Homework Help - Problem Set 5

[Turbo Design and Decodingl]

a. The transmit power is $P_x = \frac{\bar{\mathcal{E}}_x}{T}$ or the sum of the two quantities in dB. $\bar{\mathcal{E}}_x$ follows from the SNR and the noise power spectral density.

b. The capacity formula in bits per real dimension follows directly from SNR.

c. What is the gap Γ for $P_e = 10^{-6}$. The input constellation is restricted to $\bar{b} = 1$. Reverse the formulas $\bar{b} = \frac{1}{2} \cdot \log_2(1 + SNR/\Gamma)$ and $\bar{\mathcal{C}} = \frac{1/2}{\cdot} \log_2(1 + SNR)$ to obtain Γ from the known constellation and the known capacity. Coding gain γ will then be necessary to get that gap, recalling that coding gain adds fundamental gain and shaping gain. What is the shaping gain possible at $\bar{b} = 1$? (It is much less than 1.53 dB and in the text and lectures.)

d. You will need the the randi and sign functions as described in the problem statement directly. Also recommended are these functions: randn.m for the Gaussian noise; Comm.ErrorRate.m and errorStats.m , and probably biterr.m to run perhaps on one of the 1003-bit blocks. The AWGN command is

```
y= x+sqrt(10^(-SNR))*randn(1,1003);
uhat=0.5*(1+sign(y)); \% this is the ``uncoded'' decoder
```

to add noise at correct SNR level uncoded to the ± 1 input x. These functions work in the same way that they did in earlier lecture examples to track the error probability over many runs. You should have excellent agreement of theory and simulation result with the rng(7) seed.

e. Don't forget to release(errrate) to reset the error-rate counter. The SNR for a rate 1/2 code reduces by 3dB, even though the distance increases in proportion to d_{free}. Remember this 3B for your $r = 1/2$ selection and then basically rerun the previous simulation, but now with the vitdec.m replacing the simple "uncoded decoder" and of course using convenc.m to encode the 1003 input bits into twice as many output bits. You can compute the theoretical value of the codeword errors by using the correct $N_{d_{free}}, N_{d=d_{free}+1}, N_{d=d_{free}+2}$ for your code and of course the 3 smallest distances. Should be within 10% of each other.

f. Now, we take one step further and apply a rate 1/3 parallel turbo code. Remember to reset errrate again. You may need to release your turbo

code if you are running for a second time. The programs now needed are comm.TurboEncoder to set the structure and the corresponding comm.TurboDecoder. An interleaver order is also needed, so use randperm for this. With the turbo code, the number of input bits needs to be 1008 instead of 1003 (small bandwidth increase penalty that we can ignore. The SNR needs reduction, but this time corresponding to a rate 1/3 code. Again, if you use the proper N_b reduction factor for the ideal interleaver, the agreement between the simulation and the formula is pretty close (I saw roughly 10% agreement again) but used 100000 runs. More runs should produce yet tighter agreement, but takes too long. Throughout you should be seeing the error probability drop at each step and agreeing with theory. However, the deviation for the turbo code being larger is simply because to get accurate $\bar{P}_b$ estimates for very small values takes a lot of simulation runs. Don't be surprised if you get 0 errors on the turbo code.

[**Constraints and BICM**] This problem attempts to investigate BICM beyond the earlier PS4.5's simple decomposition into multiple parallel independent single-output-bit "AWGN-like" subchannels that essentially recovered the constellation expansion from uncoded in the corresponding parallel subcodes' intelligent constellation partitioning. The coding gain of an outer code (in that case a Hamming (7,4) code - PS3.4 , Text 2.5) essentially was retained with a square constellation.

a. The point chosen is close to a decision boundary for the surrounding 4 blue points. As long as $\epsilon > 0$, it's easy to see what point is closest for uncoded. However, when $\epsilon = 0$, there is a tie between 4 points. The symbol-level detector then just has to guess one of the 4, so what is the error probability if they're all equally likely? However, the surrounding points bits lead to a lower $\bar{P}_b$ that you should be able to find. Hint: with this gray code in this specific situation, all 3 bit-error rates are the same and less than the symbol error rate.

b. This is straightforward analysis of 8SQ QAM, which is not complex. For the bit-error rate, you need N_b for this uncoded 8SQ constellation, and look at the number of bit errors to nearest neighbors for any 8SQ point. It's not too difficult, there is a pattern there.

c. For this $\boldsymbol{y}$ and constellation, you should see ML and MAP being the same.

d. This part recognizes that matlab fails on its own trellis when $k > 1$, but there is enough information between Section 8.2 tables and the constellation's gray mapping on the coded 16SQ constellation to compute this anyway.

e. This is a bit tedious, and you have to compute 6 values for the 4 adjacent points. To assist you, one of them has the sample matlab calculation below is provided for cut-and-paste purposes - you'll need to change the values within but not otherwise need to get syntax correct:

```
p10=(2*pi*sig2)^(-1)*(  exp(-((1.9^2+.05^2)/(2*sig2))) +...
```

```
     exp(-((.1^2+1.95^2)/(2*sig2))) + exp(-((2.1^2+.05^2)/(2*sig2))) ) *.5;
p11=(2*pi*sig2)^(-1)*(exp(-((.1^2+2.05^2)/(2*sig2)))  )*0.5;
 LLR1=log(p10/p11)
```

f. For the gamma values - they exist only for each constellation point, so recognize there is much duplication here of γ values. You need compute only 4 values here and commands similar to the one in previous question are useful to copy/edit again. One should be much larger than the other 3.

g. This is fairly simple to do, but recognizes that the likelihood ratio is affected when the inputs are not equally likely. In this case, one of the extrinsics is sufficiently large to reverse the weakest single-bit decision.

[**LDPC Use**]

a. This follows from a graph in Section 8.3 pretty easily.

b. This is $\frac{k}{n} \cdot \frac{1}{T}$, so find all 3 quantities and compute.

c. For this, the class web site's get_h_matrix.m command is needed. You will need to provide proper inputs to the command in terms of parity, t_r, t_c , and the generic index value. Matlab's ldpc encoder and decoder programs need the last $n-k$ columns to be full rank, so the nonsinglastnk.m program at the web site needs to run on the output of the get_h_matrix program to ensure this result. You might use the command

```
H(end-9:end,end-9:end)
```

simply to ensure yourself that the submatrices look like shift matrices.

d. This part proceeds to encoding an input, which your can do using matlabs prbs(10,length) command. While the 462 input bits are divisible by 6, unfortunately it is the 529 output bits that need mapping onto 64SQ, so the extra 1 bit forces an almost dummy symbol that will use only two of its values (those two can be in one of the 32 subsets formed by partitioning the 64SQ.) You don't have to get fancy here with special last symbol - just treat it like the others except some of its bits are dummies - sometimes also known as "frozen" bits, not transmitted and known to the receiver.

e. The coding gain provided in Section 8.3's tables is actually for $\bar{P}_b$ so you can apply that to the formula $Q(\sqrt{SNR})$ directly. You can assume 1/2 the bits in the detected codeword are incorrect if one of them is in error as a worst case, so multiply by $n/2$ the bit-error rate.

f. The solutions use the rng(7) seed for all, but the agreement here is very close to theory. L10 has examples of the sequence of commands necessary to do this run.

Subsymbol- versus Symbol-Level Deterministic Interleaving This problem attempts to familiarize you with deterministic interleaving, which is used at the outer-most levels in concatenated coding systems.

a. The $G(D_{ss})$ is easily as a diagonal with $D_{ss}^{J-1)\cdot i}$ entries. For the $G(D)$, follow Section 8.6.1's examples to see where the various powers of D go.

b. The inverse needs to have nonegative powers of D to realize, so essentially the powers of D on any path all-the-way-through must add to a constant for a triangular interleaver.

c. Follow Section 8.6's examples. The delay should be constant at $(\mathcal{J}-1)\cdot(L-1)$ subsymbols. The subsymbol-based interleaver and de-interleaver are easier to draw and this part's focus.

d. Now, the more involved symbol-based interleaver. Follow Section 8.6's examples.

e. There is a slight difference between the two interleavers. $G(D)$ has the longer delay.

f. You should know that deterministic interleaving multiplies d_{free} by the depth.

Wireless Hard-Soft Challenge

a. This first part is uncoded so data rate is $R = b/T$.

b. This is direct application (in nominal state) of the well-known QAM P_e formula.

c. 99.9% of the time, it is the answer to the previous question, and .1% of the time it is the same formula application but with a very low SNR, but remembering that at some point random selection of an input creates an upper bound on the symbol-error probability.

d. Basically think how long the burst is and how many bits are then in error. It's clear the fading dominates, so this part is simple.

e. Here remember your bandwidth expansion - can you use it here for the unfaded state to basically ensure very low error rate for that part? Now, worst case look at the bursts and a simple code (like the 4-state 1/2 code) in terms of say a worst case in fading case of just hard decisions and correcting them. How long does the burst last, and how long is your survivor length in the code (which is equivalent to a block length). How many bits can you correct over this survivor length. Divide the survivor length into interburst period (and knowing your dfree) can lead to zero errors from the burst effectively and a low P_e on the remaining non burst.