HOMEWORK #3 SOLUTIONS

(Note: Matlab code appears at the end.)

1. The constraint in the optimization problem

$$\min_{\{R_k : k\in\mathcal{K}_u\}} \sum_{k\in\mathcal{K}_u} \sigma_{y_k}^2 2^{-2R_k} \quad \text{s.t.} \quad R = \frac{1}{N}\sum_{k=0}^{N-1} R_k,$$

can be rewritten

$$R = \frac{1}{N}\sum_{k\in\mathcal{K}_a} R_k + \frac{1}{N}\sum_{k\in\mathcal{K}_u} R_k$$

since $\mathcal{K}_u \cup \mathcal{K}_a = \{0, 1, \ldots, N-1\}$ and $\mathcal{K}_u \cap \mathcal{K}_a = \{\}$. Since the allocated bit rates $\{R_k : k \in \mathcal{K}_a\}$ are known, the constraint can be rewritten

$$\bar{R} := \frac{NR - \sum_{k\in\mathcal{K}_a} R_k}{\text{size}(\mathcal{K}_u)} = \frac{1}{\text{size}(\mathcal{K}_u)}\sum_{k\in\mathcal{K}_u} R_k$$

for known $\bar{R}$. The optimization problem is now in the form

$$\min_{\{R_k : k\in\mathcal{K}_u\}} \sum_{k\in\mathcal{K}_u} \sigma_{y_k}^2 2^{-2R_k} \quad \text{s.t.} \quad \bar{R} = \frac{1}{\text{size}(\mathcal{K}_u)}\sum_{k\in\mathcal{K}_u} R_k. \tag{1}$$

In the notes, we proved that a different optimization problem:

$$\min_{\{R_k\}} \sum_{k=0}^{N-1} \sigma_{y_k}^2 2^{-2R_k} \quad \text{s.t.} \quad R = \frac{1}{N}\sum_{k=0}^{N-1} R_k, \tag{2}$$

had the solution

$$R_\ell^{\text{opt}} = R + \frac{1}{2}\log_2\left(\frac{\sigma_{y_\ell}^2}{\left(\prod_{k=0}^{N-1}\sigma_{y_k}^2\right)^{1/N}}\right) \qquad \text{for} \quad \ell = 0, 1, \ldots, N-1. \tag{3}$$

But (2) is identical to (1) after setting

$$\begin{aligned} R &\to \bar{R} \\ \{0, 1, \ldots, N-1\} &\to \mathcal{K}_u. \end{aligned}$$

Thus, the solution to (1) is found by applying the notational changes above to (3):

$$R_\ell^{\text{qua}} = \frac{NR - \sum_{k\in\mathcal{K}_a} R_k}{\text{size}(\mathcal{K}_u)} + \frac{1}{2}\log_2\left(\frac{\sigma_{y_\ell}^2}{\left(\prod_{k\in\mathcal{K}_u}\sigma_{y_k}^2\right)^{1/\,\text{size}(\mathcal{K}_u)}}\right) \qquad \text{for} \quad \ell \in \mathcal{K}_u.$$

2. (a) For $x(n) = \sum_{i=0}^{\infty} h_i v(n-i)$ and unit-variance white $v(n)$, we know that

$$r_x(k) \;=\; \sum_{i=0}^{\infty} h_i h_{k+i} \;\approx\; \sum_{i=0}^{N_h} h_i h_{k+i}$$

where N_h is a suitably large number. Then

$$S_x(e^{j\omega}) \;=\; \sum_{k=-\infty}^{\infty} r_x(k) e^{-j\omega k} \;\approx\; \sum_{k=-N_h}^{N_h} r_x(k) e^{-j\omega k}$$

Fig. 1 plots a truncated version of $r_x(k)$ and the resulting approximation to $S_x(e^{j\omega})$ for $\{h_i\}$ corresponding to the system $H(z) = 1/(1-0.8z^{-1})$ and N_q chosen by Matlab's `impz` command.

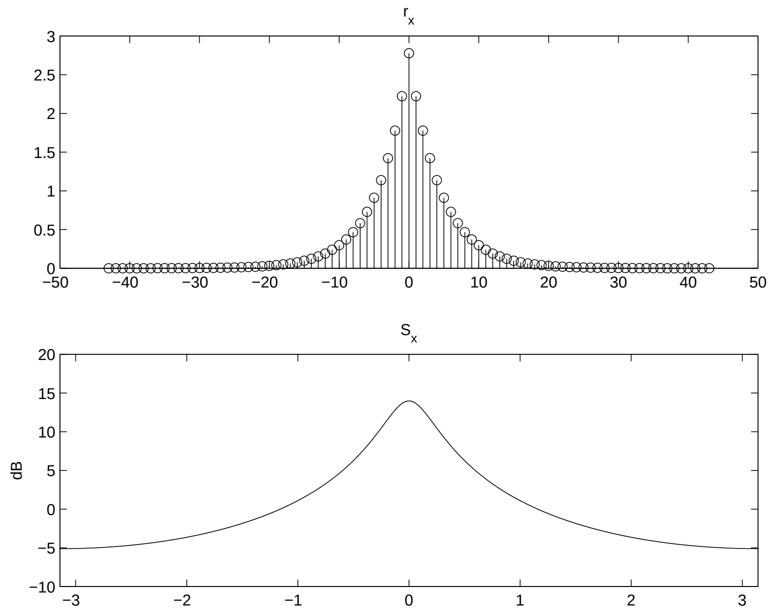

Figure 1: Truncated autocorrelation and power spectrum for source of Problem 2.

(b) From the notes, optimal transformation and bit allocation yield

$$\sigma_r^2\big|_{\mathrm{TC},N\to\infty} \;=\; \gamma_y \sigma_x^2\, 2^{-2R}\, \mathrm{SFM}_x.$$

Using

$$\gamma_y \;=\; \frac{1}{3}\frac{y_{\max}^2}{\sigma_y^2} \;\approx\; \frac{1}{3}\frac{(\phi_y\sigma_y)^2}{\sigma_y^2} \;=\; \frac{1}{3}\phi_y^2 \;=\; \frac{1}{3}3^2 \;=\; 3,$$

using $R = 4$, and using the calculated values $\sigma_x^2 = 2.7778$ and $\mathrm{SFM}_x = 0.3600$, we find that $\boxed{\sigma_r^2\big|_{\mathrm{TC},N\to\infty} = 0.0117}$.

(c) From the notes, optimal transformation and bit allocation yield

$$\sigma_r^2\big|_{\mathrm{TC},N} \;=\; \gamma_y 2^{-2R} \left(\prod_{k=0}^{N-1} \lambda_k\right)^{1/N}$$

where $\{\lambda_k\}$ are the eigenvalues of the $N\times N$ input autocorrelation matrix $\mathbf{R}_x$. Matlab computation gives $\boxed{\sigma_r^2\big|_{\mathrm{TC},N} = 0.0125}$.

(d) If we define the transform-output vector $\mathbf{y}(m) = \mathbf{T}\mathbf{x}(m)$ then

$$\begin{aligned}(\sigma_{y_0}^2, \ldots, \sigma_{y_{N-1}}^2)^t &= \operatorname{diag}\big(\mathrm{E}\{\mathbf{y}(m)\mathbf{y}^t(m)\}\big) \\ &= \operatorname{diag}\big(\mathrm{E}\{\mathbf{T}\mathbf{x}(m)\mathbf{x}^t(m)\mathbf{T}^t\}\big) \\ &= \operatorname{diag}\big(\mathbf{T}\,\mathrm{E}\{\mathbf{x}(m)\mathbf{x}^t(m)\}\mathbf{T}^t\big) \\ &= \operatorname{diag}\big(\mathbf{T}\mathbf{R}_x\mathbf{T}^t\big).\end{aligned}$$

From the notes, optimal bit allocation yields

$$\sigma_r^2\big|_{\mathrm{TC},N} = \gamma_y 2^{-2R} \left(\prod_{k=0}^{N-1} \sigma_{y_k}^2\right)^{1/N}.$$

Using $\{\sigma_{y_k}^2\}$ calculated in Matlab for DCT matrix $\mathbf{T}$, $\boxed{\sigma_r^2\big|_{\mathrm{TC},N} = 0.0126}$.

(e) Implementing the adaptive transform coder, we obtain Fig. 2 and $\boxed{\mathcal{E}_{\mathrm{TC}} = 0.0184}$.

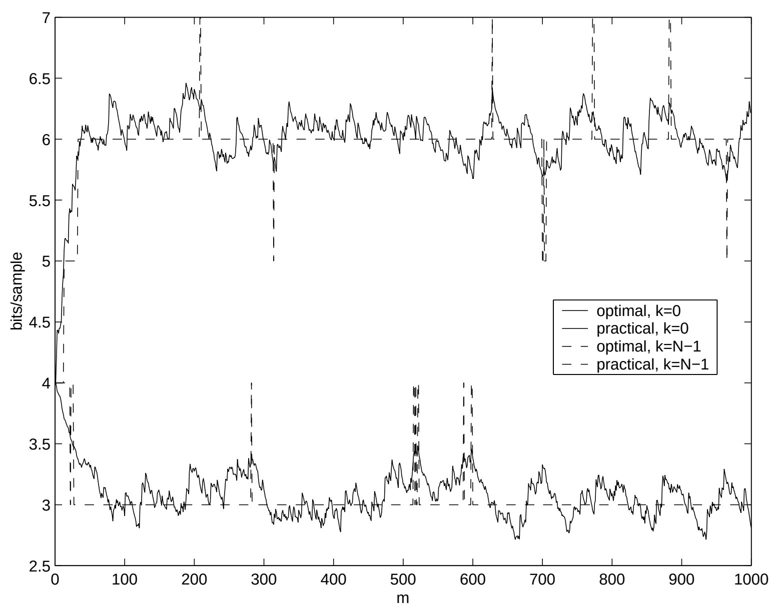

Figure 2: Optimal and practical bit allocations for output branches $k = 0$ and $k = N-1$ versus input block m.

(f) Implementing the PCM coder, we obtain $\boxed{\mathcal{E}_{\mathrm{PCM}} = 0.0341}$.

(g) The results of parts (b)-(f) are summarized below:

transform	bit allocation	σ_r^2
KLT, $N \to \infty$	optimal	0.0117
KLT, $N = 16$	optimal	0.0125
DCT, $N = 16$	optimal	0.0126
DCT, $N = 16$	practical	≈ 0.0184
PCM	n.a.	≈ 0.0341

We conclude the following

- Transform dimension $N = 16$ is large enough to give performance close to the asymptotic $N \to \infty$ case.

- For the lowpass input process $x(n)$ (see Fig. 1), the DCT performs nearly as well as the KLT.
- Practical bit allocation increases reconstruction error by about 50% over optimal bit allocation.
- For the lowpass input process $x(n)$, DCT coding with practical bit allocation yields reconstruction error that is about half of that for PCM.

3. (a) From 2(d) and the notes, we know that

$$G_{\mathrm{TC}} \;=\; \frac{\gamma_x}{\gamma_y} \frac{\sigma_x^2}{\left(\prod_{k=0}^{N-1} \sigma_{y_k}^2\right)^{1/N}} \quad \text{where} \quad (\sigma_{y_0}^2, \ldots, \sigma_{y_{N-1}}^2)^t \;=\; \mathrm{diag}\big(\mathbf{T}\mathbf{R}_x\mathbf{T}^t\big).$$

More compactly,

$$G_{\mathrm{TC}} \;=\; \frac{\sigma_x^2}{\left(\prod_{k=0}^{N-1} \left[\mathbf{T}\mathbf{R}_x\mathbf{T}^t\right]_{k,k}\right)^{1/N}}$$

where we have used the fact that Gaussianity is preserved under linear transformation, so that $\gamma_x = \gamma_y$. Fig. 3 plots G_{TC} versus N for various transforms when $x(n)$ is generated by filtering white noise with the filter

$$H(z) \;=\; \frac{1}{A(z)} \;=\; \frac{1}{1 - 0.8z^{-1} + 0.4z^{-2}}.$$

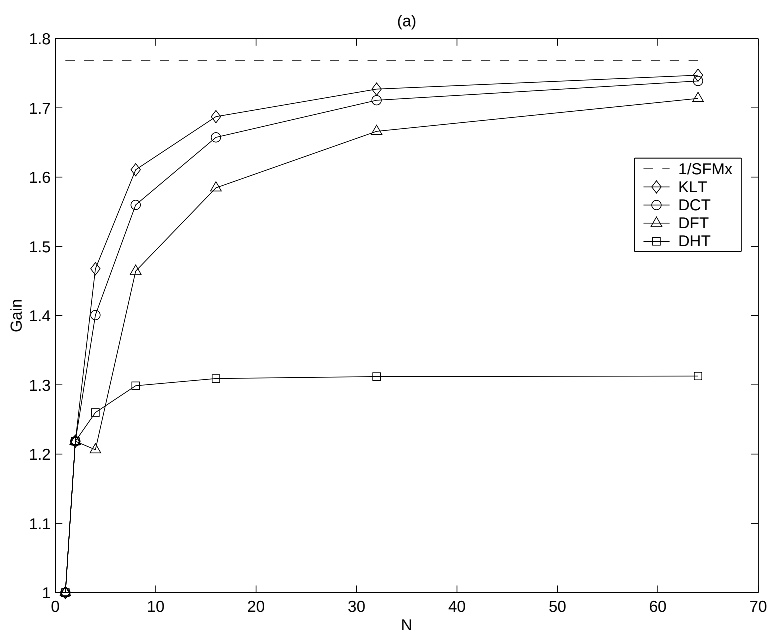

Figure 3: G_{TC} versus transform dimension for various transforms and source from 3(a).

(b) Fig. 4 plots G_{TC} versus N for various transforms when $x(n)$ is generated by filtering white noise with the filter

$$H(z) \;=\; \frac{1}{A(z)} \;=\; \frac{1}{1 + 0.7z^{-1} + 0.2z^{-2}}.$$

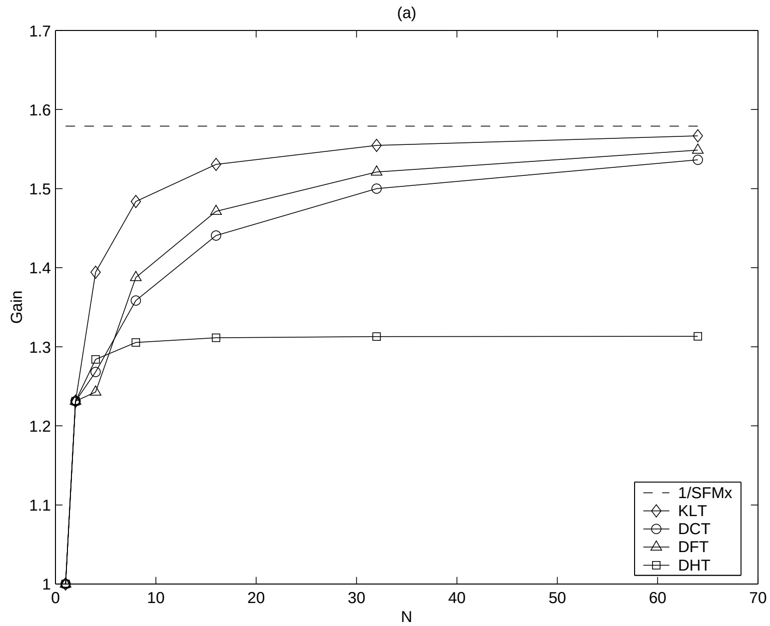

Figure 4: TC gain over PCM versus transform dimension for various transforms on source from 3(b).

(c) The following interpretations are drawn from a comparison of Fig. 3 and Fig. 4.

- The KLT performs at least as well as the other transforms for all N, as expected.
- The DCT does better than the real-DFT in 3(a) and worse in 3(b). This is expected because the input process in 3(a) is lowpass while the input process in 3(b) is highpass.
- The input spectrum in 3(b) is flatter than that of 3(a) hence less TC-gain-over-PCM is available. This might be guessed from looking at the spectra in Fig. 5 and Fig. 6.
- The KLT, real-DFT, and DCT, approach asymptotic optimal performance as $N \to \infty$, while the DHT does not.

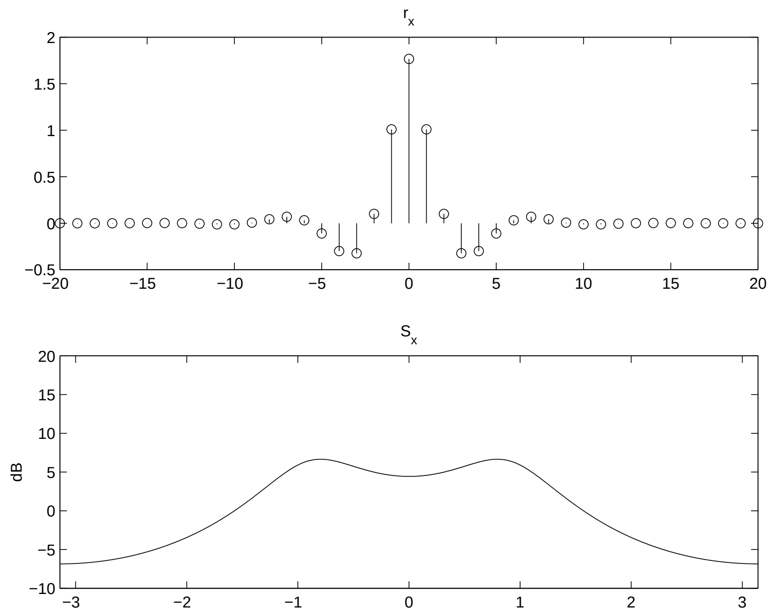

Figure 5: Truncated autocorrelation and power spectrum for source of 3(a).

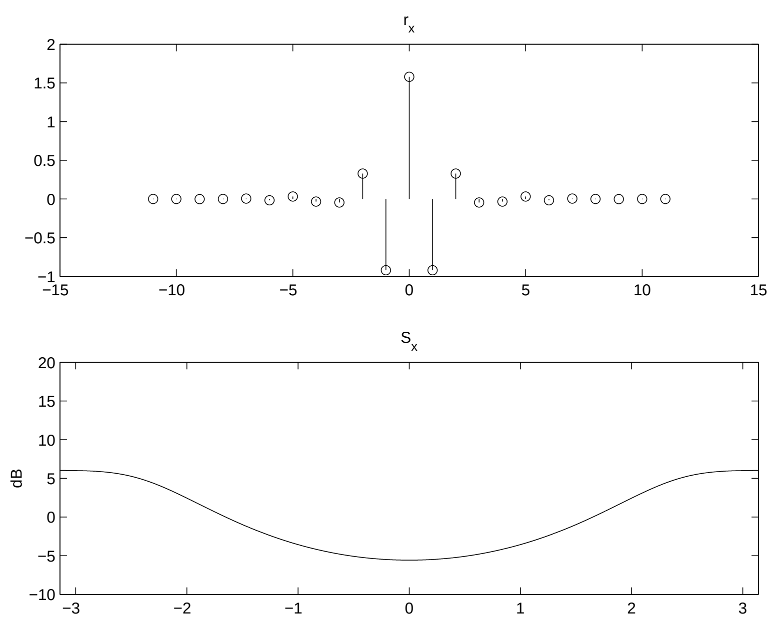

Figure 6: Truncated autocorrelation and power spectrum for source of 3(b).

Matlab code for Problem 2:

```
% parameters
A=[1,-0.8];
B=1;
sig2_v = 1; % driving noise variance
N = 16;  % transform dimension
R = 4; % average bits/sample
alf = 0.95; % variance calculation forget factor
gam = 3; % uniform quantizer factor

% 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 = 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);
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,10*log10(real(Sx)));
  axis([-pi,pi,-10,20])
  title('S_x');
  ylabel('dB')
drawnow;

% reconstruction error with optimal bit allocation and N=infty KLT
E_tc_asymt = sig2_x*gam*2^(-2*R)*SFM_x

% create NxN autocorrelation matrix
Rx = toeplitz([ rx(lag_x+[1:min(lag_x+1,N)]); zeros(N-lag_x-1,1) ]);
[Vx,Lx] = eig(Rx);

% reconstruction error with optimal bit allocation and KLT
E_tc_optRoptT = gam*2^(-2*R)*prod(diag(Lx))^(1/N)

% transform matrices
T = dctmtx(N); % DCT
%T = dftrmtx(N); % real DFT
%T = hadamard(N)/sqrt(N); % DHT
%T = Vx'; % KLT
sig2_y = diag( T*Rx*T');

% reconstruction error with optimal bit allocation
E_tc_optR = gam*2^(-2*R)*prod(sig2_y)^(1/N)

% create and transform input signal
M = 1000;
v = randn(1,M*N)*sqrt(sig2_v);
x = filter(B,A,v);
xx = zeros(N,M);
xx(:) = x; % each column is an input N-block
yy = T*xx;  % transform input signal

% adaptive coding
Ro = zeros(N,M);
Rq = zeros(N,M);
yq = zeros(N,M);
for i=1:M,
  % recursive variance estimation
  if i==1,
    sig2_y_hat = sig2_x*ones(N,1);
    %sig2_y_hat = sig2_y;
  else
    sig2_y_hat = (1-alf)*yq(:,i-1).^2 + alf*sig2_y_hat_old;
  end;
  sig_y_hat = sqrt(sig2_y_hat);
  sig2_y_hat_old = sig2_y_hat;

  % bit allocation
  R_q = zeros(N,1);
  K_u = [1:N]; % list of unalloc branches
  for length_ku = N:-1:1,
    R_opt = (N*R-sum(R_q))/length_ku*ones(length_ku,1) +...
        0.5*log2(sig2_y_hat(K_u)/(prod(sig2_y_hat(K_u))^(1/length_ku)));
    if length_ku==N, Ro(:,i) = R_opt; end;
    [R_srt,indx] = sort(R_opt); % sort unallocated {Rk}
    K_u = K_u(indx); % reorder list
    R_q(K_u(1)) = max(0,round(R_srt(1))); % quantize smallest rate
    K_u = K_u(2:length_ku); % update unalloc list
  end;
  Rq(:,i) = R_q;

  % quantization
  L = 2.^(R_q);
  for k=1:N,
    % quantizer design
    y_thresh = linspace(-gam*sig_y_hat(k),gam*sig_y_hat(k),L(k)+1);
    y_quant = y_thresh(2:L(k)+1)-gam*sig_y_hat(k)/L(k);
    y_thresh(1) = -inf; y_thresh(L(k)+1) = inf;

    %quantizer implementation
    yq(k,i) = y_quant(max(find( yy(k,i) > y_thresh )));
  end;
end;
```

```
% decoding
zz = (T.')*yq;
z = zz(:).';
E_tc = (z-x)*(z-x)'/(N*M)

% compare to PCM error...
L_x = 2^R;
x_thresh = linspace(-gam*sqrt(sig2_x),gam*sqrt(sig2_x),L_x+1);
x_quant = x_thresh(2:L_x+1)-gam*sqrt(sig2_x)/L_x;
x_thresh(1) = -inf; x_thresh(L_x+1) = inf;
z_tc = zeros(1,N*M);
for l=1:L_x,
  z_tc( find((x>x_thresh(l))&(x<x_thresh(l+1))) ) = x_quant(l);
end;
E_pcm = (z_tc-x)*(z_tc-x)'/(N*M)

figure(2)
plot([1:M],Ro([1,N],:), [1:M],Rq([1,N],:),'--');
ylabel('bits/sample'); xlabel('m')
legend('optimal, k=0','practical, k=0','optimal, k=N-1','practical, k=N-1',0);
```

Matlab code for Problem 3:

```
% parameters
A=[1,0.7,0.2];
B=1;
sig2_v = 1; % driving noise variance
NN = 2.^[0:6];  % transform dimension

% 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 = 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);
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,10*log10(real(Sx)));
  axis([-pi,pi,-10,20])
  title('S_x');
  ylabel('dB')
drawnow;

% compare KLT, DFT, DCT, and DHT
G_dft = zeros(1,length(NN));
G_dct = zeros(1,length(NN));
G_dht = zeros(1,length(NN));
for l=1:length(NN),
  % create NxN autocorrelation matrix
  N = NN(l);
  Rx = toeplitz([ rx(lag_x+[1:min(lag_x+1,N)]); zeros(N-lag_x-1,1) ]);
  Lx = eig(Rx);

  % transform matrices
  T_dft = dftrmtx(N);
  T_dct = dctmtx(N);
  T_dht = hadamard(N)/sqrt(N);

  % coefficient variances
  sig2_y_klt = Lx;
  Ry_dft = T_dft*Rx*T_dft';  sig2_y_dft = diag(Ry_dft);
  Ry_dct = T_dct*Rx*T_dct';  sig2_y_dct = diag(Ry_dct);
  Ry_dht = T_dht*Rx*T_dht';  sig2_y_dht = diag(Ry_dht);

  % gains over pcm (assuming Gaussian source)
  G_klt(l) = sig2_x/prod(sig2_y_klt)^(1/N);
  G_dft(l) = sig2_x/prod(sig2_y_dft)^(1/N);
  G_dct(l) = sig2_x/prod(sig2_y_dct)^(1/N);
  G_dht(l) = sig2_x/prod(sig2_y_dht)^(1/N);
end;

figure(2);
plot(NN,ones(size(NN))/SFM_x,'--',...
     NN,G_klt,'-d',...
     NN,G_dct,'-o',...
     NN,G_dft,'-^',...
     NN,G_dht,'-s');
legend('1/SFMx','KLT','DCT','DFT','DHT',0);
ylabel('Gain'); xlabel('N');
title('(a)');
```

dftrmtx.m:

```
% makes orthogonal real-valued DFT matrix

function T_dftr = dftrmtx(N)

 if N==1,
   T_dftr = 1;
 else,
   T_dft = dftmtx(N)/sqrt(N);
   C2R = zeros(N); C2R(1,1)=1; C2R(N,1+N/2)=1;
   for n=1:N/2-1,
     C2R(1+[2*n-1:2*n],1+n) = [-sqrt(-1);1]/sqrt(2);
     C2R(1+[2*n-1:2*n],1+N-n) = [sqrt(-1);1]/sqrt(2);
   end;
   T_dftr = real( C2R*T_dft );
 end;
```