MATLAB Implementation of Soft-Input Viterbi Decoding

Nikola Janjušević, EE '19 *The Cooper Union*

***Abstract*—A soft-input Viterbi decoding algorithm was implemented in MATLAB. Rate 1/3, 1/2, and 2/3 convolutional codes of the same free distance were compared in their error correction capabilities over a range of SNRs in an AWGN channel. The effect of the number of quantization bits for soft-decoding on the three codes is also compared.**

I. Introduction

THIS report details an implementation and brief analysis of the performance of soft-input Viterbi decoding over an AWGN channel, with various code-rates and quantization levels shown. Convolutional codes of the same free distance ($d_{free} = 10$) were chosen for a more even comparison between the codes. Messages were mapped with BPSK. The following sections detail implemented encoding, decoding, and the resulting simulations.

II. Implemented Encoding

The MATLAB built-in functions `poly2trellis` and `convenc` were used for encoding. The poly2trellis structure obtained was also used by the implemented decoding algorithm. Table I shows the parameters of the implemented convolutional codes, where K specifies the constraint length, M specifies the number of memory units used, and g_0, g_1 are the rows of the generator matrix in octal form. These codes were taken from Wicker's table of *Best Known Convolutional Codes* in ***Error Control Systems*** *for Digital Communication and Storage* (pg.285). The format used by Wicker was adapted to fit MATLAB's `poly2trellis`. All chosen codes have a free distance of $d_{free} = 10$.

TABLE I
USED CONVOLUTIONAL CODES

Rate	1/3	1/2	2/3
K	4	7	6
M	3	6	10
g_0	[13 15 17]	[13 31 71]	[31 46 63]
g_1	n\a	n\a	[32 65 61]

III. Implemented Decoding

The implemented soft-input Viterbi algorithm assumes a properly rotated and scaled incoming BPSK constellation. The desired number of quantized bits, q, then determines the conditional metrics. To form the metrics, the real-line is split into 2^q equal partitions on the interval of 80% of the (real) range of the received signal (non-uniform partitioning may be performed, however, it was not explored in this report). The conditional metric of the partition is then assigned by finding the conditional probability of a received symbol being within said partition. For BPSK, this means the conditional metric forms a $2\text{x}2^q$ matrix as each partition has two class conditionals of $+1$ and -1. With the partition intervals computed, the received codeword may have its symbols replaced with its respective quantized values. It is easy to see that hard-decoding is a special case where $q = 1$.

Following a computation of the conditional metrics and partition intervals, the standard Viterbi decoding algorithm is implemented. The implemented algorithm stores only the latest branch metrics and a traceback buffer of all the inputs (not states) leading up to each current state. No special consideration was taken to the starting and ending sequences of the codeword (where not all states are technically viable). Instead, at initialization, all branch metrics are set to negative infinity except the initial state, which is set to 0. This implicitly causes all survivor paths to lead back to the specified initial state. Then the classic add-compare-select operations of the Viterbi algorithm can be performed on all states of the system, regardless if they are actually possible. This incurs a minor cost compared to the processing time of the whole frame. As we are dealing with soft values, our branch metrics come from computing the conditional metric of the current soft-valued received word with a proposed input. The branch metrics and traceback buffer values for the current time-step are stored in temporary values until the computations for the current time-step are completed. At the end of decoding, the branch with the largest metric is chosen as the final survivor path, or alternatively (as implemented in the following simulations), a final state may be specified, where that branch is always chosen at the end. This is easily enforced by padding the input message with a sufficient length of zeros, to drive the encoder to the zero state.

IV. Simulation

Simulations of the rate 1/3, 1/2, 2/3 codes was performed for 500, 100, and 25 runs respectively with a frame size of 1000 symbols each time. Each code's BER was simulated over a range of SNRs $\in$ [-5,5], and at each SNR a decoding quantization of 1, 2, and 3 bits was simulated. Figure 1 shows the results of these simulations. SNR was translated to EbNO for Shannon Limit and coding gain comparison.

Looking at Figure 1, we predictably see that, for each code, the performance increases with the number of bits used

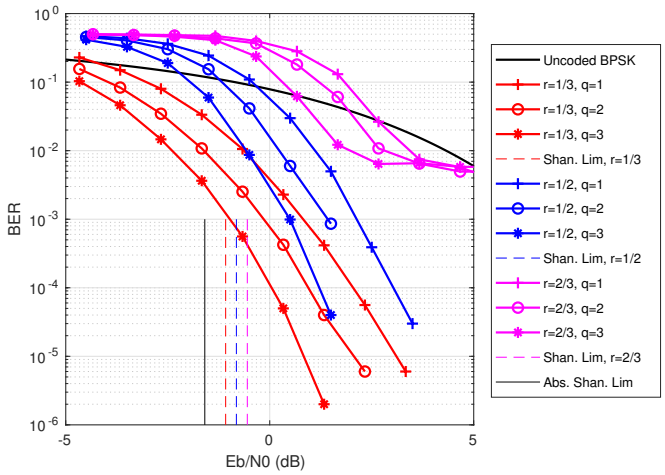

Fig. 1. BER vs. EbNo curve for BPSK mapped rate r convolutional codes with soft-input decoding quantized to q-bits. Shannon limits for each code rate, and uncoded bpsk are shown for reference.

for quantization. Interestingly, the rate 1/2 code is able to significantly out-perform the rate 1/3 code when using high quantization for soft-input. The rate 2/3 code seems to fail at the higher EbNo values. This is likely a simulation error (perhaps not enough runs), however, further investigation is needed to be certain.

V. Conclusion

The resulting simulations show the power of the convolutional codes, soft-input decoding, and the Viterbi algorithm. Soft-input decoding and calculation of branch metrics has been an educational exercise in MATLAB programing.