HOMEWORK #5 SOLUTIONS

1. Matlab code appears at end.

2. For the mono audio file `sco_29.wav` encoded using $R = 96$ kbps, the noise level ($\mathrm{L_{sb}} - \mathrm{SNR_q}(B_{\mathrm{data}})$), the minimum masking threshold ($\mathrm{L_{mask}}$), and the MNR at the second frame are shown in Fig. 1. The bit allocations for subbands 1, 8, and 28 are shown in Fig. 2. Finally, the input waveform, reconstructed output waveform, and their difference is shown in Fig. 3.

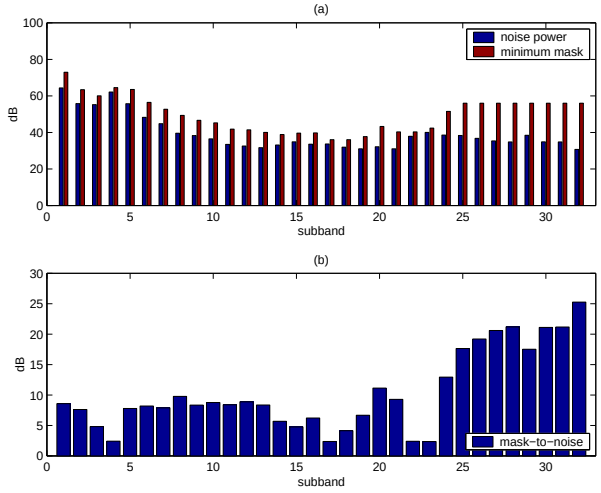

Figure 1: Example of (a) noise level ($\mathrm{L_{sb}} - \mathrm{SNR_q}(B_{\mathrm{data}})$) versus minimum masking threshold ($\mathrm{L_{mask}}$), and (b) MNR.

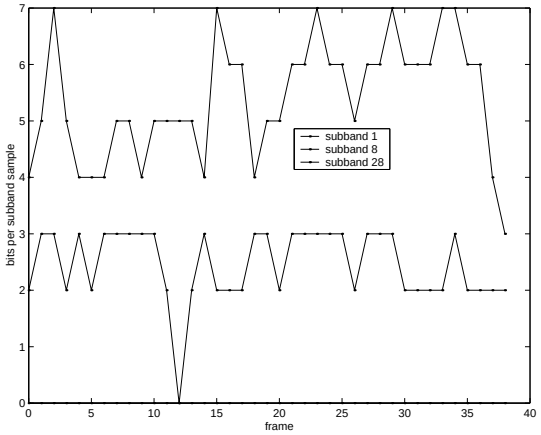

Figure 2: Example bit allocations for three subbands.

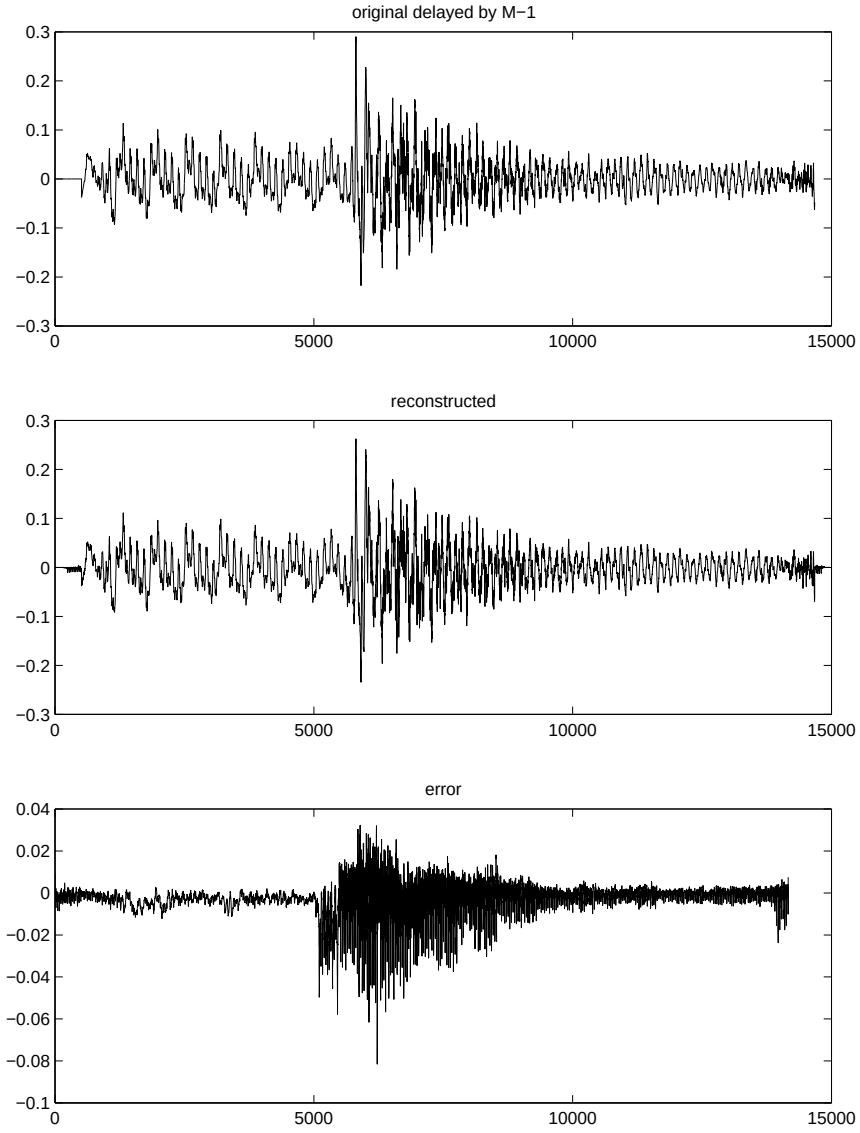

Figure 3: Example of input delayed by $M_h - 1$, reconstructed output, and their difference.

3. Testing the coder/decoder on `sco_160.wav` at bitrates $R = 192$, 160, 128, and 96 kbps, I thought that 192 sounded identical to the original, 160 sounded very very close, 128 had some noticable but not terribly annoying distortion, and 96 had annoying distortion.

Layer 1 coder/decoder:

```
% simplified MPEG Layer 1 encoder/decoder

 % general setup
 inputname = 'sco_160.wav'; % input filename (sco_160.wav)
 bitrate = 112; % in kbps (192,160,128,96)
 store_on = 1;    % set nonzero to store an output file
 plot_on = 1; % set to display NMR for each frame
 smr_plot = 0; % set to display masking plots

 % filterbank setup
 blk_per_frame = 12; % subband blocks per frame
 N = 32; % number of subbands
 N_fr = N*blk_per_frame; % samples in a frame
 load h_mpeg; M = 513; h = h(1:M-1); % subband analysis filter

 % input signal
 L_max = inf; % max number of elements to process
 %L_max = 3000;
 [x,Fs,Nbits,Opts] = wavread(inputname);
 L = min(length(x),L_max);
 x = x(1:L);
 x = x(:)';

 % compute pqf outputs
 fprintf('\n  subband analysis... ');
 S = pqf_anal(x,h,N);  % subband analysis
 fprintf('complete.\n');
 S = [S, zeros(N,blk_per_frame*ceil(size(S,2)/blk_per_frame)-size(S,2))];
 frames = size(S,2)/blk_per_frame; % total number of frames

 % fft setup
 M_fft = 512;
 window = hanning(M_fft).';
 xa = [zeros(1,(M_fft+N+M-1-N_fr)/2),x];% zero pad for proper fft delay
 xa = [xa,zeros(1,M_fft+N_fr*(frames-1)-length(xa))]; % zero pad for length

 % scale factor setup
 scale_tab = mpeg_scale; % table of scale factors
 scale_indx = zeros(N,frames); % indices to transmit

 % bit-alloc/quant setup
 bits_header = 24;
 bits_bitalloc = 4*N;
 bits_avail = floor( bitrate*1000/Fs*N_fr );

 % main encoding loop
 Sn = zeros(size(S));
 Sq = zeros(size(S));
 bits_samp = zeros(N,frames);
 fprintf('  subband encoding...');
 for f=0:frames-1,

   % scaling of subband outputs
   for i=0:N-1,
     blks = [f*blk_per_frame+1:(f+1)*blk_per_frame];
     max_val = max(abs( S(i+1,blks) ));
     scale_indx(i+1,f+1) = max(find(max_val<scale_tab));
     Sn(i+1,blks) = S(i+1,blks)/scale_tab(scale_indx(i+1,f+1));
   end;

   % fft analysis
   X = fft(window.*xa(f*N_fr+[1:M_fft]));

   % calc signal to mask ratio
   %if f==20, smr_plot=1; else smr_plot=0; end;
   [SPL,LTmin]=calc_mask(X,scale_tab(scale_indx(:,f+1)),N,bitrate,Fs,smr_plot);
   SMR = SPL - LTmin;

   % bit allocation
   SNR_table = mpeg_SNR;
   bits_scale = zeros(1,N);
   exit_loop = 0;
   while ~exit_loop,
     SNR_q = SNR_table(bits_samp(:,f+1)+1);
     MNR = SNR_q - SMR; % mask-to-noise ratio
     [MNR_sort,indx_sort] = sort(MNR);
     while bits_samp(indx_sort(1),f+1)==15,% ignore if >= 15 bits
       indx_sort = indx_sort(2:length(indx_sort));
     end;
     alloc_indx = indx_sort(1); % this index needs it the most
     bits_remaining = bits_avail - blk_per_frame*sum(bits_samp(:,f+1))...
      - sum(bits_scale) - bits_header - bits_bitalloc;
     if bits_samp(alloc_indx,f+1) > 0,
       if bits_remaining >= blk_per_frame,
         bits_samp(alloc_indx,f+1) = bits_samp(alloc_indx,f+1)+1;
       else
         exit_loop = 1; % not enough bits...
       end;
     else
       if bits_remaining >= 2*blk_per_frame+6;,
         bits_samp(alloc_indx,f+1) = 2;
         bits_scale(alloc_indx) = 6;
       else
         exit_loop = 1; % not enough bits
       end;
     end;
   end;
   if plot_on,
     SNR_q = SNR_table(bits_samp(:,f+1)+1); % final quantizer SNR
     MNR = SNR_q - SMR; % final mask-to-noise ratio
     subplot(211);
       bar([SPL-SNR_q;LTmin]','grouped')
       axe = axis; axis([0,33,0,100]);
       title('(a)'); xlabel('subband'); ylabel('dB');
       legend('noise power','minimum mask',1);
     subplot(212);
       bar(MNR)
       axe = axis; axis([0,33,axe(3:4)]);
       title('(b)'); xlabel('subband'); ylabel('dB');
       legend('mask-to-noise',4);
     drawnow;
   end;
   fprintf('%3d%%\b\b\b\b',round(100*(f+1)/frames));
   %if f==1, return; end;

   % quantization
   for i=0:N-1,
     blks = [f*blk_per_frame+1:(f+1)*blk_per_frame];
     Sq(i+1,blks) = floor( (Sn(i+1,blks)*(2^bits_samp(i+1,f+1)-1)-1)/2 );
   end;
 end;
 fprintf(' complete.\n');

%%%%%%%%%%%%

 % main decoding loop
 Su = zeros(size(S));
 Sr = zeros(size(S));
 fprintf('  subband decoding...');
 for f=0:frames-1,
   % un-quantization and rescaling of subband outputs
   for i=0:N-1,
     blks = [f*blk_per_frame+1:(f+1)*blk_per_frame];
     if bits_samp(i+1,f+1)>0,
       Su(i+1,blks) = 2*(Sq(i+1,blks)+1)/(2^bits_samp(i+1,f+1)-1);
       Sr(i+1,blks) = Su(i+1,blks)*scale_tab(scale_indx(i+1,f+1));
     end;
   end;
   fprintf('%3d%%\b\b\b\b',round(100*(f+1)/frames));
 end;
 fprintf(' complete.\n');

 % subband reconstruction
 fprintf('  subband synthesis... ');
 xr = pqf_synth(Sr,h);
 fprintf(' complete.\n\n');

%%%%%%%%%%%%

 % calc MSRE
 e = xr((M-1)+[1:length(x)])-x;
 MSRE = norm(e)^2/length(e);
 MSRE_dB = 10*log10(MSRE)

 % plot reconstructed output
 figure(2);
 subplot(311); plot([zeros(1,M-1),x]); title('original delayed by M-1');
 subplot(312); plot(xr); title('reconstructed');
 subplot(313); plot(e); title('error');
 orient tall;

 % plot bit allocations
 figure(1);
 subplot(111);
 lookbands = [1,8,28];
 plot([0:frames-1],bits_samp(lookbands,:).','.-')
 ylabel('bits per subband sample'); xlabel('frame');
 leg_str = [];
 for i=1:length(lookbands),
   leg_str = strvcat(leg_str,['subband ',num2str(lookbands(i))]);
 end;
 legend(leg_str,0)

 % store output
 outputname=[inputname(1:length(inputname)-4),'_mpeg',num2str(bitrate),'.wav'];
 if store_on,
   wavwrite(xr((M-1)+[1:length(x)]),Fs,16,['../sounds/',outputname]);
 end;
```

MPEG Scalefactor Table:

```
function scale = mpeg_scale
scale = [
   2.00000000000000; 1.58740105196820; 1.25992104989487; 1.00000000000000;
   0.79370052598410; 0.62996052494744; 0.50000000000000; 0.39685026299205;
   0.31498026247372; 0.25000000000000; 0.19842513149602; 0.15749013123686;
   0.12500000000000; 0.09921256574801; 0.07874506561843; 0.06250000000000;
   0.04960628287401; 0.03937253280921; 0.03125000000000; 0.02480314143700;
   0.01968826640461; 0.01562500000000; 0.01240157071850; 0.00984313320230;
   0.00781250000000; 0.00620078535925; 0.00492156660115; 0.00390625000000;
   0.00310039267963; 0.00246078330058; 0.00195312500000; 0.00155019633981;
   0.00123039165029; 0.00097656250000; 0.00077509816991; 0.00061519582514;
   0.00048828125000; 0.00038754908495; 0.00030759791257; 0.00024414062500;
   0.00019377454248; 0.00015379895629; 0.00012207031250; 0.00009688727124;
   0.00007689947814; 0.00006103515625; 0.00004844363562; 0.00003844973907;
   0.00003051757813; 0.00002422181781; 0.00001922486954; 0.00001525878906;
   0.00001211090890; 0.00000961243477; 0.00000762939453; 0.00000605545445;
   0.00000480621738; 0.00000381469727; 0.00000302772723; 0.00000240310869;
   0.00000190734863; 0.00000151386361; 0.00000120155435  ];
```

MPEG SNRq Table:

```
function SNR = mpeg_SNR;
  SNR = [0 NaN 7 16 25.28 31.59 37.75 43.84 49.89 55.93 61.96 67.98...
    74.01 80.03 86.05 92.01];
```

Psychoacoustic Model:

```
% Psychoacoustic Model #1 for MPEG-1 Layer 1
%
% usage:
%   [Lsig,Lmask] = calc_mask(X,scale_facs,N,bitrate,Fs,plot_on)
% where
%   Lsig : sound pressure levels, per subband, in dB
%   Lmask : minimum masking thresholds, per subband, in dB
%   X  : FFT magnitude of Hann-windowed input frame, usually 512 pts
%   scale_facs  : MPEG scale factors, per subband
%   N  : number of subbands, usually 32
%   bitrate  : bitrate (per channel) in kbps
%   Fs  : sampling frequency in Hz
%   plot_on  : set =1 to display optional plots, =0 otherwise
%
% Copyright 1999 --- Phil Schniter

function [Lsb,LTmin] = calc_mask(X,scale_facs,N_sb,bitrate,Fs,plot_on)

  % constants
  M_fft = length(X); % fft length
  X_fft = 20*log10( max( abs(X)/M_fft,1e-10) ); % normalized FFT
  X_fft = X_fft(1:M_fft/2+1)+(96-max(X_fft));   % normalize to ref SPL of 96dB

  % SPL calculation
  for i=1:N_sb,% note peak-to-avg adjustment of 10dB
    Lsb(i) = max([ 20*log10(scale_facs(i)*32768)-10,...
         X_fft([(i-1)*M_fft/2/N_sb:i*M_fft/2/N_sb-1]+1) ]);
  end;

  % load absolute threshold table and bin/table_index conversions
  [tabindx2bin, bin2tabindx, LTq] = mpeg_abs_thresh(Fs,M_fft,bitrate);
  N_tab = length(LTq);

  % identify tonal components and their SPL contribution using X_fft
  bin_locmax = [];
  for k=2:M_fft/2,
    if (X_fft(k) > X_fft(k-1)) & (X_fft(k) >= X_fft(k+1)),
      bin_locmax = [bin_locmax, k]; % bin numbers of local maxima
    end;
  end;
  bin_tonal = [];
  Xtm = [];
  X_nontonal = X_fft;
  for i=1:length(bin_locmax),
    if (bin_locmax(i) > 3) & (bin_locmax(i) < M_fft/8),
      rng = [-2,2];
    elseif (bin_locmax(i) >= M_fft/8) & (bin_locmax(i) < M_fft/4),
      rng = [-3,-2,2,3];
    elseif (bin_locmax(i) >= M_fft/4) & (bin_locmax(i) <= M_fft/2-5),
      rng = [-6:-2,2:6];
    else
      rng = NaN; % dont check for tonality
    end;
    if isfinite(rng),
      if X_fft(bin_locmax(i)) >= max(X_fft(bin_locmax(i)+rng)) + 7,
        bin_tonal = [bin_tonal,bin_locmax(i)];
        Xtm = [Xtm, 10*log10(sum(10.^( X_fft(bin_locmax(i)+[-1,0,1])/10 )))];
        X_nontonal(bin_locmax(i)+[rng,-1,0,1]) = -200;
      end;
    end;
  end;

  % calculate contribution of non-tonal components in each critical band
  critbnd_tabindx = mpeg_critbnd_bndrs; % critical band top edges
  N_crit = length(critbnd_tabindx);
  Xnm = zeros(1,N_crit);
  bin_nontonal = zeros(1,N_crit);
  for i=1:N_crit-1,
    rng=[tabindx2bin(critbnd_tabindx(i)):tabindx2bin(critbnd_tabindx(i+1))-1];
    Xnm(i) = 10*log10(sum(10.^( X_nontonal(rng)/10 )));
    bin_nontonal(i) = round(exp(mean(log(rng))));
  end;
  rng=[tabindx2bin(critbnd_tabindx(N_crit)):M_fft/2+1];
  Xnm(N_crit) = 10*log10(sum(10.^( X_nontonal(rng)/10 )));
  bin_nontonal(N_crit) = round(exp(mean(log(rng))));

  % plot undecimated tonal & nontonal in bark domain
  if plot_on,
    subplot(211);
    plot(bin2bark([1:M_fft/2+1],M_fft,Fs),X_fft);
    hold on;
    if ~isempty(bin_tonal),
      plot(bin2bark(bin_tonal,M_fft,Fs),Xtm,'ro');
    else,
      plot(bin2bark(bin_nontonal,M_fft,Fs),Xnm,'ro');  % will be covered over
    end;
    plot(bin2bark(bin_nontonal,M_fft,Fs),Xnm,'go');
    plot(bin2bark([1:M_fft/2],M_fft,Fs),LTq(bin2tabindx([1:M_fft/2])),'k:');
    hold off;
    axis([0,25,-20,100]);
    xlabel('bark'); title('tonal & nontonal components, pre-decimation')
    legend('fft','tonal component','nontonal component','abs threshold',0);
  end;

  % remove tonal components lying below absolute threshold
  keep_t = [];
  for i=1:length(bin_tonal),
    if Xtm(i) >= LTq(bin2tabindx(bin_tonal(i))), keep_t = [keep_t,i]; end;
  end;
  Xtm = Xtm(keep_t);
  bin_tonal = bin_tonal(keep_t);

  % remove nontonal components lying below absolute threshold
  keep_n = [];
  for i=1:length(bin_nontonal),
    if Xnm(i) >= LTq(bin2tabindx(bin_nontonal(i))), keep_n = [keep_n,i]; end;
  end;
  Xnm = Xnm(keep_n);
  bin_nontonal = bin_nontonal(keep_n);

  % remove tonal components within 0.5 Bark of stronger tonal component
  i=1;
  while i<length(bin_tonal)-1,
    if abs( bin2bark(bin_tonal(i+1),M_fft,Fs)...
          -bin2bark(bin_tonal(i),M_fft,Fs) ) < 0.5,
      if Xtm(i) > Xtm(i+1),
        Xtm = Xtm([1:i,i+2:length(bin_tonal)]);
        bin_tonal = bin_tonal([1:i,i+2:length(bin_tonal)]);
      else
        Xtm = Xtm([1:i-1,i+1:length(bin_tonal)]);
   bin_tonal = bin_tonal([1:i-1,i+1:length(bin_tonal)]);
      end;
    else
      i = i+1;
    end;
  end;

  % plot decimated tonal & nontonal in bark domain
  if plot_on,
    subplot(212);
    plot(bin2bark([1:M_fft/2+1],M_fft,Fs),X_fft);
    hold on;
    if ~isempty(bin_tonal),
      plot(bin2bark(bin_tonal,M_fft,Fs),Xtm,'ro');
    else,
      plot(bin2bark(bin_nontonal,M_fft,Fs),Xnm,'ro');  % will be covered over
    end;
    plot(bin2bark(bin_nontonal,M_fft,Fs),Xnm,'go');
    plot(bin2bark([1:M_fft/2],M_fft,Fs),LTq(bin2tabindx([1:M_fft/2])),'k:');
    hold off;
    axis([0,25,-20,100]);
    xlabel('bark'); title('tonal & nontonal components, post-decimation')
    legend('fft','tonal component','nontonal component','abs threshold',0);
    subplot(211); % matlab bug...
    xlabel('bark'); title('tonal & nontonal components, pre-decimation')
    pause;
  end;

  % calculate masking thresholds for each tonal component
  LTtm = -200*ones(N_tab,length(bin_tonal));
  for t=1:N_tab,
    t_z = bin2bark(tabindx2bin(t),M_fft,Fs); % current tabindx [Bark]
    for j=1:length(bin_tonal),
      j_z = bin2bark(bin_tonal(j),M_fft,Fs);
      dz = t_z - j_z; % freq difference [Bark]
      if (-3 <= dz)&(dz < 8),
        if     (-3 <= dz)&(dz < -1),
          vf = 17*(dz+1)-(0.4*Xtm(j)+6); % "masking function"
        elseif (-1 <= dz)&(dz <0),
          vf = (0.4*Xtm(j)+6)*dz; % "masking function"
        elseif  (0 <= dz)&(dz <1),
          vf = -17*dz; % "masking function"
        elseif  (1 <= dz)&(dz <8),
          vf = -(dz-1)*(17-0.15*Xtm(j))-17; % "masking function"
        end;
        avtm = -1.525 - 0.275*j_z - 4.5; % "tonal masking index"
        LTtm(t,j) = Xtm(j) + avtm + vf; % masking at t (from jth tone)
      end;
    end;
  end;

  % calculate masking thresholds for each nontonal component
  LTnm = -200*ones(N_tab,length(bin_nontonal));
  for t=1:N_tab,
    t_z = bin2bark(tabindx2bin(t),M_fft,Fs); % current tabindx [Bark]
    for j=1:length(bin_nontonal),
      j_z = bin2bark(bin_nontonal(j),M_fft,Fs);
      dz = t_z - j_z; % freq difference [Bark]
      if (-3 <= dz)&(dz < 8),
        if     (-3 <= dz)&(dz < -1),
          vf = 17*(dz+1)-(0.4*Xnm(j)+6); % "masking function"
        elseif (-1 <= dz)&(dz <0),
          vf = (0.4*Xnm(j)+6)*dz; % "masking function"
        elseif  (0 <= dz)&(dz <1),
          vf = -17*dz; % "masking function"
        elseif  (1 <= dz)&(dz <8),
          vf = -(dz-1)*(17-0.15*Xnm(j))-17; % "masking function"
        end;
        avnm = -1.525 - 0.175*j_z - 0.5; % "nontonal masking index"
        LTnm(t,j) = Xnm(j) + avnm + vf; % masking at t (from jth comp)
      end;
    end;
  end;

  % plot individual masking thresholds
  if plot_on,
    clf;
    subplot(211);
    plot(bin2bark(bin_tonal,M_fft,Fs),Xtm,'ro');
```

```
    hold on;
    plot(bin2bark(bin_nontonal,M_fft,Fs),Xnm,'go');
    plot(bin2bark([1:M_fft/2],M_fft,Fs),LTq(bin2tabindx([1:M_fft/2])),'b');
    for j=1:length(bin_tonal),
      plot(bin2bark(tabindx2bin(1:N_tab),M_fft,Fs),LTtm(:,j),'r');
    end;
    for j=1:length(bin_nontonal),
      plot(bin2bark(tabindx2bin(1:N_tab),M_fft,Fs),LTnm(:,j),'g');
    end;
    hold off;
    axis([0,25,-20,100]);
    xlabel('bark'); title('individual masking thresholds');
  end;

  % calculate global masking threshold
  LTg_lin = 10.^(LTq/10); % quiet threshold (linear)
  for i=1:length(bin_tonal),
    LTg_lin = LTg_lin + 10.^(LTtm(:,i)/10); % tonal masking
  end;
  for i=1:length(bin_nontonal),
    LTg_lin = LTg_lin + 10.^(LTnm(:,i)/10); % nontonal masking
  end;
  LTg = 10*log10(LTg_lin); % convert to dB

  % plot global masking threshold
  if plot_on,
    subplot(212);
    plot(bin2bark([1:M_fft/2],M_fft,Fs),LTg(bin2tabindx([1:M_fft/2])),'k');
    hold on;
    plot(bin2bark([1:M_fft/2],M_fft,Fs),LTq(bin2tabindx([1:M_fft/2])),'b:');
    for j=1:length(bin_tonal),
      plot(bin2bark(tabindx2bin(1:N_tab),M_fft,Fs),LTtm(:,j),'r:');
    end;
    for j=1:length(bin_nontonal),
      plot(bin2bark(tabindx2bin(1:N_tab),M_fft,Fs),LTnm(:,j),'g:');
    end;
    hold off;
    axis([0,25,-20,100]);
    xlabel('bark'); title('global masking threshold');
    pause;
  end;

  % compute minimum global masking threshold in each subband
  for m=1:N_sb,
    rng = round( [(m-1)*M_fft/2/N_sb+1:m*M_fft/2/N_sb] );
    LTmin(m) = min( LTg(bin2tabindx(rng)) );
  end;

  % compute signal-to-mask ratio
  SMR = Lsb - LTmin;

  % plot SMR
  if plot_on,
    clf;
    subplot(211);
      bar([Lsb;LTmin]','grouped'); %hold on; bar(LTmin,'w'); hold off;
      axis([0,33,0,100]);
      legend('SPL','Minimum Mask',0);
    subplot(212);
      bar(SMR,'g');
      axe = axis; axis([0,33,axe(3:4)]);
      xlabel('subband number'); title('signal-to-mask ratio');
    subplot(211);
      xlabel('subband number'); title('SPL vs. masking threshold');
    pause;
  end;

%--------------------------------

function bark = bin2bark(bin,N_fft,Fs)

  f = (bin-1)/N_fft*Fs;
  bark = 13*atan(0.76*(f/1000)) + 3.5*atan(f/7500).^2;

%--------------------------------

function bin = bark2bin(bark,N_fft,Fs)

  bin = zeros(size(bark));
  for i=1:length(bark),
    [dum,bin(i)] = min( abs(bin2bark([1:N_fft/2],N_fft,Fs)-bark(i)) );
  end;

%--------------------------------

function bndrs =  mpeg_critbnd_bndrs

 bndrs = [1, 2, 3, 5, 6, 8, 9, 11, 13, 15, 17, 20, 23, 27, 32, 37, 45, 50,...
    55, 61, 68, 75, 81, 93, 106];

%--------------------------------

function [tabindx2bin, bin2tabindx, LTq] = mpeg_abs_thresh(Fs,N_fft,bitrate)

      % Top Edge Frequency | Crit Band rate | Absolute threshold
      TH = [
            86.13   0.850  25.87  ;   172.27   1.694  14.85  ;
           258.40   2.525  10.72  ;   344.53   3.337   8.50  ;
           430.66   4.124   7.10  ;   516.80   4.882   6.11  ;
           602.93   5.608   5.37  ;   689.06   6.301   4.79  ;
           775.20   6.959   4.32  ;   861.33   7.581   3.92  ;
           947.46   8.169   3.57  ;  1033.59   8.723   3.25  ;
          1119.73   9.244   2.95  ;  1205.86   9.734   2.67  ;
          1291.99  10.195   2.39  ;  1378.13  10.629   2.11  ;
          1464.26  11.037   1.83  ;  1550.39  11.421   1.53  ;
          1636.52  11.783   1.23  ;  1722.66  12.125   0.90  ;
          1808.79  12.448   0.56  ;  1894.92  12.753   0.21  ;
          1981.05  13.042  -0.17  ;  2067.19  13.317  -0.56  ;
          2153.32  13.577  -0.96  ;  2239.45  13.825  -1.37  ;
          2325.59  14.062  -1.79  ;  2411.72  14.288  -2.21  ;
          2497.85  14.504  -2.63  ;  2583.98  14.711  -3.03  ;
          2670.12  14.909  -3.41  ;  2756.25  15.100  -3.77  ;
          2842.38  15.283  -4.09  ;  2928.52  15.460  -4.37  ;
          3014.65  15.631  -4.60  ;  3100.78  15.795  -4.78  ;
          3186.91  15.955  -4.91  ;  3273.05  16.110  -4.97  ;
          3359.18  16.260  -4.98  ;  3445.31  16.405  -4.92  ;
          3531.45  16.547  -4.81  ;  3617.58  16.685  -4.65  ;
          3703.71  16.820  -4.43  ;  3789.84  16.951  -4.17  ;
          3875.98  17.079  -3.87  ;  3962.11  17.204  -3.54  ;
          4048.24  17.327  -3.19  ;  4134.38  17.447  -2.82  ;
          4306.64  17.680  -2.06  ;  4478.91  17.904  -1.33  ;
          4651.17  18.121  -0.64  ;  4823.44  18.331  -0.04  ;
          4995.70  18.534   0.47  ;  5167.97  18.730   0.89  ;
          5340.23  18.922   1.23  ;  5512.50  19.108   1.51  ;
          5684.77  19.288   1.74  ;  5857.03  19.464   1.93  ;
          6029.30  19.635   2.11  ;  6201.56  19.801   2.28  ;
          6373.83  19.963   2.45  ;  6546.09  20.120   2.63  ;
          6718.36  20.273   2.82  ;  6890.63  20.421   3.03  ;
          7062.89  20.565   3.25  ;  7235.16  20.705   3.49  ;
          7407.42  20.840   3.74  ;  7579.69  20.971   4.02  ;
          7751.95  21.099   4.32  ;  7924.22  21.222   4.64  ;
          8096.48  21.341   4.98  ;  8268.75  21.457   5.35  ;
          8613.28  21.676   6.15  ;  8957.81  21.882   7.07  ;
          9302.34  22.074   8.10  ;  9646.88  22.253   9.25  ;
          9991.41  22.420  10.54  ; 10335.94  22.575  11.97  ;
         10680.47  22.721  13.56  ; 11025.00  22.857  15.30  ;
         11369.53  22.984  17.23  ; 11714.06  23.102  19.33  ;
         12058.59  23.213  21.64  ; 12403.13  23.317  24.15  ;
         12747.66  23.414  26.88  ; 13092.19  23.506  29.84  ;
         13436.72  23.592  33.05  ; 13781.25  23.673  36.51  ;
         14125.78  23.749  40.24  ; 14470.31  23.821  44.26  ;
         14814.84  23.888  48.58  ; 15159.38  23.952  53.21  ;
         15503.91  24.013  58.17  ; 15848.44  24.070  63.48  ;
         16192.97  24.124  68.00  ; 16537.50  24.176  68.00  ;
         16882.03  24.225  68.00  ; 17226.56  24.271  68.00  ;
         17571.09  24.316  68.00  ; 17915.63  24.358  68.00  ;
         18260.16  24.398  68.00  ; 18604.69  24.436  68.00  ;
         18949.22  24.473  68.00  ; 19293.75  24.508  68.00  ;
         19638.28  24.541  68.00  ; 19982.81  24.573  68.00
      ];
      N = length(TH(:, 1));

      % Convert N frequencies in table from Hz to FFT bins.
      tabindx2bin = round(TH(:,1)/Fs*N_fft);

      % Generate a mapping between FFT bins and table indicies.
      for j = 1:tabindx2bin(1),
         bin2tabindx(j) = 1; % first index
      end
      for i = 2:N-1,
         for j = tabindx2bin(i):tabindx2bin(i+1,1),
            bin2tabindx(j) = i; % middle indicies
         end
      end
      for j = tabindx2bin(N):N_fft/2,
         bin2tabindx(j) = N; % last index
      end

      % An offset depending on the overall bit rate is used for the absolute
      % threshold. This offset is -12dB for bit rates >=- 96kbits/s and 0dB
      % for bit rates < 96 kbits/s per channel.
      if (bitrate >= 96)
         LTq = TH(:,3) - 12;
      else
         LTq = TH(:,3);
      end
```

PQF Analysis Bank:

```
% Polyphase Quadrature Filterbank: Analysis
%
% usage:
%   S = pqf_anal(x,h,N);
% where:
%   N   : number of subband branches
%   h     : coefficients of lowpass prototype filter (cutoff at pi/2N radians)
%   x     : vector of time-domain input samples
%   S     : matrix of decimated subband outputs (having N rows)
%
% Copyright 1999 --- Phil Schniter

function S = pqf_anal(x,h,N);

 % determine constants
 M = length(h);
 L = length(x);
 if N*floor(M/N)~=M, error('Filter length must be a multiple of N!'); end;

 % zero-pad input for convenience
 xp = [zeros(1,N-1),x(:).',zeros(1,N-mod(L+N-1,N)),zeros(1,M)];

 % create analysis window
 hw = 2*h;
 for k=1:2:M/2/N,
   hw(k*2*N+[1:2*N]) = -hw(k*2*N+[1:2*N]);
 end;

 % analysis
 W = zeros(2*N,ceil(length(xp)/N));
 xx = zeros(1,M);
 for m=1:ceil(length(xp)/N),
```

```
  % load new input block
  xx = [xp((m-1)*N+[N:-1:1]), xx(1:M-N)];

  % window using filter coeffs
  xh = xx.*hw;

  % combine
  for k=1:2*N,
    W(k,m) = sum(xh(k:2*N:M));
  end;
end;

% IDCT
Wbar = zeros(N,ceil(length(xp)/N));
Wbar(1,:) = sqrt(2)*W(N/2+1,:);
Wbar(2:N/2+1,:) = W(N/2+2:2*N/2+1,:) + W(N/2:-1:1,:);
Wbar(N/2+2:N,:) = W(N+2:N/2+N,:) - W(2*N:-1:N+N/2+2,:);
S = sqrt(N/2)*idct(Wbar);
```

PQF Synthesis Bank:

```
% Polyphase Quadrature Filterbank: Synthesis
%
% usage:
%   x = pqf_synth(S,h);
% where:
%   S     : matrix of decimated subband outputs (N rows <=> N subbands)
%   h     : coefficients of lowpass prototype filter (cutoff at pi/2N radians)
%   x     : vector of time-domain reconstructed samples
%
% Copyright 1999 --- Phil Schniter

function u = pqf_synth(S,h);

N = size(S,1);
L = size(S,2);
M = length(h);

% create synthesis window
hw = 2*h; % last sample is zero
for k=1:2:M/2/N,
  hw(k*2*N+[1:2*N]) = -hw(k*2*N+[1:2*N]);
end;

% fast cosine matrix transformation
Vbar = sqrt(N/2)*dct(S); Vbar(1,:) = sqrt(2)*Vbar(1,:);
Vnew = zeros(2*N,L);
Vnew(1:N/2,:) = Vbar(N/2+1:2*N/2,:);
Vnew(N/2+2:3*N/2,:) = -Vbar(2*N/2:-1:2,:);
Vnew(3*N/2+1:2*N,:) = -Vbar(1:2*N-3*N/2,:);

% synthesis
vv = zeros(1,2*M);
v = zeros(1,M);
u = zeros(1,N*L);
for m=1:L,
  % insert new input block
  vv = [Vnew(:,m).',vv(1:2*M-2*N)];

  % extract valid sub-blocks
  for k=0:M/2/N-1,
    v(k*2*N+1:(k+1)*2*N) = vv(k*4*N+[1:N,3*N+1:4*N]);
  end;

  % window using filter coeffs
  vh = v.*hw;

  % combine
  for k=1:N,
    u((m-1)*N+k) = sum(vh([k:N:M]));
  end;
end;
```