HOMEWORK #2 SOLUTIONS

(Note: Matlab code appears at the end.)

1. (a) Using the whiteness of $\{v(n)\}$,

$$\begin{aligned}
r_x(k) &= \mathrm{E}\{x(n)x(n-k)\} \qquad \text{where} \qquad x(n) = \sum_i b_i v(n-i) \\
&= \mathrm{E}\left\{\sum_i b_i v(n-i) \sum_l b_l v(n-k-l)\right\} \\
&= \sum_l \sum_i b_l b_i \underbrace{\mathrm{E}\{v(n-i)v(n-k-l)\}}_{\sigma_v^2 \delta(i-(k+l))} \\
&= \sigma_v^2 \sum_l b_l b_{k+l}
\end{aligned}$$

(b) Using the fact that $r_x(k) = r_x(-k)$,

$$\begin{aligned}
S_x(e^{j\omega}) &= \sum_{k=-\infty}^{\infty} r_x(k) e^{-j\omega k} \\
&= -r_x(0) + \sum_{k=0}^{\infty} r_x(k)\big(e^{-j\omega k} + e^{j\omega k}\big) \\
&= -r_x(0) + 2\sum_{k=0}^{\infty} r_x(k)\cos(\omega k).
\end{aligned}$$

Using $\sigma_v^2 = 1$ and the specified $\{b_i\}$, nonzero values of $r_x(k)$ are plotted in Fig. 1. The power spectral density (PSD) $S_x(e^{j\omega})$ was computed at 4096 uniformly-spaced values of ω in the interval $[-\pi, \pi]$ and plotted in Fig. 1.

(c) Matlab found that $\mathrm{SFM}_x = \boxed{0.3400}$ and $\sigma_e^2\big|_{\min, N\to\infty} = \boxed{0.9992}$ where

$$\begin{aligned}
\mathrm{SFM}_x &= \frac{\exp\left(\frac{1}{2\pi}\int_{-\pi}^{\pi} \ln S_x(e^{j\omega})d\omega\right)}{\frac{1}{2\pi}\int_{-\pi}^{\pi} S_x(e^{j\omega})d\omega} \\
\sigma_e^2\big|_{\min, N\to\infty} &= \exp\left(\frac{1}{2\pi}\int_{-\pi}^{\pi} \ln S_x(e^{j\omega})d\omega\right)
\end{aligned}$$

The results suggest that $\sigma_e^2\big|_{\min, N\to\infty} = \sigma_v^2$, as expected from the notes.

(d) The following predictor coefficients are found via $\mathbf{h} = \mathbf{R}_N^{-1}\mathbf{r}_x$:

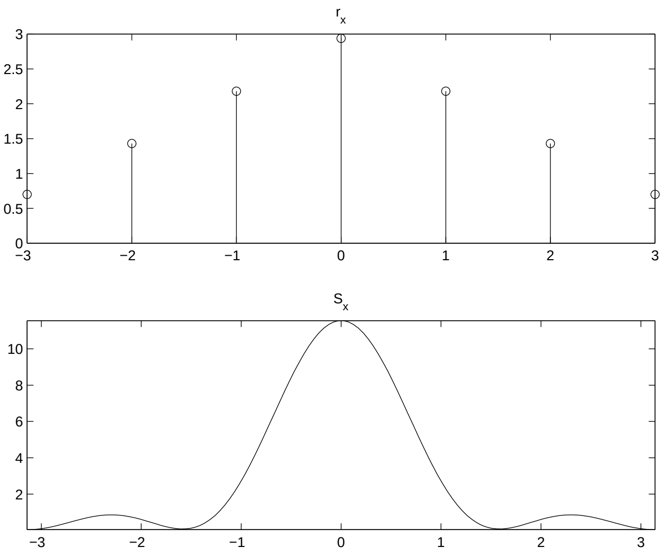

Figure 1: Nonzero values of $r_x(k)$ versus k, and $S_x(e^{j\omega})$ versus ω.

$h_i : N = 3$	$h_i : N = 20$
0.8420	0.8985
-0.0088	-0.0101
-0.1562	-0.0111
	-0.6052
	0.5580
	-0.0127
	-0.0134
	-0.3592
	0.3385
	-0.0117
	-0.0120
	-0.2021
	0.1923
	-0.0090
	-0.0089
	-0.0957
	0.0872
	-0.0052
	-0.0050
	-0.0145

The prediction error variance $\sigma_e^2\big|_{\min,N} = r_x(0) - \mathbf{r}_x^t \mathbf{R}_N^{-1} \mathbf{r}_x$ was found to be $\boxed{1.2656}$ for $N = 3$ and $\boxed{1.0036}$ for $N = 20$.

(e) Based on the derivation in (a), $r_e(k)$ can be written

$$r_e(k) \;=\; \sigma_v^2 \sum_l q_l q_{k+l} \qquad \text{where} \qquad q_l \;=\; \{b_0, b_1, b_2, b_3\} * \{1, -h_1, -h_2, \ldots, -h_N\}$$

since the impulse response of the linear system taking $v(n)$ to $e(n)$ equals the convolution of the input-coloring impulse response $\{b_i\}$ and the DPCM impulse response $\{1, -h_1, -h_2, \ldots, -h_N\}$. Calculating $S_e(e^{j\omega})$ as before, $r_e(k)$ and $S_e(e^{j\omega})$ are plotted in Fig. 2 for the case $N = 3$ and Fig. 3 for the case $N = 20$. Since a white sequence is characterized by (i) a Kronecker-delta autocorrelation and (ii) a flat PSD, we see that the error sequences are not perfectly white, though nearly-white for $N = 20$.

The "flatness" of a spectrum may be quantified using the spectral flatness measure, which in these cases can be calculated as $\text{SFM}_e = \boxed{0.7899}$ for $N = 3$ and $\text{SFM}_e = \boxed{0.9964}$ for $N = 20$. Since the maximum value of SFM is one, we confirm that the $N = 20$ error spectrum is, indeed, very close to white.

(f) Using a 10000-length version of $x(n)$, experimental values of $\sigma_e^2\big|_{\min,N}$ were found to be $\boxed{1.2655}$ for $N = 3$ and $\boxed{0.9879}$ for $N = 20$. These agree reasonably well with the theo-

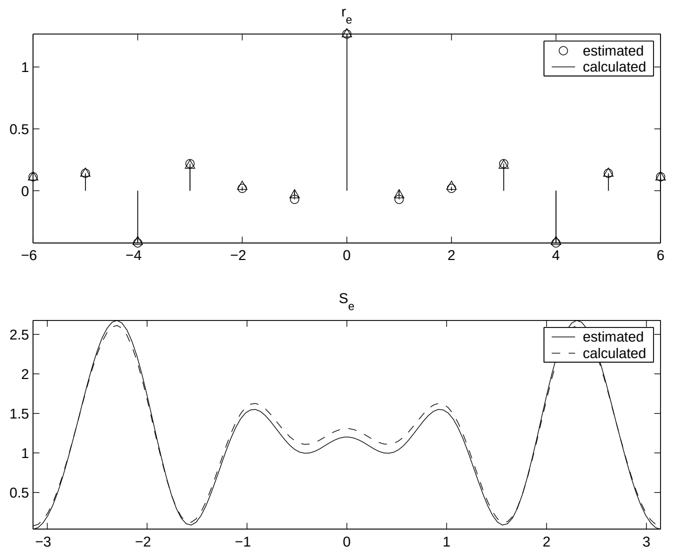

Figure 2: Nonzero values of $r_e(k)$ versus k, and $S_e(e^{j\omega})$ versus ω, for $N = 3$.

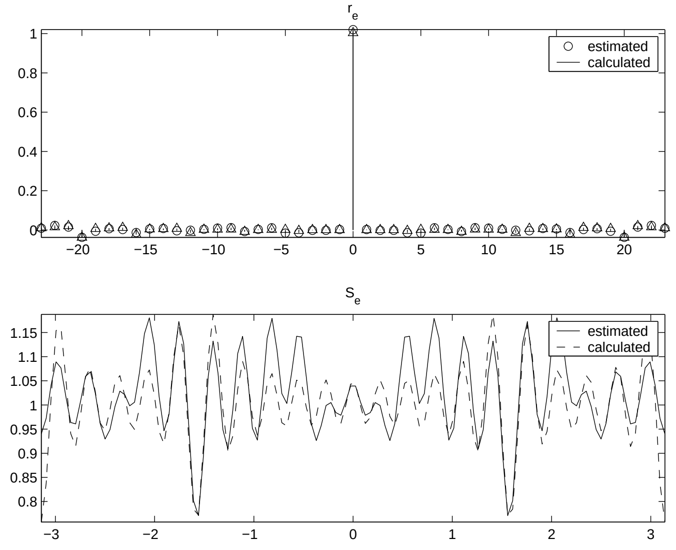

Figure 3: Nonzero values of $r_e(k)$ versus k, and $S_e(e^{j\omega})$ versus ω, for $N = 20$.

retical values. In addition, experimental versions of $r_e(k)$ and $S_e(e^{j\omega})$ were determined using `xcorr(x,x,max_lag,'unbiased')` and plotted in Fig. 2 and Fig. 3.

2. (a) The mean-squared reconstruction error (MSRE) of the Prediction Error Transmission system was essentially zero: Matlab returned $\mathcal{E} = \boxed{1.6451 \times 10^{-32}}$.

   (b) The MSRE of the PCM system (using $L = 32$ uniform quantization with $\Delta = 6\sigma_x^2/L$) was $\mathcal{E} = \boxed{0.0100}$. The choice of Δ was based on the assumption $x_{\max} = 3\sigma_x^2$, which was found in Homework 1 to minimize quantization error variance.

   (c) The MSRE of the Predictive Coding system (using $L = 32$ uniform quantization with $\Delta = 6\sigma_e^2/L$) was $\mathcal{E} = \boxed{0.0109}$, significantly higher than the quantization error variance var$(e(n) -$

$\tilde{e}(n)$), calculated as $\boxed{0.0047}$ ($\approx \Delta^2/12 = 0.0037$).

The reason for this increase was discussed in class: the quantization error becomes amplified by the filtering in the decoder.

(d) The DPCM system gave $\mathcal{E} = \boxed{0.0047}$, essentially equal to the quantization error variance and significantly less than that of the simple PCM system. This agrees with the derivations in class, where it was found that the MSRE of DPCM is equal to the quantization error variance.

(e) Though the Prediction Error Transmission system in (a) generated the lowest error, it is impractical for digital storage/transmission since the transmitted signal is a continuous-valued quantity.

The PCM system in (b) has the advantage of simplicity, though its MSRE can be reduced by more complicated systems.

The Predictive Coding system in (c) does not seem useful since it is more complex than PCM while generating similar MSRE.

The DPCM system in (d) generates the lowest MSRE at the expense of increased complexity over PCM. (Note, though, that decent gain is possible even with a short $N = 3$ predictor.)

3. (a) Using the DPCM system with uniform quantization of L reconstruction levels and the suggested values of quantizer stepsize Δ, we obtained the following SNR $= 10\log_{10}(\sigma_x^2/\mathcal{E})$.

L	2	4	8	16	32	64	128
SNR [dB]	6.6119	12.8394	18.5987	24.1665	29.3978	34.7145	39.9293

(b) Assuming zero-mean Gaussian $e(n)$, we can calculate entropy as in Homework #1.

$$H_{\tilde{e}} \;=\; -\sum_k P_k \log_2 P_k \qquad \text{with} \qquad P_k \;=\; \frac{1}{2}\text{erfc}\left(e_k/\sqrt{2\sigma_e^2}\right) - \frac{1}{2}\text{erfc}\left(e_{k+1}/\sqrt{2\sigma_e^2}\right)$$

where $\{e_k\}$ denote the quantizer decision thresholds. Estimating σ_e^2 from our experimental data, we found the following.

L	2	4	8	16	32	64	128
$H_{\tilde{e}}$ [bits]	1.0000	1.9470	2.8049	3.5972	4.4054	5.2715	6.1488

(c) With optimal entropy coding of $\tilde{e}(n)$, the bit rate required for transmission is $H_{\tilde{e}}$ bits/sample. Using $H_{\tilde{e}}$ from (b), this minimum bit rate was plotted versus SNR from (a) in Fig. 4.

For optimal coding of (nonwhite) $x(n)$, the notes say that SNR $= 6.02\,R - 10\log_{10}\text{SFM}_x$, i.e.,

$$R \;=\; \left(\text{SNR} + 10\log_{10}\text{SFM}_x\right)/6.02.$$

This is also plotted in Fig. 4 (using $\text{SFM}_x = 0.34$ from Problem 1).

The following can be observed in Fig. 4:

- For high SNR, the rate for $\tilde{e}(n)$ is approximately 0.3 bits/sample above the optimal, which is close to what the theory predicts (i.e., 0.255 bits/sample).
- For lower SNRs, the rate for $\tilde{e}(n)$ is farther from optimal than the (simple) theory predicts. Most likely, this is due to the feedback of (coarsely) quantized predictions, causing $\sigma_e^2 > \sigma_e^2\big|_{\min,N}$ (which can be verified), hence inefficient coding.

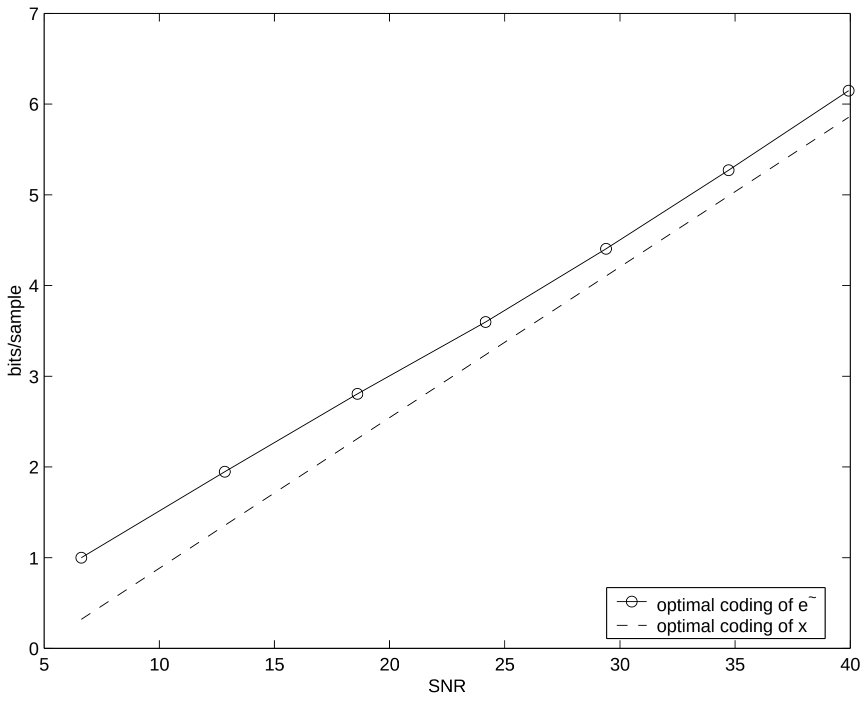

Figure 4: Bit rates for optimal entropy coding of $\tilde{e}(n)$ and optimal coding of $x(n)$.

Matlab code for Problems 1 & 2:

```
% parameters
M = 10000; % length of input sequence
A = 1; % input coloring: denominator
B = [1,0.9,0.8,0.7]; % input coloring: numerator
sig2_v = 1; % driving noise variance
N = 20;  % linear prediction order
L = 32; % uniform quantizer levels

% create input sequence
v = sqrt(sig2_v)*randn(1,M);
x = filter(B,A,v);

% calculate autocorrelation
b = impz(B,A);
rx = xcorr(b,b); rx=rx(:);
lag_x = (length(rx)-1)/2;
sig2_x = rx(lag_x+1);

% calculate power spectrum
N_w = 4096;
w = linspace(-pi,pi,N_w).';
dw = 2*pi/N_w;
Sx = -sig2_x*ones(N_w,1);
for k=0:lag_x, Sx = Sx + 2*rx(k+lag_x+1)*cos(w*k); end;
sig2_e_min = exp( sum(log(Sx))*dw/2/pi )
SFM_x = exp( sum(log(Sx))*dw/2/pi )/( sum(Sx)*dw/2/pi )
%
figure(1);
subplot(211)
  stem([-lag_x:lag_x],rx);
  title('r_x');
subplot(212)
  plot(w,real(Sx));
  axis('tight')
  title('S_x');
drawnow;

% calculate optimal predictor
if lag_x>=N,
  rx_tmp = rx(lag_x+[1:N+1]);
else
  rx_tmp = [rx(lag_x+[1:lag_x+1]); zeros(N-lag_x,1)];
end
R = toeplitz(rx_tmp(1:N));
h = inv(R)*rx_tmp(2:N+1);
sig2_e_minN = rx_tmp(1)-rx_tmp(2:N+1)'*inv(R)*rx_tmp(2:N+1)

% calculate prediction error statistics
q = conv(b,[1;-h]);
re = xcorr(q,q); re=re(:);
lag_e = (length(re)-1)/2;
sig2_e = re(lag_e+1);
```

```
Se = -sig2_e*ones(N_w,1);
for k=0:lag_e, Se = Se + 2*re(k+lag_e+1)*cos(w*k); end;
SFM_e = exp( sum(log(Se))*dw/2/pi )/( sum(Se)*dw/2/pi )

% calculate prediction error and estimate statistics
e = conv(x,[1;-h]);
lag_eh = lag_x+N;
reh = xcorr(e,e,lag_eh,'unbiased'); reh=reh(:);
sig2_eh = reh(lag_eh+1)
Seh = -sig2_eh*ones(N_w,1);
for k=0:lag_eh, Seh = Seh + 2*reh(k+lag_eh+1)*cos(w*k); end;
SFM_eh = exp( sum(log(Seh))*dw/2/pi )/( sum(Seh)*dw/2/pi );
%
figure(2);
subplot(211)
  stem_hndl1 = stem([-lag_eh:lag_eh],reh,'o');
  hold on; stem_hndl2 = stem([-lag_e:lag_e],re,'r^'); hold off;
  title('r_e');
  axis('tight')
  legend([stem_hndl1,stem_hndl2],'estimated','calculated');
subplot(212)
  plot(w,Seh, w,Se,'r--');
  axis('tight')
  title('S_e');
  legend('estimated','calculated');
drawnow;

%%%%%%%%%%%%%%%%%%%%
% ENCODING/DECODING %
%%%%%%%%%%%%%%%%%%%%

% uniform quantizer design for colored input
x_max = 3.0*sqrt(sig2_x);
Delta_x = 2*x_max/L;
Qx = linspace(-x_max,x_max,L+1); Qx(1)=-inf; Qx(L+1)=inf;
Qy = Qx(1:L)+Delta_x/2; Qy(1)=Qx(2)-Delta_x/2;
sig2_qx = Delta_x^2/12;

% PCM
y0 = zeros(1,M);
for k=1:L, y0( find((x>Qx(k))&(x<Qx(k+1))) ) = Qy(k); end; % quantize
z0 = x-y0;
E_pcm = (z0*z0')/length(z0)

% lossless preditive decoding
y1 = zeros(1,N+M); % pad by N
for n=1:M,
  y1(n+N) = e(n) + y1(n+N-[1:N])*h; % decode
end;
y1 = y1(N+[1:M]); % unpad by N
z1 = x-y1;
E_lossless = (z1*z1')/length(z1)

% uniform quantizer design for prediction error
e_max = 3.0*sqrt(sig2_e);
Delta_e = 2*e_max/L;
Qe = linspace(-e_max,e_max,L+1); Qe(1)=-inf; Qe(L+1)=inf;
Qet = Qe(1:L)+Delta_e/2; Qet(1)=Qe(2)-Delta_e/2;

% quantized predictive encoding/decoding
et = zeros(size(e));
for k=1:L, et( find((e>Qe(k))&(e<Qe(k+1))) ) = Qet(k); end; % quantize
y2 = zeros(1,N+M); % pad by N
for n=1:M,
  y2(n+N) = et(n) + y2(n+N-[1:N])*h; % decode
end;
y2 = y2(N+[1:M]); % unpad by N
z2 = x-y2;
sig2_qe = var(et-e);
E_quant = (z2*z2')/length(z2)

% DPCM encoding/decoding
y3 = zeros(1,N+M); % pad by N
xt = zeros(1,N+M);
e3 = zeros(1,N+M);
et3 = zeros(1,N+M);
for n=1:M,
  xth_cur = xt(n+N-[1:N])*h;
  e3(n+N) = x(n)-xth_cur;
  et3(n+N) = Qet(max(find( e3(n+N) > Qe )));
  xt(n+N) = et3(n+N) + xth_cur;
  y3(n+N) = et3(n+N) + y3(n+N-[1:N])*h;
end;
y3 = y3(N+[1:M]); % unpad by N
xt = xt(N+[1:M]);
e3 = e3(N+[1:M]);
et3 = et3(N+[1:M]);
z3 = x-y3;
E_dpcm = (z3*z3')/length(z3)
```

Matlab code for Problem 3:

```
% parameters
M = 10000; % length of input sequence
A = 1; % input coloring: denominator
B = [1,0.9,0.8,0.7]; % input coloring: numerator
sig2_v = 1; % driving noise variance
N = 20;  % linear prediction order
L = 2.^[1:7]; % uniform quantizer levels
gamma = [1.6,2,2.3,2.7,3.1,3.4,3.7]; % uniform quantizer design constants
```

```
 % create input sequence
 v = sqrt(sig2_v)*randn(1,M);
 x = filter(B,A,v);

 % calculate autocorrelation
 b = impz(B,A);
 rx = xcorr(b,b); rx=rx(:);
 lag_x = (length(rx)-1)/2;
 sig2_x = rx(lag_x+1);

 % calculate power spectrum
 N_w = 128;
 w = 2*pi*[0:N_w-1].'/N_w;
 dw = 2*pi/N_w;
%Sx = zeros(N_w,1);
%for k=-lag_x:lag_x, Sx = Sx + rx(k+lag_x+1)*exp(-sqrt(-1)*w*k); end;
%Sx = real(Sx);
 Sx = -sig2_x*ones(N_w,1);
 for k=0:lag_x, Sx = Sx + 2*rx(k+lag_x+1)*cos(w*k); end;
 if min(Sx)<0, error('negative PSD!'); end;
 SFM_x = exp( sum(log(Sx))*dw/2/pi )/( sum(Sx)*dw/2/pi )
 %
 figure(1);
 subplot(211)
   plot([-lag_x:lag_x],rx);
   axis('tight')
   title('r_x');
 subplot(212)
   plot(w,real(Sx));
   axis('tight')
   title('S_x');

 % calculate optimal predictor
 if lag_x>=N,
   rx_tmp = rx(lag_x+[1:N+1]);
 else
   rx_tmp = [rx(lag_x+[1:lag_x+1]); zeros(N-lag_x,1)];
 end
 R = toeplitz(rx_tmp(1:N));
 h = inv(R)*rx_tmp(2:N+1);
 sig2_e_minN = rx_tmp(1)-rx_tmp(2:N+1)'*inv(R)*rx_tmp(2:N+1)


 %%%%%%%%%%%%%%%%%%%%
 % ENCODING/DECODING %
 %%%%%%%%%%%%%%%%%%%%

 SNR_dpcm = zeros(1,length(L));
 H_e = zeros(1,length(L));
 R_dpcm = zeros(1,length(L));
 for l=1:length(L);
   % uniform quantizer design for prediction error
   e_max = gamma(l)*sqrt(sig2_e_minN);
   Delta_e = 2*e_max/L(l);
   Qe = linspace(-e_max,e_max,L(l)+1); Qe(1)=-inf; Qe(L(l)+1)=inf;
   Qet = Qe(1:L(l))+Delta_e/2; Qet(1)=Qe(2)-Delta_e/2;
   sig2_qe = Delta_e^2/12;

   % DPCM encoding/decoding
   y = zeros(1,N+M); % pad by N
   xt = zeros(1,N+M);
   e = zeros(1,N+M);
   et = zeros(1,N+M);
   for n=1:M,
     xth_cur = xt(n+N-[1:N])*h;
     e(n+N) = x(n)-xth_cur;
     et(n+N) = Qet(max(find( e(n+N) > Qe )));
     xt(n+N) = et(n+N) + xth_cur;
     y(n+N) = et(n+N) + y(n+N-[1:N])*h;
   end;
   y = y(N+[1:M]); % unpad by N
   xt = xt(N+[1:M]);
   e = e(N+[1:M]);
   et = et(N+[1:M]);
   z = x-y;
   SNR_dpcm(l) = 10*log10( (x*x')/(z*z') );

   % quantized-prediction-error entropy
   sig2_eh = (e*e')/M;
   H_e(l) = calc_entropy(Qe,sig2_eh);
 end;

 SNR_dpcm
 H_e

 figure(2);
 subplot(111);
 Rmin = ( SNR_dpcm + 10*log10(SFM_x) )/6.02;
 plot(SNR_dpcm,H_e,'ro-', SNR_dpcm,Rmin,'--');
 legend('optimal coding of e^~',...
'optimal coding of x',0);
 xlabel('SNR'); ylabel('bits/sample');
```