

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % All rights reserved by Krishna Sankar, http://www.dsplog.com
3  % The file may not be re-distributed without explicit authorization
4  % from Krishna Sankar.
5  % Checked for proper operation with Octave Version 3.0.0
6  % Author      : Krishna Sankar M
7  % Email       : krishna@dsplog.com
8  % Version     : 1.0
9  % Date        : 24 January 2010
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11
12 % Script for computing the BER for BPSK modulation in 3 tap ISI
13 % channel. Minimum Mean Square Error (MMSE) equalization with 7 tap
14 % and the BER computed (and is compared with Zero Forcing equalization)
15
16 clear
17 N = 10^6; % number of bits or symbols
18 Eb_N0_dB = [0:15]; % multiple Eb/N0 values
19 K = 3;
20
21 mH = 3; nH = 2^mH-1; kH = nH-mH; % Hamming (7,4)
22
23 ref = [0 0 ; 0 1; 1 0 ; 1 1 ];
24
25 ipLUT = [ 0  0  0  0;...
26           0  0  0  0;...
27           1  1  0  0;...
28           0  0  1  1 ];
29
30 for ii = 1:length(Eb_N0_dB)
31
32     % Transmitter
33     ip = rand(1,N)>0.5; % generating 0,1 with equal probability
34     s = 2*ip-1; % BPSK modulation 0 -> -1; 1 -> 0
35
36     % Channel model, multipath channel
37     nTap = 3;
38     ht = [0.2 0.9 0.3];
39     L = length(ht);
40
41     chanOut = conv(s,ht);
42     n = 1/sqrt(2)*[randn(1,N+length(ht)-1) + j*randn(1,N+length(ht)-1)]; % white gaussian noise,
% dB variance
43
44     % Noise addition
45     y = chanOut + 10^(-Eb_N0_dB(ii)/20)*n; % additive white gaussian noise
46
47     % Channel coding - block code
48     ip_bc = encode(ip,nH,kH,'hamming/binary'); % Hamming coding
49     ip_bc = reshape(ip_bc,1,size(ip_bc));
50     s_bc = 2*ip_bc-1; % BPSK modulation 0 -> -1; 1 -> 0
51     chanOut_bc = conv(s_bc,ht);
52     n_bc = 1/sqrt(2)*[randn(1,size(ip_bc,2)+length(ht)-1) + j*randn(1,size(ip_bc,2)+length
(ht)-1)]; % white gaussian noise, 0dB variance
53     y_bc = chanOut_bc + 10^(-Eb_N0_dB(ii)/20)*n_bc; % additive white gaussian noise
54
55     % Channel coding - convolutional coding, rate - 1/2, generator polynomial - [7,5] octal
56     ip_cc1 = mod(conv(ip,[1 1 1 ]),2);
57     ip_cc2 = mod(conv(ip,[1 0 1 ]),2);
58     ip_cc = [ip_cc1;ip_cc2];
59     ip_cc = ip_cc(:).';
60     s_cc = 2*ip_cc-1; % BPSK modulation 0 -> -1; 1 -> 0
61     chanOut_cc = conv(s_cc,ht);
62     n_cc = 1/sqrt(2)*[randn(1,size(ip_cc,2)+length(ht)-1) + j*randn(1,size(ip_cc,2)+length
(ht)-1)]; % white gaussian noise, 0dB variance
63     y_cc = chanOut_cc + 10^(-Eb_N0_dB(ii)/20)*n_cc; % additive white gaussian noise
64
65     % zero forcing equalization
66     hM = toeplitz([ht([2:end]) zeros(1,2*K+1-L+1)], [ ht([2:-1:1]) zeros(1,2*K+1-L+1) ]);
67     d = zeros(1,2*K+1);
68     d(K+1) = 1;
69     c_zf = [inv(hM)*d.'].';

```

```

70     yFilt_zf = conv(y,c_zf);
71     yFilt_zf = yFilt_zf(K+2:end);
72     yFilt_zf = conv(yFilt_zf,ones(1,1)); % convolution
73     ySamp_zf = yFilt_zf(1:1:N); % sampling at time T
74
75     % mmse equalization
76     hAutoCorr = conv(ht,fliplr(ht));
77     hM = toeplitz([hAutoCorr([3:end]) zeros(1,2*K+1-L)], [ hAutoCorr([3:end]) zeros(1,2*K+1-L) ]);
78     hM = hM + 1/2*10^(-Eb_N0_dB(ii)/10)*eye(2*K+1);
79     d = zeros(1,2*K+1);
80     d([-1:1]+K+1) = fliplr(ht);
81     c_mmse = [inv(hM)*d.'].';
82     yFilt_mmse = conv(y,c_mmse);
83     yFilt_mmse = yFilt_mmse(K+2:end);
84     yFilt_mmse = conv(yFilt_mmse,ones(1,1)); % convolution
85     ySamp_mmse = yFilt_mmse(1:1:N); % sampling at time T
86
87     % zero forcing equalization - block code
88     hM = toeplitz([ht([2:end]) zeros(1,2*K+1-L+1)], [ ht([2:-1:1]) zeros(1,2*K+1-L+1) ]);
89     d = zeros(1,2*K+1);
90     d(K+1) = 1;
91     c_zf = [inv(hM)*d.'].';
92     yFilt_zf_bc = conv(y_bc,c_zf);
93     yFilt_zf_bc = yFilt_zf_bc(K+2:end);
94     yFilt_zf_bc = conv(yFilt_zf_bc,ones(1,1)); % convolution
95     ySamp_zf_bc = yFilt_zf_bc(1:1:size(ip_bc,2)); % sampling at time T
96
97     % zero forcing equalization - convolutional code
98     hM = toeplitz([ht([2:end]) zeros(1,2*K+1-L+1)], [ ht([2:-1:1]) zeros(1,2*K+1-L+1) ]);
99     d = zeros(1,2*K+1);
100    d(K+1) = 1;
101    c_zf = [inv(hM)*d.'].';
102    yFilt_zf_cc = conv(y_cc,c_zf);
103    yFilt_zf_cc = yFilt_zf_cc(K+2:end);
104    yFilt_zf_cc = conv(yFilt_zf_cc,ones(1,1)); % convolution
105    ySamp_zf_cc = yFilt_zf_cc(1:1:size(ip_cc,2)); % sampling at time T
106
107    % mmse equalization - block code
108    hAutoCorr = conv(ht,fliplr(ht));
109    hM = toeplitz([hAutoCorr([3:end]) zeros(1,2*K+1-L)], [ hAutoCorr([3:end]) zeros(1,2*K+1-L) ]);
110    hM = hM + 1/2*10^(-Eb_N0_dB(ii)/10)*eye(2*K+1);
111    d = zeros(1,2*K+1);
112    d([-1:1]+K+1) = fliplr(ht);
113    c_mmse = [inv(hM)*d.'].';
114    yFilt_mmse_bc = conv(y_bc,c_mmse);
115    yFilt_mmse_bc = yFilt_mmse_bc(K+2:end);
116    yFilt_mmse_bc = conv(yFilt_mmse_bc,ones(1,1)); % convolution
117    ySamp_mmse_bc = yFilt_mmse_bc(1:1:size(ip_bc,2)); % sampling at time T
118
119    % mmse equalization - convolutional code
120    hAutoCorr = conv(ht,fliplr(ht));
121    hM = toeplitz([hAutoCorr([3:end]) zeros(1,2*K+1-L)], [ hAutoCorr([3:end]) zeros(1,2*K+1-L) ]);
122    hM = hM + 1/2*10^(-Eb_N0_dB(ii)/10)*eye(2*K+1);
123    d = zeros(1,2*K+1);
124    d([-1:1]+K+1) = fliplr(ht);
125    c_mmse = [inv(hM)*d.'].';
126    yFilt_mmse_cc = conv(y_cc,c_mmse);
127    yFilt_mmse_cc = yFilt_mmse_cc(K+2:end);
128    yFilt_mmse_cc = conv(yFilt_mmse_cc,ones(1,1)); % convolution
129    ySamp_mmse_cc = yFilt_mmse_cc(1:1:size(ip_cc,2)); % sampling at time T
130
131    % receiver - hard decision decoding
132    ipHat_zf = real(ySamp_zf)>0;
133    ipHat_zf_bc = real(ySamp_zf_bc)>0;
134    ipHat_zf_bc = decode(ipHat_zf_bc,nH,kH,'hamming/binary');
135    ipHat_zf_bc = reshape(ipHat_zf_bc,1,N);
136    ipHat_mmse = real(ySamp_mmse)>0;
137    ipHat_mmse_bc = real(ySamp_mmse_bc)>0;
138    ipHat_mmse_bc = decode(ipHat_mmse_bc,nH,kH,'hamming/binary');
139    ipHat_mmse_bc = reshape(ipHat_mmse_bc,1,N);
140    ipHat_zf_cc = real(ySamp_zf_cc)>0;
141    ipHat_mmse_cc = real(ySamp_mmse_cc)>0;

```

```

142
143 for kk = 1:2
144 % Viterbi decoding
145 pathMetric = zeros(4,1); % path metric
146 if (kk == 1)
147 survivorPath_v_zf = zeros(4,length(ySamp_zf_cc)/2); % survivor path
148 length_y = length(ySamp_zf_cc)
149 else
150 survivorPath_v_mmse = zeros(4,length(ySamp_mmse_cc)/2); % survivor path
151 length_y = length(ySamp_mmse_cc)
152 endif
153
154 for iii = 1:length_y/2
155 if (kk == 1)
156 r = ipHat_zf_cc(2*iii-1:2*iii); % taking 2 coded bits
157 else
158 r = ipHat_mmse_cc(2*iii-1:2*iii); % taking 2 coded bits
159 endif
160
161 % computing the Hamming distance between ip coded sequence with [00;01;10;11]
162 rv = kron(ones(4,1),r);
163 hammingDist = sum(xor(rv,ref),2);
164
165 if (iii == 1) || (iii == 2)
166 % branch metric and path metric for state 0
167 bm1 = pathMetric(1,1) + hammingDist(1);
168 pathMetric_n(1,1) = bm1;
169 survivorPath(1,1) = 1;
170
171 % branch metric and path metric for state 1
172 bm1 = pathMetric(3,1) + hammingDist(3);
173 pathMetric_n(2,1) = bm1;
174 survivorPath(2,1) = 3;
175
176 % branch metric and path metric for state 2
177 bm1 = pathMetric(1,1) + hammingDist(4);
178 pathMetric_n(3,1) = bm1;
179 survivorPath(3,1) = 1;
180
181 % branch metric and path metric for state 3
182 bm1 = pathMetric(3,1) + hammingDist(2);
183 pathMetric_n(4,1) = bm1;
184 survivorPath(4,1) = 3;
185
186 else
187 % branch metric and path metric for state 0
188 bm1 = pathMetric(1,1) + hammingDist(1);
189 bm2 = pathMetric(2,1) + hammingDist(4);
190 [pathMetric_n(1,1) idx] = min([bm1,bm2]);
191 survivorPath(1,1) = idx;
192
193 % branch metric and path metric for state 1
194 bm1 = pathMetric(3,1) + hammingDist(3);
195 bm2 = pathMetric(4,1) + hammingDist(2);
196 [pathMetric_n(2,1) idx] = min([bm1,bm2]);
197 survivorPath(2,1) = idx+2;
198
199 % branch metric and path metric for state 2
200 bm1 = pathMetric(1,1) + hammingDist(4);
201 bm2 = pathMetric(2,1) + hammingDist(1);
202 [pathMetric_n(3,1) idx] = min([bm1,bm2]);
203 survivorPath(3,1) = idx;
204
205 % branch metric and path metric for state 3
206 bm1 = pathMetric(3,1) + hammingDist(2);
207 bm2 = pathMetric(4,1) + hammingDist(3);
208 [pathMetric_n(4,1) idx] = min([bm1,bm2]);
209 survivorPath(4,1) = idx+2;
210
211 end
212
213 pathMetric = pathMetric_n;

```

```

214     if (kk == 1)
215         survivorPath_v_zf(:,iii) = survivorPath;
216     else
217         survivorPath_v_mmse(:,iii) = survivorPath;
218     endif
219
220 end
221 end
222
223 % trace back unit - ZF
224 currState = 1;
225 ipHat_zf_cc = zeros(1,length(ySamp_zf_cc)/2);
226 for jj = length(ySamp_zf_cc)/2:-1:1
227     prevState = survivorPath_v_zf(currState,jj);
228     ipHat_zf_cc(jj) = ipLUT(currState,prevState);
229     currState = prevState;
230 end
231
232 % trace back unit - MMSE
233 currState = 1;
234 ipHat_mmse_cc = zeros(1,length(ySamp_mmse_cc)/2);
235 for jj = length(ySamp_mmse_cc)/2:-1:1
236     prevState = survivorPath_v_mmse(currState,jj);
237     ipHat_mmse_cc(jj) = ipLUT(currState,prevState);
238     currState = prevState;
239 end
240
241 % counting the errors
242 nErr_zf(1,ii) = size(find([ip- ipHat_zf]),2);
243 nErr_zf_bc(1,ii) = size(find([ip- ipHat_zf_bc]),2);
244 nErr_zf_cc(1,ii) = size(find([ip- ipHat_zf_cc(1:N)]),2);
245 nErr_mmse(1,ii) = size(find([ip- ipHat_mmse]),2);
246 nErr_mmse_bc(1,ii) = size(find([ip- ipHat_mmse_bc]),2);
247 nErr_mmse_cc(1,ii) = size(find([ip- ipHat_mmse_cc(1:N)]),2);
248
249 end
250
251 simBer_zf = nErr_zf/N; % simulated ber
252 simBer_zf_bc = nErr_zf_bc/N; % simulated ber
253 simBer_zf_cc = nErr_zf_cc/N; % simulated ber
254 simBer_mmse = nErr_mmse/N; % simulated ber
255 simBer_mmse_bc = nErr_mmse_bc/N; % simulated ber
256 simBer_mmse_cc = nErr_mmse_cc/N; % simulated ber
257 theoryBer = 0.5*erfc(sqrt(10.^(Eb_N0_dB/10))); % theoretical ber
258
259 % plot
260 close all
261 figure
262 semilogy(Eb_N0_dB,simBer_zf(1,:), 'b+- ', 'Linewidth',2);
263 hold on
264 semilogy(Eb_N0_dB,simBer_mmse(1,:), 'ro-', 'Linewidth',2);
265 hold on
266 semilogy(Eb_N0_dB,simBer_zf_bc(1,:), 'g*- ', 'Linewidth',2);
267 hold on
268 semilogy(Eb_N0_dB,simBer_mmse_bc(1,:), 'bx-', 'Linewidth',2);
269 hold on
270 semilogy(Eb_N0_dB,simBer_zf_cc(1,:), 'ms-', 'Linewidth',2);
271 hold on
272 semilogy(Eb_N0_dB,simBer_mmse_cc(1,:), 'cd-', 'Linewidth',2);
273 axis([0 14 10^-5 0.5])
274 grid on
275 legend('sim-zf', 'sim-mmse', 'sim-zf-hamming', 'sim-mmse-hamming', 'sim-zf-convolutional', 'sim-
mmse-convolutional');
276 xlabel('Eb/No, dB');
277 ylabel('Bit Error Rate');
278 title('Bit error probability curve for BPSK in ISI with ZF & MMSE equalizer');
279 print("q4.jpg");

```