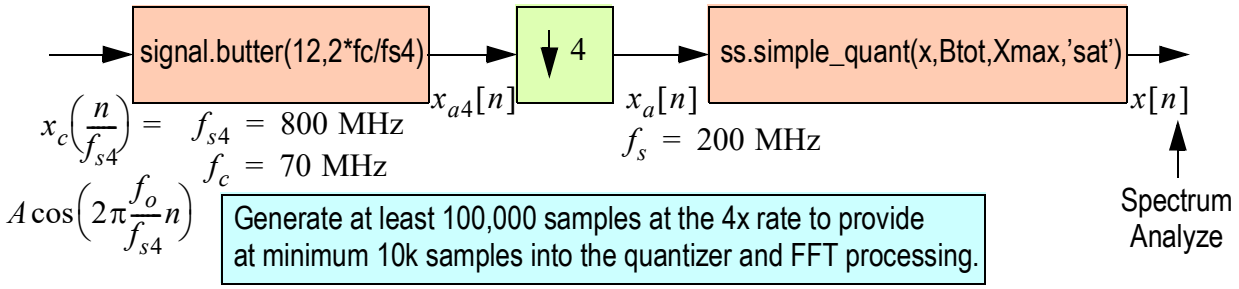
3.) Consider the following A/D digitizing system.



6 pts. a.) Suppose $A = 1.5$v. and $f_o = 50$ MHz. What is the signal-to-quantization noise ratio (SNR_q) at the A/D output $x[n]$.

6 pts. b.) Repeat part (a) with f_o increased to 90 MHz.

8 pts. c.) Build a behavioral level simulation model in Python and take measurements. To get a better feel for how quantization noise impacts real systems, here you develop a simple simulation model that crosses the boundaries between continuous and discrete-time systems. The simulation block diagram is shown below. Note 4x oversampling is used to model the



continuous-time portion of the system (left side). The downsampler and quantizer together form the ADC model. Implement this system to verify the change in dB (call it ΔdB) from the spectral peak at 50 and 90 MHz to the quantization noise floor surrounding the peak agrees with your earlier analysis. A calibration factor is however needed to compare your theory with the measurements:

$$\text{SNR}_{Q,\,\text{dB}} = \Delta\text{dB} - 10\log_{10}(\text{Nfft}/1.5) + 10\log_{10}(2)$$

Since the signal at the output of the quantizer has random characteristics you need a true power spectrum estimation function. I want you to use the function below for this purpose:

```
# Wrap ss.simple_sa() (used in Project 1 too) into an easier
# interface for periodogram averaging. Also include some additional scaling.
def simple_sa2(x,Nfft,fs,dB=True):
    """
    Sx = simple_sa2(x,Nfft,fs)

    Mark Wickert November 2014
    """
    Q = len(x)
    K = int(floor(Q/Nfft)) # max number of subrecords available
    f, Sx = ss.simple_sa(x,Nfft,Nfft,fs,K,'hann');
    w = signal.get_window('hann',Nfft,False)
    Sx /= sum(w**2)/Nfft # this normalized the window gain
    if dB == True:
        Sx = 10*log10(Sx)
    return f, Sx
```

To avoid some of the spurious signals (spectrum spurs), you will need to tweak the sinusoid frequencies a bit. You might start by trying 50.1234 MHz and 89.985 MHz. Your results will be two estimated values of SNR_Q in dB: one at 50 MHz and one at 90 MHz. Remember ΔdB is the change in dB from the spectrum peak to the noise floor surrounding the peak. If the noise floor has ripples in it, just use the average values of the noise floor in dB near the peak. Avoid spurs by tweaking the input sinusoid frequency. Your spectrum estimation calls should look something like:

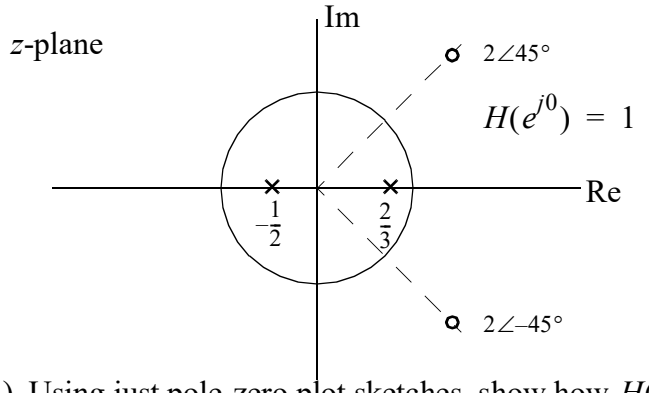
```
f200, Sx50d4q = simple_sa2(xa50d4q, 2**13, 200)
plot(f200,Sx50d4q)
```

15 pts. 4.) A finite impulse response filter is designed using `fir_remez_bpf()` found in the Python module `fir_design_helper.py`. The impulse response $h[n]$ is obtained as the coefficient array `b` to make a bandpass filter as follows:

```
import sk_dsp_comm.fir_design_helper as fir_d
b = fir_d.fir_remez_bpf(3200,4000,10000,10800,0.2,50,48000,n_bump=3)
```

The passband runs from 4000 Hz to 10,000 Hz, relative to a sampling rate of 48 kHz, and the stopband lies below 3200 Hz and above 10,800 Hz. The passband ripple is 0.2 dB and the stopband attenuation is 50 dB. Setting `n_bump = 3` fine tunes the stopband attenuation. Plot the magnitude response of the filter in dB and the pole-zero plot. The filter input is driven by a *Gaussian white noise process* having autocorrelation function $\phi_{ww}[m] = 8\delta[m]$. Find the theoretical output noise variance σ_y^2, where $y[n] = w[n] * h[n]$. **Hint**: To obtain this result consider the result of text Problem 2.92b. Check your work via a simple Python simulation that uses `w = sqrt(8)*randn(100000)` as the filter input. Then find `var(y)` at the filter output.

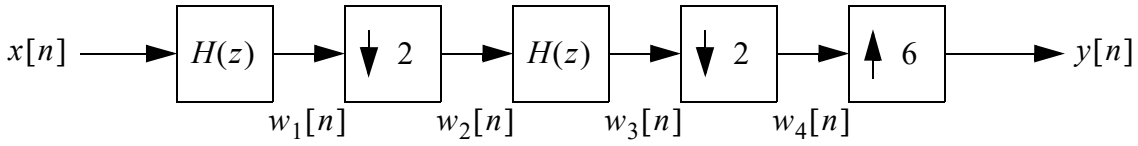
5.) An LTI system $H(z)$ has pole zero plot as shown below:

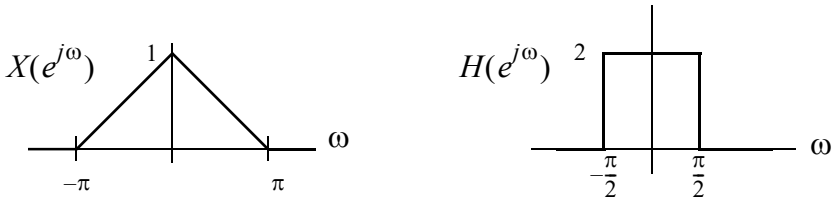


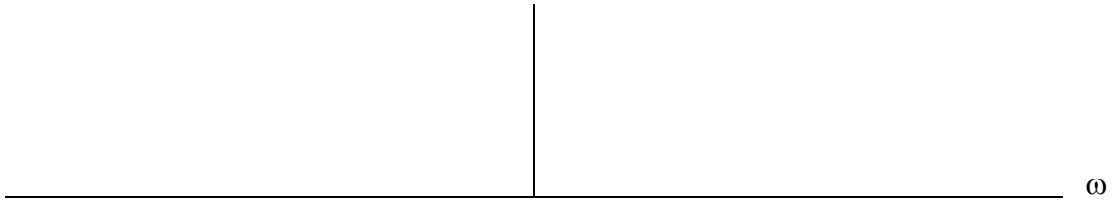
10 pts. a.) Using just pole-zero plot sketches, show how $H(z)$ can be expressed as a product of a minimum phase system in cascade with an allpass system, i.e., $H(z) = H_{\text{min}}(z)H_{\text{ap}}(z)$.

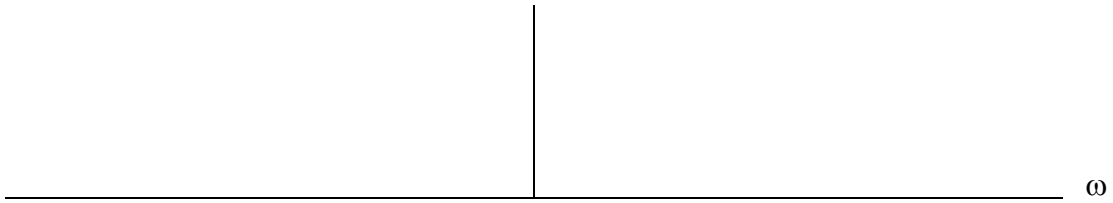
10 pts. b.) Write out the exact mathematical form for the cascade of $H_{\text{min}}(z)H_{\text{ap}}(z)$ using negative powers of z. Introduce a gain scale factor K so that the dc gain of the original system is unity.

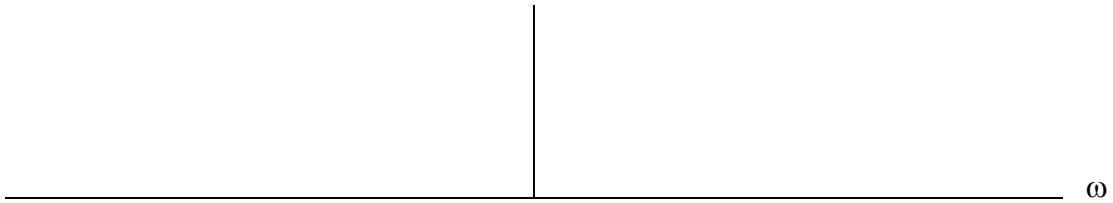
20 pts. 6.) Consider the multirate sampling system shown below. Carefully sketch the magnitude spectrum at all five locations beyond the input in the block diagram shown below.

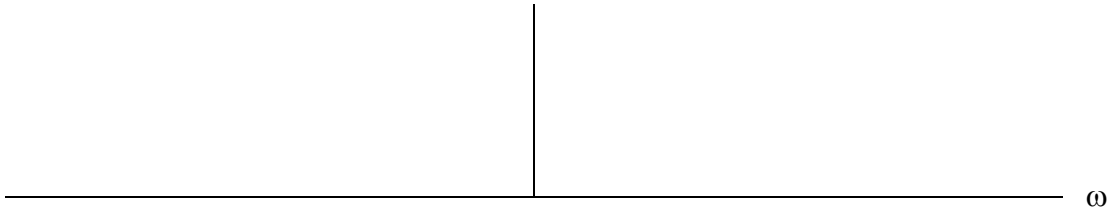


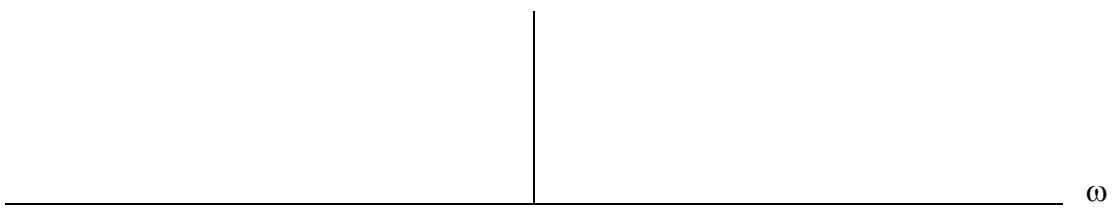












Comments

1.) This problem seem simple, but think things through to not make a careless error.

2.) To add some additional context, mobile communications suffers from multipath. A stem plot of the channel (FIR filter weights) reveals four impulses spread over the integer time axis. With just Ns = 8 samples per bit this channel impulse response *disperses* the digital communications waveform in time (so-called *delay spread*). The result is inter-symbol interference which following the matched filter results in *partial eye closure.* To learn more about eye plots see p5-39 here. The equalizer filter (**note** must be causal) tries to restore the waveform, but may in practice enhance the noise. The eye should open wider with the equalizer.

3.) Parts (a) and (b) of this problem are very similar to a homework Problem 3 from Set #5. The difference is that there is an antialiasing filter (7th-order Butterworth) at the input to the ADC, that will attenuate the input sinusoid depending upon the frequency of the sinusoid. You only need to be concerned with the amplitude at the filter output. The cutoff frequency is 65 MHz. In 3c notice that `signal.butter()` returns b and a coefficient arrays, since it is an IIR filter. Just load both b and a into `signal.lfilter()`. Very accurate results (< 1 dB) can be obtained by increasing the FFT length to $2^{13} = 8192$. This gives better frequency resolution for estimating the spectral peak. Scalloping loss is a concern[1].
FYI: To explain more about what is going on with the 3c calculation, the factor $10\log_{10}(2)$ accounts for the fact that 1/2 the total power is present in the visible spectral line, the other 1/2 is at the negative frequency which is not shown. The factor $10\log_{10}(\texttt{Nfft}/1.5)$ is due to the fact that the FFT (actually mathematically the DFT) acts as a bank of bandpass filters to the noise and signal. The signal has a discrete spectral line, so it's power passes through one of the bandpass filters (it may have *scalloping loss* unless `Nfft` is large, see notes Chapter 7), but the noise has a continuous spectrum and the power passed by each filter is registered as the filter *noise equivalent bandwidth*, for the hann window1.5/`Nfft`, times the noise spectral density. The hann window scales the `1/Nfft` noise bandwidth by 1.5 (see Notes Chapter 7 p. 7-65).

4.) In the analysis part of this problem just carefully evaluate the sum in the *hint*, that is use the formula

$$\sigma_y^2 = \sigma_w^2 \sum_{n=-\infty}^{\infty} h^2[n].$$

This formula is based on the discrete-time random process property for LTI systems that

$$\sigma_y^2 = \phi_{yy}[0] = \frac{1}{2\pi}\int_{-\pi}^{\pi} \Phi_{yy}(e^{j\omega})e^{j\omega n}d\omega\Big|_{n=0} = \frac{1}{2\pi}\int_{-\pi}^{\pi} \Phi_{ww}(e^{j\omega})\left|H(e^{j\omega})\right|^2 d\omega$$

$$= \frac{1}{2\pi}\int_{-\pi}^{\pi} \sigma_w^2\left|H(e^{j\omega})\right|^2 d\omega = \sigma_w^2 \sum_{n=-\infty}^{\infty} h^2[n]$$

since $\Phi_{ww}(e^{j\omega}) = \sigma_w^2$ for a white process. The last step follows from Parseval's theorem. In the Python portion of this problem I want you use `signal.lfilter()` to produce the filtered output $y[n]$.

5.) This problem is similar to a homework Problem of Set #6. See the examples I have posted next to the exam link: 5650Exam2_Examples.pdf. The hardest part about this problem is making sure the filter gain at $\omega = 0$ or $z = 1$ is unity as specified in the pole-zero plot. This just requires including a gain constant in the original $H(z)$.

6.) For a related problem examples see 5650Exam2_Examples.pdf.

---

1. http://en.m.wikipedia.org/wiki/Window_function