**Punctured Convolutional Coding**

Try This Example

[View MATLAB Command](matlab:openExample('comm/PuncturedConvCodingExample'))

This example shows how to use the convolutional encoder and Viterbi decoder System objects to simulate a punctured coding system. The complexity of a Viterbi decoder increases rapidly with the code rate. Puncturing is a technique that allows the encoding and decoding of higher rate codes using standard rate 1/2 encoders and decoders.

**Introduction**

This example showcases the simulation of a communication system consisting of a random binary source, a convolutional encoder, a BPSK modulator, an additive white Gaussian noise (AWGN) channel, and a Viterbi decoder. The example shows how to run simulations to obtain bit error rate (BER) curves and compares these curves to a theoretical bound.

**Initialization**

***Convolutional Encoding with Puncturing***

Create a rate 1/2, constraint length 7 [comm.ConvolutionalEncoder](https://www.mathworks.com/help/comm/ref/comm.convolutionalencoder-system-object.html) System object. This encoder takes one-bit symbols as inputs and generates 2-bit symbols as outputs. If you assume 3-bit message words as inputs, then the encoder will generate 6-bit codeword outputs.

convEncoder = comm.ConvolutionalEncoder(poly2trellis(7, [171 133]));

Specify a puncture pattern to create a rate 3/4 code from the previous rate 1/2 code using the puncture pattern vector [1;1;0;1;1;0]. The ones in the puncture pattern vector indicate that bits in positions 1, 2, 4, and 5 are transmitted, while the zeros indicate that bits in positions 3 and 6 are punctured or removed from the transmitted signal. The effect of puncturing is that now, for every 3 bits of input, the punctured code generates 4 bits of output (as opposed to the 6 bits produced before puncturing). This results in a rate 3/4 code. In the example at hand, the length of the puncture pattern vector must be an integer multiple of 6 since 3-bit inputs get converted into 6-bit outputs by the rate 1/2 convolutional encoder.

To set the desired puncture pattern in the convolutional encoder System object, hConvEnc , set the PuncturePatternSource property to Property and the PuncturePattern property to [1;1;0;1;1;0] .

convEncoder.PuncturePatternSource = 'Property';

convEncoder.PuncturePattern = [1;1;0;1;1;0];

***Modulator and Channel***

Create a [comm.BPSKModulator](https://www.mathworks.com/help/comm/ref/comm.bpskmodulator-system-object.html) System object to transmit the encoded data using binary phase shift keying modulation over a channel.

bpskMod = comm.BPSKModulator;

Create an [comm.AWGNChannel](https://www.mathworks.com/help/comm/ref/comm.awgnchannel-system-object.html) System object. Set the NoiseMethod property of the channel to Signal to noise ratio (Eb/No) to specify the noise level using the energy per bit to noise power spectral density ratio (Eb/No). When running simulations, test the coding system for different values of Eb/No ratio by changing the EbNo property of the channel object. The output of the BPSK modulator generates unit power signals; set the SignalPower property to 1 Watt. The system at hand is at the symbol rate; set the SamplesPerSymbol property to 1.

channel = comm.AWGNChannel('NoiseMethod', 'Signal to noise ratio (Eb/No)',...

'SignalPower', 1, 'SamplesPerSymbol', 1);

***Viterbi Decoding with Depuncturing***

Configure a [comm.ViterbiDecoder](https://www.mathworks.com/help/comm/ref/comm.viterbidecoder-system-object.html) System object so it decodes the punctured code specified for the convolutional encoder. This example assumes unquantized inputs to the Viterbi decoder, so set the InputFormat property to Unquantized .

vitDecoder = comm.ViterbiDecoder(poly2trellis(7, [171 133]), ...

'InputFormat', 'Unquantized');

In general, the puncture pattern vectors you use for the convolutional encoder and Viterbi decoder must be the same. To specify the puncture pattern, set the PuncturePatternSource property of the Viterbi decoder System object, hVitDec , to Property . Set the PuncturePattern property to the same puncture pattern vector you use for the convolutional encoder.

Because the punctured bits are not transmitted, there is no information to indicate their values. As a result, the decoding process ignores them.

vitDecoder.PuncturePatternSource = 'Property';

vitDecoder.PuncturePattern = convEncoder.PuncturePattern;

For a rate 1/2 code with no puncturing, you normally set the traceback depth of a Viterbi decoder to a value close to 40. Decoding punctured codes requires a higher value, in order to give the decoder enough data to resolve the ambiguities that the punctures introduce. This example uses a traceback depth of 96. Set this value using the TraceBackDepth property of the Viterbi decoder object, hVitDec .

vitDecoder.TracebackDepth = 96;

***Calculating the Error Rate***

Create an [comm.ErrorRate](https://www.mathworks.com/help/comm/ref/comm.errorrate-system-object.html) calculator System object to compare decoded bits to the original transmitted bits. The output of the error rate calculator object is a three-element vector containing the calculated bit error rate (BER), the number of errors observed, and the number of bits processed. The Viterbi decoder creates a delay in the output decoded bit stream equal to the traceback length. To account for this delay set the ReceiveDelay property of the error rate calculator System object to 96.

errorCalc = comm.ErrorRate('ReceiveDelay', vitDecoder.TracebackDepth);

**Stream Processing Loop**

Analyze the BER performance of the punctured coding system for different noise levels.

***Uncoded and Coded Eb/No Ratio Values***

Typically, you measure system performance according to the value of the energy per bit to noise power spectral density ratio (Eb/No) available at the input of the channel encoder. The reason for this is that this quantity is directly controlled by the systems engineer. Analyze the performance of the coding system for Eb/No values between 2 and 5 dB.

EbNoEncoderInput = 2:0.5:5; % in dB

The signal going into the AWGN channel is the encoded signal. Convert the Eb/No values so that they correspond to the energy ratio at the encoder output. If you input three bits to the encoder and obtain four bit outputs, then the energy relation is given by the 3/4 rate as follows:

EbNoEncoderOutput = EbNoEncoderInput + 10\*log10(3/4);

***Simulation loop***

To obtain BER performance results, transmit frames of 3000 bits through the communications system. For each Eb/No value, stop simulations upon reaching a specific number of errors or transmissions. To improve the accuracy of the results, increase the target number of errors or the maximum number of transmissions.

frameLength = 3000; % this value must be an integer multiple of 3

targetErrors = 300;

maxNumTransmissions = 5e6;

Loop through the encoded Eb/No values (the simulation will take a few seconds to complete).

BERVec = zeros(3,length(EbNoEncoderOutput)); % Allocate memory to store results

for n=1:length(EbNoEncoderOutput)

reset(errorCalc)

reset(convEncoder)

reset(vitDecoder)

channel.EbNo = EbNoEncoderOutput(n); % Set the channel EbNo value for simulation

while (BERVec(2,n) < targetErrors) && (BERVec(3,n) < maxNumTransmissions)

% Generate binary frames of size specified by the frameLength variable

data = randi([0 1], frameLength, 1);

% Convolutionally encode the data

encData = convEncoder(data);

% Modulate the encoded data

modData = bpskMod(encData);

% Pass the modulated signal through an AWGN channel

channelOutput = channel(modData);

% Pass the real part of the channel complex outputs as the unquantized

% input to the Viterbi decoder.

decData = vitDecoder(real(channelOutput));

% Compute and accumulate errors

BERVec(:,n) = errorCalc(data, decData);

end

end

***Compare Results to Theoretical Curves***

We compare the simulation results using an approximation of the bit error probability bound for a punctured code as per [ 1 ]. The following commands compute an approximation of this bound using the first seven terms of the summation for Eb/No values in 2:0.5:5. The values used for nerr are found in Table 2 of [ 2 ].

dist = 5:11;

nerr = [42 201 1492 10469 62935 379644 2253373];

codeRate = 3/4;

bound = nerr\*(1/6)\*erfc(sqrt(codeRate\*(10.0.^((2:.02:5)/10))'\*dist))';

Plot results. If the target number of errors or maximum number of transmissions you specify for the simulation are too small, the curve fitting algorithm might fail.

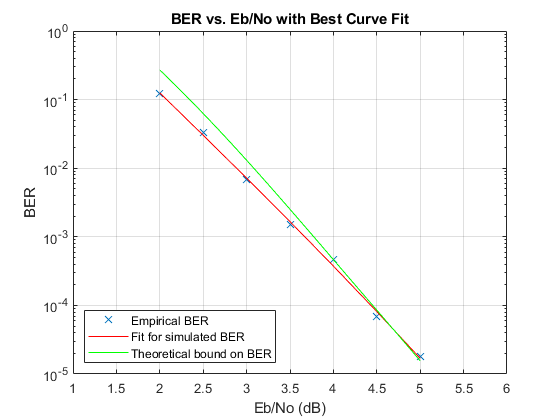
berfit(EbNoEncoderInput,BERVec(1,:)); % Curve-fitted simulation results

hold on;

semilogy((2:.02:5),bound,'g'); % Theoretical results

legend('Empirical BER','Fit for simulated BER', 'Theoretical bound on BER')

axis([1 6 10^-5 1])



In some cases, at lower bit error rates, simulation results appear to indicate error rates slightly above the bound. This results from simulation variance (if fewer than 500 bit errors are observed) or from the finite traceback depth in the decoder.

**Summary**

We utilized several System objects to simulate a communications system with convolutional coding and puncturing. We simulated the system to obtain BER performance versus different Eb/No ratio values. The BER results were compared to theoretical bounds.

**Selected Bibliography**

1. Yasuda, Y., K. Kashiki, and Y. Hirata, "High Rate Punctured Convolutional Codes for Soft Decision Viterbi Decoding," IEEE® Transactions on Communications, Vol. COM-32, March, 1984, pp. 315-319
2. Begin, G., Haccoun, D., and Paquin, C., "Further results on High-Rate Punctured Convolutional Codes for Viterbi and Sequential Decoding," IEEE Transactions on Communications, Vol. 38, No. 11, November, 1990, p. 1923