

Homework Help - Problem Set 4

The first 2 problems take more effort than the latter 3 this week

[4.1 Extended Example] This encoder is the same as in the examples in class and in the text, but for an extra input bit that simply passes through and inverts the other two bits. Basically it attempts to gain familiarity with circuits, generators, and trellises by asking simple questions, and then provides short decoding effort with Viterbi detector. There is nothing particularly difficult here, just basics to get you comfortable by doing what you saw in lecture.

- a. The code rate r is simply the number of input bits k (on left in circuit diagram) divided by the number of output bits on the right. This is simple.
- b. Start by looking at bit u_2 's path through the circuit diagram as the upper left entry of the 2×3 generator $G(D)$. How many delays does it incur on the way to the upper output bit. Is anything added to it? This is the same code in the example in text as top row, but there will be a 0 for the new output bit. The lower row of $G(D)$ again sees if this input bit affects any (some, or all) output bits. As given in the problem, it passes through no delays so there should be no D 's in the bottom row.

The parity matrix essentially must add another column to the one for the text's example. What is the value in that new column? you must satisfy $G(D) \cdot H^t(D) = 0$ for both rows of $G(D)$ and the single row of $H(D)$. The first two entries can be those for the text's example. Thus, you have only one entry to find, the last one on right in $H(D)$. This should be easy - remember in $GF(2)$ that any $x + x = 0$, no matter what the x is (that is even if it includes powers of D).

- c. It may be simplest here to use the matlab commands. See the poly2trellis command and its use in L7's Binary Code Use section, and also plot-nextstates. Note that matlab's constraint length is the text's plus 1 for the poly2trellis inputs. The commands from there can be reused as long as you get the right poly2trellis (even if you want to reproduce the conventional trellis, which is not asked by this question). The output labeling is a little tricky in that matlab's trellis.outputs essentially provides the input bits in reversed order, so in each row the stack corresponding to bit u_1 being 0 and 1 (when $u_2 = 0$ respectively correspond to the first and third trellis.outputs values. Similarly the second stack for $u_2 = 1$

contains the second and fourth entries. When using `plotnextstates`, remember the input is `trellis.nextStates`, not `trellis`.

- d. It's fun to look at the trellis and get the d_{free} , which you can easily do, but since this code has parallel transitions (so each branch has one of two values), they must also be considered. Matlab's `distspec` program actually will take this into account, so it is a simply run to simply call `distspec(trellis,1)` where the 1 causes this program to produce only the `dfree` output.
- e. You can use the `convenc` function for the given input to produce the 9 output bits. Start in state zero, and it will end in state zero also. You should find that these two paths differ by the free distance, and they can never differ by less than that. Thus, what do you think about these two paths (all zeros is other path) relative to the ensemble set of paths that could be compared against all zeros - are some of them further away?
- f. The easiest way to work this is to use Viterbi decoding, and in particular the example `vitdec` examples in L8, to find the ML estimate. This is a relatively short span of bits compared to a survivor length of 5ν or longer, thus it is possible that the 1 bit error cannot yet be corrected. There is a way for you to verify this using the 'cont' program input instead of the 'hard' input.
- g. Basically we fix the previous question's issue here. You can put extra zeros in to fix the problem, so the survivor lengths are longer. You may want to review "tail biting" for this question to finish, but even that might not help. Think in terms of packet length of how to make this bandwidth loss small.

[4.2 Systematic Encoder] This problem continues your familiarity with convolutional codes by looking at systematic implementations, which are very often used in the field because if the decoder outputs the bits $v_{i,m}$, then it is simple to find the input bits (no inversion of a generator's sub matrices is necessary because they equal the input bits). It also explores how two different encoders (if over properly terminated sequences so all end in same state) produce the same ML sequence estimate that soft information/confidence can differ.

- a. Are k input bits equal to k output bits? (If not, the encoder is not systematic.) The rest follows easily by counting inputs and outputs to find the rate. Remember the number of states is equal to $2^{\text{number of delay elements}}$.
- b. The parity matrix is a little more challenging because it has two rows this time. $G_{min}(D)$'s first and third terms are almost equal so if you add them, only one term remains, but it's not quite equal to the middle term yet. However, supposed you multiple the first and third by D and add them? Does it look easy for one row possibly now? Repeat this trick now by looking at the 2nd and 3rd terms, suppose you add them, what is left? You may note that in $GF(2)$ that $1 + D^2 = (1 + D) \cdot (1 + D)$.
I don't ask you to check this, but this encoder is indeed minimal at 4 states (it cannot be produced with less than 4 states), If you are interested in how to check this, see Appendix B.

- c. Since this is a rate $1/n$ code, creating of a systematic encoder is easy. Just divide all elements by any, but best to divide by minimal polynomial that cannot be factored - for this code that appears on the right instead of left; however, without changing any code properties, we can reindex the subsymbol bits' order and swap $1 \leftrightarrow 3$ positions. Fortunately, as in class, matlab's bug does not affect $r = 1/n$ codes. The $k > 1$ for matlab to fail.
- d. Note that matlab has a convention that requires reading the nextStates information field to understand the outputs field. Because the next-state label swaps, for instance from 0 2 to then 2 0, this means for the second state on the left that the output corresponding to an input bit of 0 goes to the LOWER state. With this in mind, feel free now to use the plotnextstates.
- e. Try clearing the denominators of the two systematic entries in $A(D)$, and we're left with 1 in summing those two terms. The last can be zeroed.
- f. You should use convenc.m with the two encoders G_{min} and G_{sys} . They will not produce the same output. Why? Remember being the same code only applies that the overall sets of codewords are the same, whatever the mapping of input bits to codewords.
- g. You will need BSJR_BSC for this. The error can be

`[0 0 0 0 1 0 0 0 0 0 1 0 zeros(1,18)] .`

The error can be added to the convenc output using the xor function. You can convert soft output to hard decisions basically by

$$-\text{sign}(\text{BCJR_soft} + 1)/2 .$$

Two different codewords may correspond to the same inputs for different encoders, but when decoded, they should produce the same (hard) input decision from the same error pattern on the two different outputs; if one is correct, the other should be also. Soft information, however, can be different. So in one of these finite-length input soft decisions (so not really the full codeword as that has semi-infinite length) can provide more confidence than the other because BSJR minimizes bit-error probability. If there were decoding errors made, the confidence levels for different bits may be different in the two decoders for same bit. That does not happen in this problem.

- h. Here repeat the same encoder as a block-diagonal matrix with the correct two columns deleted, one from each. The rate of the code should be higher, and this is simple. You can use matlab and should expect free distance to reduce. You can do this for either G_{min} or G_{sys} . The solution does both.
- i. With the punctured code repeated, you'll need twice as many zeros as before to force all to zeros state. The decoder may be in trouble now with reduced free distance. Thus the BCJR outputs can disagree. The confidence where they differ may be helpful to note.

[4.3 No Free Lunch] This problem illustrates the fallacy of rate 1 binary codes.

- a. You should be good at this by now after first two problems, just no redundancy this time.
- b. As it turns out, the matlab programs will produce a trellis for the correct $G(D)$ inputs. Recall that the constraint length is specified for each row (kind of a bug, but no feedback here so ok, in matlab). Thus a power of D in a row of $G(D)$ corresponds in matlab to an input of 2 for that row.
- c. Yes, this coding gain really is greater than 0 dB. So far so good, but you are right to be worried, it's coming.
- d. semi-infinite sequence of 1's that starts at time 0, which has transform $1/(1 + D)$. Try on either or both inputs. What happens? This is not good and it is called a catastrophic encoder. The catastrophe is finite errors on output lead to infinite input errors.
- e. It should be obvious at this point that an infinite N_b means the P_e is so high that the Q function no longer matters, thus the coding gain is meaningless for a catastrophic encoder.
- f. Adding the extra independent bit basically causes the infinite-length input sequence now to force an infinite-length output sequence, so the catastrophe has been averted at the expense of a bandwidth loss. A hint here also is the rate has dropped below 1. However, even with $r < 1$, it is possible to have catastrophic encoders (they have too many states in them - incidentally, this is what matlab bug produces on the $k > 1$ encoders with feedback, a catastrophic encoder, which matlab's distspec program proceeds to inform you on the code matlab just designed itself, which is ironic.)
- g. Additionally, by looking at the new trellis, which is not required to draw but will help you, there are more output levels so it should be also clear there that this is a better code.

4.4 Exploring Power Bandwidth (again) We're pounding home now two weeks in a row the power-bandwidth concept. This is widely misunderstood by many practicing engineers (even some famous ones), so we're trying to ensure those through this class have it firmly understood.

- a. This is back to your basic $P_e = Q(\sqrt{SNR})$ calculation. A low SNR should correspondingly not have great uncoded P_e .
- b. Basically, for no more than 64 states, this asks you to read the Tables in Section 8.2. The correct table should have a code rate that corresponds to the doubling here.
- c. These tabulated codes are guaranteed to be minimal and non catastrophic, so indeed they have gains shown. The whopping reduction in P_e is real.
- d. So, the bandwidth power tradeoff is still good IF YOU CAN GET THE EXTRA BANDWIDTH, even when $\bar{b} < 1$.

4.5 Satellite This is a simpler basic design problem. The idea is that satellite channels are basically AWGNs and may allow bandwidth expansion by

- a. Basically, you get to increase the symbol rate but retain the same constellation. Thus, there is redundancy for coding at the higher symbol rate, but a loss in SNR. What is that loss as a function of r .
- b. This is standard QAM formula plugging here. It should be easy for you now.
- c. We will need coding gain to offset any SNR loss and indeed much of it to overcome the SNR loss and reduce the error probability. This is the $r \cdot d_{free}$ but now exploiting Gray Coding.
- d. Fortunately, the original SNR was not that deficient so a relatively small number of states and good code selection should work. Try 50% bandwidth expansion and there is a well-developed code in the lectures that should do it.
- e. You should be able to label the gray coding independently in the two dimensions of a QAM signal, so this should be easy. The circuit follows directly from the $G(D)$ that you chose.