Homework Help - Problem Set 3

This area begins with general help/summary on coded OfDM.

Data-rate computation multiplies the (sub)symbol-rate by the total number of bits per (sub)symbol. For instance, if the sampling rate (for each of in-phase and quadrature) of a digital-video broadcast system has $T' = 109.375$ ns with $\bar{N} = 8192$ and a cyclic prefix that is 1/4 the FFT size, then the symbol period is

$$T = 8192(1 + \frac{1}{4}) \cdot T' = 10240 \cdot 109.375 \times 10^{-9} = 1.12 \times 10^{-3}$$

so the symbol rate is then 893 Hz. Thus, a 64 QAM system with no code ($r = 1$) would carry, with for example 6000 used tones,

$$R = 893 \cdot 6 \cdot 6000 \approx 32 \text{ Mbps}$$

Coded-OFDM usually has a limited set of "MCS" (Modulation Coding Scheme) options that basically are the code rates allowed for a set of constellations. Thus, the data rates R are basically found by multiplying r (code rate) by $\log_2(C)$ (number of bits in constellation) times the number of used tones times the symbol rate. If there were 4 codes and 5 constellations allowed, there would be 20 possible data rates.

With the MCS, the code will have certain free distance improvements that multiply the SNR so $SNR \rightarrow d_{free} \cdot SNR$ at the expense of a lower r for higher d_{free}. The MCS-style loading knows the distances for each of the codes, and has measured the SNR. Loading amounts to finding the code and d_{free} at this particular SNR that provides largest reliable data rate. Because there are only a few choices, simple trial-and-error over the MCS sets is usually sufficient. So for instance, if a Wi-Fi device (like a smart phone) moves closer to the Wi-Fi router, the channel gain presumably would improve. Improvement may then lead to a higher-data-rate MCS choice. For each channel gain g, there will thus be a best MCS choice. So at design time, a simple table of code/rate versus measured channel gain may suffice as long as the channel-gain distribution remains constant. In reality, the channel-gain distribution may vary and so a simple table is not sufficient and thus calculation of $< P_e >$ and/or P_{out} with selection of MCS to meet both criteria improves with respect to single fixed table.

Coded OFDM (4.13) This problem attempts a basic MCS loading process.

a. This part needs the symbol rate, number of USED tones per symbol and then multiplies by the possible number of bits in constellation reduced by the code rate. There are 9 possibilities.

b. Note $r = 1$ is not an option (this is uncoded and leaves the wireless system too susceptible to multipath fading notches). Evaluate a few options using the 64-state code and its known rates and distances.

c. The 6.3 dB was just enough to increase the data rate, but the granularity of the constellation (6 dB steps roughly) and this margin causes a significant data-rate loss.

d. This allows a yet higher data rate, but be careful on code choices.

e. Basically we're looking for the lowest data rate possible so smallest constellation (BPSK) and lowest rate code allowed. What is the data rate then in safe mode?

Discrete Loading (4.14) This is more elaborate MCS selection with a channel fading distribution.

a. This first part just asks you to write the g values (try organizing as row vector in matlab) and compute the average. Try to interpret in terms of averages and worst case SNRs.

b. The attached code here may be helpful, but you'll need to understand it to answer the question.

```
%% P4.14
clear all;clc;
gbounds = [0 .0105 .0223 .0357 .0511 .0693 .0916 .1204 .1609 .2303 .4];
gs = (gbounds(2:end)+gbounds(1:end-1))/2;
EsN0 = 10^4.3;
%EsN0 = 10^3.3;
% part a
disp('------ part a ------')
ave_g = mean(gs);
ave_snr = EsN0*ave_g;
ave_snr_db = 10*log10(ave_snr);
fprintf('<g>: %.4f, <SNR>: %.4f dB\n', ave_g, ave_snr_db);
M = [4 16 64];
% if consider the average snr
Pebar = 2*qfunc(sqrt(3*1*ave_snr./(M-1)));
% if consider the worst channel
Pebar = 2*qfunc(sqrt(3*1*gs(1)*EsN0./(M-1)));

% part b
disp('------ part b ------')
gaps_db = [9 3];
gaps = 10.^([.9 .3]);
```

```
for gap_idx = 1:2
    gap = gaps(gap_idx);
    for m = M
        for numG = 10:-1:1
            log_Kma_over_gap = 1/numG*(10*log2(m)-sum(log2(gs(11-numG:end))));
            if -log2(gs(11-numG))<log_Kma_over_gap
                Es = gap*(2^log_Kma_over_gap-1./gs(11-numG:end));
                meanEs = sum(Es)/10;
                margin=10*log10(EsN0/meanEs);
                fprintf('Gap: %d dB, M: %d, |Gstar|: %d, margin: %.4f dB\n', gaps_db(ga
                break;
            end
        end
    end
end
```

Ths script produces:

```
------ part a ------
<g>: 0.0992, <SNR>: 32.9656 dB
------ part b ------
Gap: 9 dB, M: 4, |Gstar|: 8, margin: 18.2580 dB
Gap: 9 dB, M: 16, |Gstar|: 10, margin: 10.2621 dB
Gap: 9 dB, M: 64, |Gstar|: 10, margin: 3.7684 dB
Gap: 3 dB, M: 4, |Gstar|: 8, margin: 24.2580 dB
Gap: 3 dB, M: 16, |Gstar|: 10, margin: 16.2621 dB
Gap: 3 dB, M: 64, |Gstar|: 10, margin: 9.7684 dB
```

c. Similarly this code may be helpful, but you'll need to understand it.

```
% part c
disp('------ part c ------')
for gap_idx = 1:2
    gap = gaps(gap_idx);
    for numG = 10:-1:1
        Kra = (10*EsN0+gap*sum(1./gs(11-numG:end)))/numG;
        if Kra>gap/gs(11-numG)
            Es = Kra - gap./gs(11-numG:end);
            meanEs = sum(Es)/10;
            b = 0.1*sum(log2(Kra*gs(11-numG:end)/gap));
            fprintf('Gap: %d dB, |Gstar|: %d, Echeck: %.4f, b: %.4f\n', gaps_db(gap_idx
            break;
        end
    end
end
```

with output

```
------ part c ------
Gap: 9 dB, |Gstar|: 10, Echeck: 19952.6231, b: 7.2237
Gap: 3 dB, |Gstar|: 10, Echeck: 19952.6231, b: 9.2014
```

MIMO Loading (4.22) Best Discrete loading adds an extra unit of information, which for SQ QAM restriction means two bits/tone (one bit in each dimension), simply adds bits in the positions of next least energy. This is the "greedy algorithm" that is used by Levin Campello. Thus, the incremental-energy table is just a way to enumerate how much energy needed for each additional unit on each tone. The better tones have smaller incremental energy until they get heavily loaded and each additional bit on them looks less attractive that putting a few bits on some of the lower-SNR tones.

SQ QAM is pretty easy and basically has (Γ in dB)

$$\mathcal{E}_1(2) = 2 \cdot \frac{10^{\Gamma/10}}{g_n} \left(2^2 - 1\right) \ ,$$

and thus

$$\begin{aligned} e_1(4) &= 4 \cdot \mathcal{E}_1(2) \\ e_1(6) &= 4 \cdot e_1(4) \\ e_1(8) &= 4 \cdot e_1(6) \ , \end{aligned}$$

and so on. Tables can be created from this and greedy applied.

a. A good way to work problems like this is to divide channels provided into the least-common divisor channel, replicating the wider bandwidth channels by repeated instances of a common bandwidth. (Hint, like four 10 MHz wide channels.) Use the general information above for the specifics of this problem.

b. Execute the Levin-Campello (Greedy) procedure now for RA (until no more energy can be added without violating the energy constraint).

c. Execute again, but his time for MA at desired data rate.

Binning (4.16) This binning problem 4.16 attempts to be self-explanatory through steps that lead to comparison of the sampled/learned distribution to random samples generated from a known distribution. Of course in practice, the distribution won't be known for comparison.

a. The inverse $P_e = 10^{-5}$ expression (any $\bar{P}_e$ could be reversed. For instance, 10^{-6} produces $10^{1.37}$), so do this generally. Evaluate also at the given $\bar{P}_e$ as the next part needs it.

b. We just use matlab to tabulate the various $3/(M-1) \cdot SNR$ values here. This creates a set of threshold SNRs, using the g found in Part a.

   so one can generate g values for potential bin interval endpoints from such an expression, substituting in the possible constellation sizes. For instance the matlab commands run through a set of values:

c. The data rates form from $r \cdot |C|$, running through r values and constellation sizes.

d. The reshape is g=reshape(SNR,[1,20]); Evaluation of the set of g's generated can form a distribution, for instance exponential distribution with

`pg=[exp(-.1*([0,g]))]-[exp(-.1*(g)),0]`

following

$$p_i = \int_{g_i}^{g_{i+1}} \frac{e^{-\frac{x}{\mathcal{E}_g}}}{\mathcal{E}_g} dx = e^{-g_i/\mathcal{E}_g} - e^{-g_{i+1}/\mathcal{E}_g} \quad .$$

that should sum to 1.

e. compute the various capacities and average them.

f. The cumulative distribution is formed by cumulative sums, $F(g) = 1 - e^{-g/\mathcal{E}_g}$: Basically plotting this versus the matlab g distribution. They're pretty close, but not identical.

g. It gets even closer with more samples.

Wi-FI Loading (4.15) The analog front-end noise of any wireless receiver simply adds the noise figure to the ambient psd (which at room temperature is -174 dBm/Hz). To find the power over a frequency range, simply add $10 \cdot \log_{10} W$ to the psd. Wider range (larger W) is larger power. The transmit power is 2 times the energy/real-dimension for complex signals, of course reduced by any channel attenuation to be used at the channel output.

The rest of Problem 4.15 uses similar calculations to the examples in the notes and in Problem 4.22.

a. Add the noise figure (in dB) to the thermal noise (-174 dBm/Hz) to get the actual receiver noise (or $\mathcal{N}_0$). Multiply (add in dB) by the bandwidth (positive freqs only) to get the noise power.

b. This classes SNR uses same bandwidth in numerator and denominator, or is equivalently a ratio of power-spectral densities. The 40 MHz wide SNR will be 3dB less than 20 MHz wide IF the POWER is fixed, as it is in this problem.

c. Take ratio of number of used tones.

d. Recall that $g = \frac{|h|^2}{\sigma^2}$ has a two-sided PSD in the denominator. Thus, if the PSD's in this problem are one-sided (so a factor of 2 higher than $\bar{\mathcal{E}}_x$) relative to the two-sided denominator PSD σ^2, so if one-sided noise psd's are used, then we need to increase the g value by 3 dB. Essentially, $g = \frac{|h|^2}{\mathcal{N}_0/2} = 2 \cdot \frac{|h|^2}{\mathcal{N}_0}$, and then now we can directly multiply by the one-sided PSD $\tilde{\mathcal{E}}_n$ to get a tonal SNR. Find the average of the distribution and set it equal to the $\mathcal{E}_g$, remembering to account for the 3 dB above.

e. This part involves computing performance for several r and d_{free} values. You might use the matlab commands like those in L5:12-13. This is repeating that process with some different numbers.

f. You should get a better use of power for the wider band here, so the 40 MHz solution is significantly higher.