Scripts and functions involved in the code arranged alphabetically.

CCLIP.m

```
% BE491 Group CCLIP
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function clipped = cclip(x,minval,maxval)
%CCLIP Performs center clipping of input signal
%   Y = CCLIP(X,MINVAL,MAXVAL) center clips the signal X. MINVAL and MAXVAL set
%   the lower and upper clipping threshold, respectively. Signal components
%   between MINVAL and MAXVAL are 'center clipped', while components below
%   MINVAL are shifted up and compoents above MAXVAL are shifted down. MINVAL
%   must be negative and MAXVAL must be positive. Each elements of X is
%   processed as follows:
%      If X(i) > MAXVAL, then Y(i) = X(i) - MAXVAL;
%      If MINVAL < X(i) < MAXVAL, then Y(i) = 0;
%      If X(i) < MINVAL, then Y(i) = X(i) - MINVAL;

%% Check input arguments
if nargin < 3
    error('You must enter three input arguments.');
end;
if (size(x,1) > 1) && (size(x,2) > 1)
   error ('Signal must be a vector');
end
if (length(maxval) > 1) || (length(minval) > 1)
   error ('Minimum and maximum values must be scalars');
end
if (maxval < 0) || (minval > 0)
   error ('Minimum value must be negative and maximum value must be positive');
end

% Perform center clipping
x = x(:);
nx = length(x);
zz = zeros(nx,1);
oo = ones(nx,1);
maxx = maxval * oo;
minn = minval * oo;

upper = max(x-maxx,zz);
lower = min(x-minn,zz);
clipped = upper + lower;
```

CHVOC_MAIN.m

```
% BE491 Group MAIN SCRIPT for saving sound files and formatting figures in
% the presentation and report
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

%% This script is designed to:
%       (1) Record an utterance and format this object into a format that
%           can be read into the channel vocoder.
%       (1) Determine the pitch values produced for the entire utterance and
%           compare to the performance of the automated pitch detector to
%           the original signal. Though it will not be perfect, our results
```

```
%           are reasonable.
%       (2) Produce monotone, whispered, male, and female utterances by
%           changing the pitch vector.

%% Record utterance and format for the channel vocoder
%{
Set recording time
duration = 5; %s
% Create recording object
Fs_o = 44100;
SNDREC = audiorecorder(Fs_o, 16, 1);
    % creates a 16 bit, 1 channel audiorecorder object

% Collection
pause(1)
disp('Start speaking.');
recordblocking(SNDREC, duration);
disp('End of recording.');

% Extract data
snd.data = getaudiodata(SNDREC);
audiowrite('signal_44k_BlakeP.wav', snd.data, Fs_o);
% obj_rec = audioplayer(signal_object.data, Fs_o);

% Resample the data at 8kHz to minimize processing time and work best with
chvoc
signal_o = resample(snd.data, 2, 11);
Fs = 8E3;

% Blake: Not sure if lines 39-46 are absolutely necessary, but they seem to
be nice
% Write audio file from data
    % Option to save original, unsampled recording:
    % audiowrite('signal_original.wav', signal_object.data, Fs_o);
audiowrite('signal_8k_BlakeP.wav', signal_o, Fs);

% Read
[signal_o, Fs] = audioread('signal_8k_Blake2.wav');
% soundsc(signal_o, Fs);
%}
%% Run through the Channel Vocoder
% The empty flask stood on the tin tray
load cw161_8k.mat
Fs = 8000; %Hz
signal_o = cw161;
audiowrite('cw161_o.wav', signal_o/norm(signal_o,inf), Fs);
D = 10;
N = 18;
[signal_synPI, Fs] = chvoc_over(signal_o, D, N, Fs, 'PI');
sound(signal_synPI)
audiowrite('cw161_synPI.wav', signal_syn, Fs);
    % chvoc_over generates a NORMALIZED signal synthesized in the channel
vocoder,
    %  as well as returning the Fs and the pitch vector
    % Inputs include:
```

```
    % varargin{1} can be used to specify the sampling frequency, default
8kHz
    % varargin{2} can also be used as a string input to change the voice:
        % ORIGINAL: Leave pitch vector (p) as is; this is default
        %  p = p;
        % FEMALE: Multiply pitch vector (p) by a factor of 2
        %  p = p * 2;
        % MALE:   Multiply pitch vector (p) by a factor of 0.5
        %  p = p * 0.5;
        % WHISPER: Set pitch vector (p) to zeros
        %  p = zeros(1,length(p));
        % MONOTONE: Set pitch vector (p) to contsant value (eg. 100Hz)
        %  p= ones(1,length(p)).*100;

%% Time Domain Plot
%{
figure
plot((0:length(signal_o)-1)/Fs, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
hold on
plot((0:length(signal_syn)-1)/Fs, signal_syn, 'Color', [0.302 0.745 0.933],
'Linewidth', 2)
xlabel('Time (s)', 'FontSize', 30)
ylabel('Normalized Amplitude', 'FontSize', 30)
str = sprintf('Time Domain of Recorded Utterance:\nNormalized Amplitude v.
Time');
title(str,'FontSize', 35)
legend('Recorded', 'Synthesized in Channel Vocoder')
axis([-0.1 3.1 -1.1 1.1])
set(gca, 'FontSize', 20)
%}
%% Spectrogram plot

%{
figure
subplot(2,1,1)
[So,Fo,To] = spectrogram(snd.data,2^10,2^9,[],Fs_o);
set(gcf,'windowstyle','docked')
imagesc(To,Fo,20*log10(abs(So)),[-126 34])
colorbar
axis xy
set(gca, 'FontSize', 25)
xlabel('Time (s)','FontSize', 35)
ylabel('Frequency (Hz)','FontSize', 35)
title('Spectrogram for Utterrance as Originally Recorded','FontSize', 35)
ylim([0 Fs_o/2])

%}
figure
subplot(1,3,1)
[So,Fo,To] = spectrogram(signal_o/norm(signal_o,inf),2^10,2^9,[],Fs);
set(gcf,'windowstyle','docked')
imagesc(To,Fo,20*log10(abs(So)),[-126 34])
% colorbar
axis xy
set(gca, 'FontSize', 25)
```

```
xlabel('Time (s)','FontSize', 35)
ylabel('Frequency (Hz)','FontSize', 30)
str = sprintf('Original Utterance');
title(str,'FontSize', 30)
ylim([0 Fs/2])

subplot(1,3,2)
[S_syn,F_syn,T_syn] = spectrogram(signal_synOR,2^10,2^9,[],Fs);
% set(gcf,'windowstyle','docked')
imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
% colorbar
set(gca, 'FontSize', 25)
axis xy
xlabel('Time (s)','FontSize', 35)
ylabel('Frequency (Hz)','FontSize', 30)
str = sprintf('"Original" Synthesized');
title(str,'FontSize', 30)
ylim([0 Fs/2])

subplot(1,3,3)
[S_syn,F_syn,T_syn] = spectrogram(signal_synMA,2^10,2^9,[],Fs);
% set(gcf,'windowstyle','docked')
imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
colorbar
set(gca, 'FontSize', 25)
axis xy
xlabel('Time (s)','FontSize', 35)
ylabel('Frequency (Hz)','FontSize', 30)
str = sprintf('"Male" Synthesized');
title(str,'FontSize', 30)
ylim([0 Fs/2])

%%
figure
set(gcf,'windowstyle','docked')
subplot(1,3,1)
[S_syn,F_syn,T_syn] = spectrogram(signal_synFE,2^10,2^9,[],Fs);
% set(gcf,'windowstyle','docked')
imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
% colorbar
set(gca, 'FontSize', 25)
axis xy
xlabel('Time (s)','FontSize', 35)
ylabel('Frequency (Hz)','FontSize', 30)
str = sprintf('"Female" Synthesized');
title(str,'FontSize', 30)
ylim([0 Fs/2])

subplot(1,2,2)
[S_syn,F_syn,T_syn] = spectrogram(signal_synMO,2^10,2^9,[],Fs);
% set(gcf,'windowstyle','docked')
imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
% colorbar
set(gca, 'FontSize', 25)
axis xy
xlabel('Time (s)','FontSize', 35)
```

```
ylabel('Frequency (Hz)','FontSize', 30)
str = sprintf('"Monotone" Synthesized');
title(str,'FontSize', 30)
ylim([0 Fs/2])

subplot(1,3,3)
[S_syn,F_syn,T_syn] = spectrogram(signal_synWH,2^10,2^9,[],Fs);
% set(gcf,'windowstyle','docked')
imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
% colorbar
set(gca, 'FontSize', 25)
axis xy
xlabel('Time (s)','FontSize', 35)
ylabel('Frequency (Hz)','FontSize', 30)
str = sprintf('"Whispered" Synthesized');
title(str,'FontSize', 30)
ylim([0 Fs/2])
%%
subplot(1,3,3)
[S_syn,F_syn,T_syn] = spectrogram(signal_synWH2,2^10,2^9,[],Fs);
% set(gcf,'windowstyle','docked')
imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
% colorbar
set(gca, 'FontSize', 25)
axis xy
xlabel('Time (s)','FontSize', 35)
ylabel('Frequency (Hz)','FontSize', 30)
str = sprintf('"Pitchless" Synthesized');
title(str,'FontSize', 30)
ylim([0 Fs/2])
```

CHVOC_OVER.m

```
% BE491 Group Digital Channel Vocoder over for simple call from GUI
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function [signal_syn, Fs, p] = chvoc_over(signal_o, D, N, varargin)
%% chvoc_over generates a NORMALIZED signal synthesized in the channel
vocoder,
%  as well as returning the Fs and the pitch vector
% Inputs include:
% varargin{1} can be used to specify the sampling frequency, default 8kHz
% varargin{2} can also be used as a string input to change the voice:
    % ORIGINAL: Leave pitch vector (p) as is; this is default
    %  p = p;
    % FEMALE: Multiply pitch vector (p) by a factor of 5/3
    %  p = p * 5/3;
    % MALE:   Multiply pitch vector (p) by a factor of 3/4
    %  p = p * 3/4;
    % WHISPER: Set pitch vector (p) to zeros
    %  p = zeros(1,length(p));
    % MONOTONE: Set pitch vector (p) to contsant value (eg. 100Hz)
    %  p= ones(1,length(p)).*100;
```

```
%% Address inputs
% Change signal length to standard
signal_o = [signal_o; zeros(ceil(size(signal_o,1)/D)*D-size(signal_o,1),1)];
% Fs
if nargin < 4
    Fs = 8E3; %Hz
else
    Fs = varargin{1};
end
% type
if nargin < 5
    type = 'OR';
elseif nargin == 5 && ischar(varargin{2})
    type = upper(varargin{2}(1:2));
else
    error('Incorrect formatting of chvoc_over inputs.\n')
end
%% Run utterence through the channel vocoder analyzer
[y,p] = chvocod_ana(signal_o, D, N, Fs);
    %CHVOCOD_ANA Channel vocoder analyzer
    %   [BAND_ENVELOPES,PITCH] = CHVOCOD_ANA(X,DECIMATE,N)
    %   encodes speech signal into pitch values and band envelope values
    %   corresponding to a number of frequency channels
    %   X               UNFILTERED speech signal that will be split into 30
ms frames
    %   N               Number of frequency bands into which each 30ms frame
is
    %                   split, enveloped, lowpass filtered, and decimated
    %   varargin/Fs     Sampling frequency, default 8kHz
    %   DECIMATE        Decimation factor by which the signal is decimated
    %   BAND_ENVELOPES Y, Output return of decimated band envelope values,
    %                   a matrix with size num_frames by N (where
    %                   num_frames is the number of data frames dividing the
    %                   signal).
    %   PITCH           P, The pitch of each frame is detected by the pitch
    %                   detector, and the pitch outputs are returned in the
    %                   output variable.
    %
    %   This code has two separate stages, corresponding to the
    %   source-filter model of speech production:
    %   (1) The first stage involves characterizing the "source" by pitch
detection.
    %       Pitch detection is accomplished by breaking up the original
signal
    %       into frames and then determining if each frame is is voiced or
unvoiced.
    %       If the frame is voiced, then we also estimate the fundamental
frequency
    %       of the glottal source.
    %   (2) The second stage involves characterizing the "filter", that is
    %       determining the band envelope values.  This is accomplished by
filtering
    %       the original signal into frequency bands, determining the
envelope of
    %       each band and decimating.
```

```
%% Run the pitch vector through the channel vocoder synthesizer within each
type
% signal_syn = chvocod_syn(y,p,D);
    % CHVOCOD_SYN  Synthesizes speech waveform from pitch and band envelope
signals
    %   OUT = CHVOCOD_SYN(BAND_ENVELOPES,PITCH,UPSAMPLE) synthesizes the
speech
    %   signal encoded by a channel vocoder with frequency band envelopes
specified
    %   by matrix BAND_ENVELOPES and pitch values specified by vector PITCH.
The
    %   signal is upsampled by the value specified in UPSAMPLE. An optional
input,
    %   varargin/Fs is the sampling frequency, where the default is 8kHz.
    %
    %   Each column of BAND_ENVELOPES contains all frame information within
each
    %   frequency band. Each row of BAND_ENVELOPES contains all frequency
band
    %   information within each data frame. PITCH contains the pitch
information for
    %   each data frame.
%% TYPE
if type == 'MO'
    % MONOTONE: Set pitch vector (p) to contsant value (eg. 100Hz)
    p= ones(1,length(p)).*100;
    % Run the pitch vector through the channel vocoder synthesizer
    signal_syn = chvocod_syn(y, p, D);
    signal_syn = signal_syn/norm(signal_syn, inf);
elseif type == 'FE'
    % FEMALE: Multiply pitch vector (p) by a factor of 5/3
    p = p * 5/3;
    % Run the pitch vector through the channel vocoder synthesizer
    signal_syn = chvocod_syn(y, p, D);
    signal_syn = voc_p(signal_o, 3/5);
    signal_syn = resample(signal_syn, 3, 5);
    signal_syn = signal_syn/norm(signal_syn, inf);
elseif type == 'MA'
    % MALE:   Multiply pitch vector (p) by a factor of 3/4
    p = p * 3/4;
    % Run the pitch vector through the channel vocoder synthesizer
    signal_syn = chvocod_syn(y, p, D);
    signal_syn = voc_p(signal_o, 4/3);
    signal_syn = resample(signal_syn, 4,3);
    signal_syn = signal_syn/norm(signal_syn, inf);
elseif type == 'PI'
    % PITCHLESS: Set pitch vector (p) to zeros
    p = zeros(1,length(p));
    % Run the pitch vector through the channel vocoder synthesizer
    signal_syn = chvocod_syn(y, p, D);
    signal_syn = signal_syn/norm(signal_syn, inf)*0.3;
elseif type == 'WH'
    % WHISPER: Set pitch vector (p) to whitenoise (an aperiodic signal with
    % random frequencies of equal intensities)
    p = sqrt(2)*randn(length(p),1);
    % Run the pitch vector through the channel vocoder synthesizer
    signal_syn = chvocod_syn(y, p, D);
```

```
    signal_syn = signal_syn/norm(signal_syn, inf)*0.3;
else
    % ORIGINAL: Leave pitch vector (p) as is
    %  p = p;
    % Run the pitch vector through the channel vocoder synthesizer
    signal_syn = chvocod_syn(y, p, D);
    signal_syn = mean([signal_o/norm(signal_o, inf)
signal_syn/norm(signal_syn, inf)],2);
end

% soundsc(signal_syn);
```

CHVOC_VOWEL.m

```
% BE491 Group Project Vowel Pitch Detection
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

%% This script is designed to show the inner workings of the analyzer by
generating:
%        (1) plot the time domain of a stressed vowel,
%        (2) plot the low-pass filtered time domain signal,
%        (3) plot the center-clipped time domain signal,
%        (4) plot the autocorrelation plot and demarcate the peak for
%            fundamental frequency. (In the main script this is done
%            automatically, but for explanation's sake, we will show the
%            process here.)

%% Load the data in question
%  If the data is long, clip it to only a stressed vowel.
load cw161_8k.mat
% soundsc(cw161)
% Note the sampling rate
Fs = 8000; %Hz

% Isolate vowel
%   For this recording, I will clip to "ay."
 vowel = cw161(1980:2620); % "em"
% vowel = cw161(2.07E4:2.189E4);
% vowel = cw161(2.0E4:2.3E4);
soundsc(vowel)

%% Plot the time domain of a stressed vowel
figure
subplot(3,1,1)
% Plot the normalized magnitude of the utterance against a time axis
plot((0:length(vowel)-1)/Fs, vowel/norm(vowel,inf), 'b-','Linewidth',2)
set(gca, 'FontSize', 20)
xlabel('Time (s)', 'FontSize', 30)
ylabel('Normalized Amplitude', 'FontSize', 30)
str = sprintf('Time Domain of Utterance');
title(str,'FontSize', 35)
legend('Original')
axis([-0.01 0.09 -1.1 1.1])
```

```
%% Plot the low-pass filtered time domain signal

subplot(3,1,2)
% Using code from chvocod_ana.m lines 46-51 and 62-67
    % Set Nyquist frequency
    Fny = Fs/2; %Hz
    % Set low cutoff frequency of low pass filter for filtering the signal
    % [pitch is only 80-320 Hz for adult voices]
    FL = 350; %Hz
    % Set filter order to filter the speech signal
    order = 200;
    % Design lowpass filter by the windowing method
    Bfir1 = fir1(order, FL/Fny);
    % Filtering the speech signal
    vowel_lpf = fftfilt(Bfir1, vowel);

% Plot the normalized magnitude of the utterance against a time axis
plot((0:length(vowel_lpf)-1)/Fs, vowel_lpf/norm(vowel_lpf,inf), 'Color', [0
0.447 0.741],'Linewidth',2)
set(gca, 'FontSize', 20)
xlabel('Time (s)', 'FontSize', 30)
ylabel('Normalized Amplitude', 'FontSize', 30)
str = sprintf('Time Domain of Low-Pass Filtered Utterance');
title(str,'FontSize', 35)
legend('200th-Order 350Hz LPF')
axis([-0.01 0.09 -1.1 1.1])

%% Plot the unoffset, center-clipped time domain signal
subplot(3,1,3)
% Using code from pitch_detect.m lines 39-51
    % Remove DC offset
    vowel_cclip = vowel - mean(vowel);
    % Find min and max samples, account for thresholds, and center clip
using cclip function
    vowel_cclip = cclip(vowel_cclip, min(vowel_cclip)*0.75,
max(vowel_cclip)*0.75);
    %{
        Center clips the signal x setting the lower and upper clipping
        thresholds from the MINVAL and MAXVAL respectively.
        Signal components between MINVAL and MAXVAL are 'center clipped',
        while components below MINVAL are shifted up and compoents above
        MAXVAL are shifted down. MINVAL must be negative and MAXVAL must
        be positive. Each elements of X is processed as follows:
        If X(i) > MAXVAL, then Y(i) = X(i) ? MAXVAL;
        If MINVAL < X(i) < MAXVAL, then Y(i) = 0;
        If X(i) < MINVAL, then Y(i) = X(i) - MINVAL;
        Motivation:
        In order to use the autocorrelation function for automatic pitch
        detection, it is helpful to suppress the peaks due to the vocal
        tract transfer function. This center clipping will accomplish the
        suppression.
    %}

% Plot the normalized magnitude of the utterance against a time axis
```

```
plot((0:length(vowel_cclip)-1)/Fs, vowel_cclip/norm(vowel_cclip,inf),
'Color',[0.302 0.745 0.933],'Linewidth',2)
set(gca, 'FontSize', 20)
xlabel('Time (s)', 'FontSize', 30)
ylabel('Normalized Amplitude', 'FontSize', 30)
str = sprintf('Time Domain of Unoffset, Center-Clipped Utterance');
title(str,'FontSize', 35)
leg = sprintf('DC-Offset removed,\nCenter-Clipped to 75%');
legend(leg)
axis([-0.01 0.09 -1.1 1.1])

%% Plot the autocorrelation plot
% Using code from pitch_detect.m lines 53-56
    % Compute the autocorrelation of the frame
    Rx = xcorr(vowel_cclip,'coeff');
    % Calculates the autocorrelation and also normalizes to 1
    % Note that the zeroth lag of the correlation, Rx[0], is in the middle
of the output sequence.
    % Find the maximum peak following Rx[0] by calling peak function

%% Demarcate the peak for fundamental frequency
% and print the number in the command window
pitch = pitch_detect(vowel_lpf);

% Using code from pitch_detect.m lines 58-69
    % Find the maximum peak following Rx[0] by calling peak function
    % To find the index of the maximum value; should be at Rx[0]
    max_index = find(Rx == max(Rx));
    % Extract the positive part of the correlation (e.g. on x axis)
    Rx_pos = Rx(max_index: length(Rx));
    % Find the maximum peak following Rx[0]
    [peakVAL, index] = peak(Rx_pos);
    %PEAK Detects autocorrelation fundamental peak
    %   [PEAKVAL, PEAKINDEX] = PEAK(X) locates the value and index of the
largest
    %   peak in the vector X other than Rx[0].  X must be an autocorrelation
    %   function with maximum value Rx[0] as its first element.

% Plot the autocorrelation plot
figure
plot(((1:length(Rx))-max_index)/Fs, Rx, 'b','Linewidth',1)
hold on
% Plot the fundamental frequency
plot((index)/Fs, peakVAL,'o',...
    'LineWidth',5,...
    'MarkerSize',20,...
    'MarkerEdgeColor', [1 0 1],...
    'MarkerFaceColor', [1 0.8 1])
set(gca, 'FontSize', 35)
xlabel('Time Lag (s)', 'FontSize', 40)
ylabel('Normalized Correlation', 'FontSize', 40)
str = sprintf('Autocorrelation of Utterance');
title(str,'FontSize', 45)
legend('Autocorrelation', 'Fundamental Peak')
axis([-0.085 0.085 -0.3 1.1])
```

```
fprintf('The fundamental frequency of the utterance is %d.\n',pitch)
```

CHVOCOD_ANA.m

```
% BE491 Group Digital Channel Vocoder Analyzer
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function [y,p] = chvocod_ana(x, D, N, varargin)
%CHVOCOD_ANA Channel vocoder analyzer
%   [BAND_ENVELOPES,PITCH] = CHVOCOD_ANA(X,DECIMATE,N)
%   encodes speech signal into pitch values and band envelope values
%   corresponding to a number of frequency channels
%   X               UNFILTERED speech signal that will be split into 30 ms
frames
%   N               Number of frequency bands into which each 30ms frame is
%                   split, enveloped, lowpass filtered, and decimated
%   varargin/Fs     Sampling frequency, default 8kHz
%   DECIMATE        Decimation factor by which the signal is decimated
%   BAND_ENVELOPES Y, Output return of decimated band envelope values,
%                   a matrix with size num_frames by N (where
%                   num_frames is the number of data frames dividing the
%                   signal).
%   PITCH           P, The pitch of each frame is detected by the pitch
%                   detector, and the pitch outputs are returned in the
%                   output variable.
%
%   This code has two separate stages, corresponding to the
%   source-filter model of speech production:
%   (1) The first stage involves characterizing the "source" by pitch
detection.
%       Pitch detection is accomplished by breaking up the original signal
%       into frames and then determining if each frame is is voiced or
unvoiced.
%       If the frame is voiced, then we also estimate the fundamental
frequency
%       of the glottal source.
%   (2) The second stage involves characterizing the "filter", that is
%       determining the band envelope values.  This is accomplished by
filtering
%       the original signal into frequency bands, determining the envelope
of
%       each band and decimating.
%% Initialize variables
% Make x a column vector just to be sure.
x = x(:,1);
% Set sampling frequency
if nargin == 4
    Fs = varargin{1};
elseif nargin == 3
    Fs = 8000; % Hz
else
    error('You must enter 3 or 4 input arguments.');
end
% Set Nyquist frequency
```

```
Fny = Fs/2;
% Set low cutoff frequency of low pass filter for filtering the signal
FL = 350;  % [pitch is only 80-320 Hz for adult voices]
% Set filter order to filter the speech signal
order = 200;
% Set 30 ms frame length
frlen = floor(0.030 * Fs);
% Set frame number
    % Note that this depends on decimation rate
nframes = ceil(length(x)/D);
% Preallocate pitch vector output for speed
p = zeros(nframes, 1);
% Preallocate output matrix "y" for efficiency.
y = zeros(nframes,N);

%% Retrieve "source parameters" (pitch detection)
% (i) LPF the signal with 350 Hz cutoff frequency,
%     [pitch is only 80-320 Hz for adult voices]
%filtering the vowel to preserve only frequencies below 350Hz
Bfir1 = fir1(order, FL/Fny); %design lowpass filter by the windowing method
xlpf = fftfilt(Bfir1,x); %filtering the speech signal

%% Loop
% Each iteration processes one frame of data.
for i = 1:nframes
    startseg = (i-1)*D+1;
    endseg = startseg+frlen-1;
    if endseg > length(xlpf)
        endseg = length(xlpf);
    end
    seg = xlpf(startseg:endseg);
    % Call the pitch detector
    p(i) = pitch_detect(seg);
    % Algorithm to determine the fundamental frequency of the voice
        % f = pitch__detect(filtered_signal);
        % x: a vector containing frame of speech data sampled at 8 kHz
        % clip_thresh: a scaling factor of the max/min values, 0.75 (75%)
default
        % unvoiced_thresh: the minimum allowable relative pitch peak
amplitude,
        % below which the segment is considered unvoiced, 0.25 (25%) default
        % f: a scalar containing pitch of frame in Hz or 0 if unvoiced
end
% Remove spurious values from pitch signal with median filter
p = medfilt1(p, 65);

%% Determine band envelope values by setting filter parameters
% Compute FIR coefficients for filter bank (using 65-point filters).
% The variable bank should be a 65xN matrix with each column containing
% the impulse response of one filter
bank = filt_bank(N, 65);

%% Apply the filterbank to the input signal, x
% Process each band by looping
for i = 1:N
    % Apply filter for this band (bank(:,i)) to input x
```

```
     segbpf = fftfilt(bank(:,i),x);
    % Take magnitude of signal and decimate.
     segbpf = abs(segbpf);
    %Decimate
        % Note that MATLAB fcn 'decimate.m' includes lowpass filtering
     y(:,i) = decimate(segbpf,D);
% At this point, each row in Y has a length of D (number of points)
% resulting from the decimation. Since we also have D number of segments,
% Y is a matrix in which each column represents a segment of the signal
% and each row represents a band in the filter bank
end
```

CHVOCOD_SYN.m

```
% BE491 Group Digital Channel Vocoder Synthesizer
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function y = chvocod_syn(band_envelopes, pitch, R, varargin)
%CHVOCOD_SYN  Synthesizes speech waveform from pitch and band envelope
signals
%   OUT = CHVOCOD_SYN(BAND_ENVELOPES,PITCH,UPSAMPLE) synthesizes the speech
%   signal encoded by a channel vocoder with frequency band envelopes
specified
%   by matrix BAND_ENVELOPES and pitch values specified by vector PITCH. The
%   signal is upsampled by the value specified in UPSAMPLE. An optional
input,
%   varargin/Fs is the sampling frequency, where the default is 8kHz.
%
%   Each column of BAND_ENVELOPES contains all frame information within each
%   frequency band. Each row of BAND_ENVELOPES contains all frequency band
%   information within each data frame. PITCH contains the pitch information
for
%   each data frame.
%% Initialize variables
% Set sampling frequency
if nargin == 4
    Fs = varargin{1};
elseif nargin == 3
    Fs = 8000; % Hz
else
    error('You must enter 3 or 4 input arguments.');
end

% Length of each frame in samples
frame_length = R;
% Determine number of bands from input matrix
N = size(band_envelopes,2);
% Compute FIR coefficients for the filter bank
L = 65;  % length of each filter
bank = filt_bank(N,L);

% Generate a voiced source signal using pulse_train
src = sw_source(pitch,Fs,frame_length);
```

```
% Compute length of source signal
M = length(src);

% Preallocate output matrix for efficiency
ybands = zeros(M,N);

% In loop, process each band:
for i = 1:N
    % Interpolate (upsample) each decimated band envelope
    % and replace any negative values with zeros
    xint = interp(band_envelopes(:,i),R);
    xint(xint<0) = 0;

    % Multiply with source, trimming the interpolated signal to
    % match pulse train length, M.
    yint = xint(1:M) .* src;

    % Apply bandpass filter . . .
    ybands(:,i) = fftfilt(bank(:,i),yint);
end

% Add up the output of all of the bands to generate result
y = sum(ybands,2);
y(y<0) = y(y<0)/abs(min(y)) * max(y);
```

ECHO_GUI.m

```
% BE491 Group ECHO GUI
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function RecordTemplate
%% Establishes settings to allow recording

record_dur = 5;
%Time in secs to record
%Can be set in GUI

sampRate_rec = 44100;
% Sampling rate in Hz
Fs = 8000;
%lower sampling rate

SNDREC = audiorecorder(sampRate_rec,16,1);
%creates an object to save recording data

signal_syn=[];
signal_syn_MO=[];
signal_syn_FE=[];
signal_syn_MA=[];
signal_syn_WH=[];
signal_syn_PI=[];
```

```
signal_o=[];
%proallocates signal vectors to span all workspaces

rec=0;
%creates variable to check if a recoding exists




%% Creates Figure Window

f=figure('Visible','off','color','white','Position', [50,50,1200,650]);
%Creates window set to turn off all features
set(f,'Name','Record Template')
%Adds a name to the window


%% Adds TEXT
% sentence=uicontrol('Style','text',...
%     'BackgroundColor','green', ...
%     'FontSize',30,...
%     'Units','normalized',...
%     'Position',[.1, .9, .8, .1],...
%     'String', 'The empty flask stood on the tin tray');
% %create txt to show sentence to be read



Info=uicontrol('Style','text',...
    'BackgroundColor','white', ...
    'Units','normalized',...
    'Position',[.60, .15, .4, .05],...
    'String', '');
%create txt to pass info to user
%left blank to begin



Label=uicontrol('Style','text',...
    'BackgroundColor','white', ...
    'Units','normalized',...
    'Position',[.1, .2, .05, .025],...
    'String', 'Record Time');
%create txt to label the recording time input


%% Adds the BUTTONS to the GUI


RecordButton=uicontrol('Style','pushbutton',...
    'String','Record',...
    'Units','normalized',...
    'Position',[.05,.15,.05,.05],...
```

```
    'Callback',@Record);
%creates a button that start the RECORDING


PlaybackButton=uicontrol('Style','pushbutton',...
    'String','Play Unaltered',...
    'Units','normalized',...
    'Position',[.15,.15,.075,.05],...
    'Callback',@PlaybackO);
%creates a button that will start the PLAYBACK Unaltered


PlaybackButton=uicontrol('Style','pushbutton',...
    'String','Play Original',...
    'Units','normalized',...
    'Position',[.225,.15,.075,.05],...
    'Callback',@Playback);
%creates a button that will start the PLAYBACK ORIGINAL



PlaybackButton=uicontrol('Style','pushbutton',...
    'String','Play Monotone',...
    'Units','normalized',...
    'Position',[.30,.15,.075,.05],...
    'Callback',@PlaybackMono);
%creates a button that will start the PLAYBACK MONOTONE

PlaybackButton=uicontrol('Style','pushbutton',...
    'String','Play Whisper',...
    'Units','normalized',...
    'Position',[.375,.15,.075,.05],...
    'Callback',@PlaybackWhisp);
%creates a button that will start the PAYBACK WHISPER

PlaybackButton=uicontrol('Style','pushbutton',...
    'String','Play Male',...
    'Units','normalized',...
    'Position',[.45,.15,.075,.05],...
    'Callback',@PlaybackMale);
%creates a button that will start the PLAYBACK Male

PlaybackButton=uicontrol('Style','pushbutton',...
    'String','Play Female',...
    'Units','normalized',...
    'Position',[.525,.15,.075,.05],...
    'Callback',@PlaybackFemale);
%creates a button that will start the PLAYBACK Female

PlaybackButton=uicontrol('Style','pushbutton',...
    'String','Play Pitchless',...
    'Units','normalized',...
    'Position',[.6,.15,.075,.05],...
    'Callback',@PlaybackPitchless);
```

```
%creates a button that will start the PLAYBACK PITCHLESS


%% Adds NUMBER INPUT to set recording time

RecordTime=uicontrol('Style','edit',...
    'Units','normalized',...
    'Position',[.10,.15,.05,.05],...
    'String',num2str(record_dur));
%create input string to allow input of recoding time
%units are seconds
%% Prealocates locations


%Freq plot 2

TR=[.55,.65,.3,.25];
subplot('Position',TR)
axis off


%Time plot 2

BR=[.55,.275,.3,.25];
subplot('Position',BR)
axis off

%Freq plot 1

TL=[.05,.65,.3,.25];
subplot('Position',TL)
axis off

%Time plot 1

BL=[.05,.275,.3,.25];
subplot('Position',BL)
axis off










%% Makes GUI visible
set(f,'Visible','on')
%Turns on all features. This allows the buttons/text to be loaded quickly

%% Sub functions
```

```
%% RECORDING SUBFUNCTION
function Record(hObject, eventdata)
    record_dur =str2double(get(RecordTime,'String'));
    %gets length of recording from GUI

    if record_dur<=0
        %ERROR CHECKING

        set(Info, 'BackgroundColor','red')
        %Background changed to red to emphasize error
        set(Info, 'String', 'Record time must be greater than zero')
        %if else used to tell to select a recoding time greater than 0
    else
        %NO ERROR in order of steps

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Starting Recoding')
        %informs the user recoding is started

        %pause allows person to prepare after clicking the button
        set(Info, 'String', 'RECORDING...')

        recordblocking(SNDREC, record_dur);
        %Recording is done during this step

        set(Info, 'String', 'Recoding Ended')
        %informs the user the recoding has ended


        set(Info, 'String', 'proccesing...')
        snd.data = getaudiodata(SNDREC);
        signal_o = resample(snd.data, 2, 11,100);




        D = 10;
        N = 18;
        [signal_syn, Fs] = chvoc_over(signal_o, D, N, Fs, 'OR');
        [signal_syn_MO, Fs] = chvoc_over(signal_o, D, N, Fs, 'MO');
        [signal_syn_FE, Fs] = chvoc_over(signal_o, D, N, Fs, 'FE');
        [signal_syn_MA, Fs] = chvoc_over(signal_o, D, N, Fs, 'MA');
        [signal_syn_WH, Fs] = chvoc_over(signal_o, D, N, Fs, 'WH');
        [signal_syn_PI, Fs] = chvoc_over(signal_o, D, N, Fs, 'PI');

        %Time plot
        subplot('Position',TL)
        plot((0:length(signal_o)-1)/8000, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
        hold on

        plot((0:length(signal_syn)-1)/8000, signal_syn, 'Color', [0.302
0.745 0.933], 'Linewidth', 2)
        hold off
```

```
        legend('Recorded', 'Synthesized in Channel Vocoder',...
            'Location',[.05,.03,.4,.1]);
        set(gca, 'FontSize', 10)
        xlabel('Time (s)', 'FontSize', 15)
        ylabel('Normalized Amplitude', 'FontSize', 15)
        str = sprintf('Time Domain of Recorded Utterance:\nNormalized
Amplitude v. Time');
        title(str,'FontSize', 15)



        rec=1;
        %Remembers there is data to playback
        set(Info, 'String', 'Done')
        pause(0.5)
        set(Info, 'String', 'Try Playback')

        %Frequency plot
        subplot('Position',BL)

        [S_syn,F_syn,T_syn] = spectrogram(signal_syn,2^10,2^9,[],Fs);
        imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
        colorbar

        axis xy
        ylim([0 Fs/2])
        set(gca, 'FontSize', 10)

        xlabel('Time (s)', 'FontSize', 15)

        ylabel('Frequency (Hz)', 'FontSize', 15)

        title('Spectrogram for Synthesized "Original" ','FontSize', 15)




    end
end

%% Playback subfunction
%% PLAYBACK NORMAL SUBFUNCTION
%Subfuction utilizes progonal object
%There is no major issue with playback here
function Playback(hObject,eventdata)
    if rec~=1
        %ERROR CHECKING

        set(Info, 'BackgroundColor','red')
        %Background changed to red to emphasize error
        set(Info, 'String', 'Data must be recorded before playback')
        %if else used to tell user to record first
    else
        %NO ERROR
```

```
        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')
        soundsc(signal_syn)
        set(Info, 'String', 'Playback is done')

        %plots

        %time plot
        subplot('Position',TR)
        plot((0:length(signal_o)-1)/8000, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
        hold on


        plot((0:length(signal_syn)-1)/8000, signal_syn, 'Color', [0.302
0.745 0.933], 'Linewidth', 2)
        legend('Recorded', 'Synthesized in Channel Vocoder',...
            'Location',[.55,.03,.4,.1]);

        set(gca, 'FontSize', 10)
        xlabel('Time (s)', 'FontSize', 15)
        ylabel('Normalized Amplitude', 'FontSize', 15)
        str = sprintf('Time Domain of Recorded Utterance:\nNormalized
Amplitude v. Time');
        title(str,'FontSize', 17)
        hold off
        %Frequency Plot
        subplot('Position',BR)

        [S_syn,F_syn,T_syn] = spectrogram(signal_syn,2^10,2^9,[],Fs);
        imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
        colorbar

        axis xy

        xlabel('Time (s)')

        ylabel('Frequency (Hz)')

        title('Spectrogram for "Original" Utterrance Synthesized in Channel
Vocoder','FontSize', 15)

        ylim([0 Fs/2])
    end
end

%% PLAYBACK Orginal SUBFUNCTION
%Subfuction utilizes progonal object
%There is no major issue with playback here
function PlaybackO(hObject,eventdata)
    if rec~=1
        %ERROR CHECKING
```

```
        set(Info, 'BackgroundColor','red')
        %Background changed to red to emphasize error
        set(Info, 'String', 'Data must be recorded before playback')
        %if else used to tell user to record first
    else
        %NO ERROR

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')
        soundsc(signal_o)
        set(Info, 'String', 'Playback is done')

        %plots

        %time plot
        subplot('Position',TR)
        plot((0:length(signal_o)-1)/8000, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
        hold on

        plot((0:length(signal_syn)-1)/8000, signal_syn, 'Color', [0.302
0.745 0.933], 'Linewidth', 2)
        legend('Recorded', 'Synthesized in Channel Vocoder',...
            'Location',[.55,.03,.4,.1]);

        set(gca, 'FontSize', 10)
        xlabel('Time (s)', 'FontSize', 15)
        ylabel('Normalized Amplitude', 'FontSize', 15)
        str = sprintf('Time Domain of Recorded Utterance:\nNormalized
Amplitude v. Time');
        title(str,'FontSize', 17)

        hold off
        %Frequency Plot
        subplot('Position',BR)

        [S_syn,F_syn,T_syn] = spectrogram(signal_syn,2^10,2^9,[],Fs);
        imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
        colorbar
        axis xy
        set(gca, 'FontSize', 10)

        xlabel('Time (s)')

        ylabel('Frequency (Hz)')

        title('Spectrogram for Original Utterance Synthesized in Channel
Vocoder','FontSize', 15)

        ylim([0 Fs/2])
    end
end
```

```
%% PLAYBACK MONOTONE SUBFUNCTION
function PlaybackMono(hObject,eventdata)
    if rec~=1
        %ERROR CHECKING

        set(Info, 'BackgroundColor','red')
        %Background changed to red to emphasize error
        set(Info, 'String', 'Data must be recorded before playback')
        %if else used to tell user to record first
    else
        %NO ERROR

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')
        soundsc(signal_syn_MO)
        set(Info, 'String', 'Playback is done')

        %plots

        %time plot
        subplot('Position',TR)
        plot((0:length(signal_o)-1)/8000, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
        hold on

        plot((0:length(signal_syn_MO)-1)/8000, signal_syn_MO, 'Color',
[0.302 0.745 0.933], 'Linewidth', 2)
        legend('Recorded', 'Synthesized in Channel Vocoder',...
            'Location',[.55,.03,.4,.1]);

        set(gca, 'FontSize', 10)
        xlabel('Time (s)', 'FontSize', 15)
        ylabel('Normalized Amplitude', 'FontSize', 15)
        str = sprintf('Time Domain of Recorded Utterance:\nNormalized
Amplitude v. Time');
        title(str,'FontSize', 15)

        hold off
        %Frequency Plot
        subplot('Position',BR)

        [S_syn,F_syn,T_syn] = spectrogram(signal_syn_MO,2^10,2^9,[],Fs);
        imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
        colorbar

        axis xy
        set(gca, 'FontSize', 10)

        xlabel('Time (s)')

        ylabel('Frequency (Hz)')
```

```
        title('Spectrogram for "Monotone" Utterance Synthesized in Channel
Vocoder','FontSize', 15)

        ylim([0 Fs/2])
    end


end

function PlaybackWhisp(hObject,eventdata)
    if rec~=1
        % ERROR CHECKING

        set(Info, 'BackgroundColor','red')
        %Background changed to red to emphasize error
        set(Info, 'String', 'Data must be recorded before playback')
        %if else used to tell user to record first
    else
        %NO ERROR

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')
        sound(signal_syn_WH*0.6)
        set(Info, 'String', 'Playback is done')

        %plots

        %time plot
        subplot('Position',TR)
        plot((0:length(signal_o)-1)/8000, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
        hold on

        plot((0:length(signal_syn_WH)-1)/8000, signal_syn_WH, 'Color',
[0.302 0.745 0.933], 'Linewidth', 2)

        legend('Recorded', 'Synthesized in Channel Vocoder',...
            'Location',[.55,.03,.4,.1]);

        set(gca, 'FontSize', 10)
        xlabel('Time (s)', 'FontSize', 15)
        ylabel('Normalized Amplitude', 'FontSize', 15)
        str = sprintf('Time Domain of Recorded Utterance:\nNormalized
Amplitude v. Time');
        title(str,'FontSize', 15)

        hold off
        %Frequency Plot
        subplot('Position',BR)
```

```
        [S_syn,F_syn,T_syn] = spectrogram(signal_syn_WH,2^10,2^9,[],Fs);
        imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
        colorbar

        axis xy

        xlabel('Time (s)')

        ylabel('Frequency (Hz)')

        title('Spectrogram for "Whisper" Utterance Synthesized in Channel
Vocoder','FontSize', 15)

        ylim([0 Fs/2])
    end
end

function PlaybackMale(hObject,eventdata)
    if rec~=1
        % ERROR CHECKING

        set(Info, 'BackgroundColor','red')
        %Background changed to red to emphasize error
        set(Info, 'String', 'Data must be recorded before playback')
        %if else used to tell user to record first
    else
        %NO ERROR

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')
        soundsc(signal_syn_MA)
        set(Info, 'String', 'Playback is done')

        %plots

        %time plot
        subplot('Position',TR)
        plot((0:length(signal_o)-1)/8000, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
        hold on

        plot((0:length(signal_syn_MA)-1)/8000, signal_syn_MA, 'Color',
[0.302 0.745 0.933], 'Linewidth', 2)
        legend('Recorded', 'Synthesized in Channel Vocoder',...
            'Location',[.55,.03,.4,.1]);

        set(gca, 'FontSize', 10)
        xlabel('Time (s)', 'FontSize', 15)
```

```
        ylabel('Normalized Amplitude', 'FontSize', 15)
        str = sprintf('Time Domain of Recorded Utterance:\nNormalized
Amplitude v. Time');
        title(str,'FontSize', 15)

        hold off
        %Frequency Plot
        subplot('Position',BR)

        [S_syn,F_syn,T_syn] = spectrogram(signal_syn_MA,2^10,2^9,[],Fs);
        imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
        colorbar

        axis xy

        xlabel('Time (s)')

        ylabel('Frequency (Hz)')

        title('Spectrogram for "Male" Utterance Synthesized in Channel
Vocoder','FontSize', 15)

        ylim([0 Fs/2])
    end
end


function PlaybackFemale(hObject,eventdata)
    if rec~=1
        % ERROR CHECKING

        set(Info, 'BackgroundColor','red')
        %Background changed to red to emphasize error
        set(Info, 'String', 'Data must be recorded before playback')
        %if else used to tell user to record first
    else
        %NO ERROR

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')
        soundsc(signal_syn_FE)
        set(Info, 'String', 'Playback is done')

        subplot('Position',TR)
        plot((0:length(signal_o)-1)/8000, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
        hold on
```

```
        plot((0:length(signal_syn_FE)-1)/8000, signal_syn_FE, 'Color',
[0.302 0.745 0.933], 'Linewidth', 2)
        legend('Recorded', 'Synthesized in Channel Vocoder',...
            'Location',[.55,.03,.4,.1]);

        set(gca, 'FontSize', 10)
        xlabel('Time (s)', 'FontSize', 15)
        ylabel('Normalized Amplitude', 'FontSize', 15)
        str = sprintf('Time Domain of Recorded Utterance:\nNormalized
Amplitude v. Time');
        title(str,'FontSize', 15)

        hold off
        %Frequency Plot
        subplot('Position',BR)

        [S_syn,F_syn,T_syn] = spectrogram(signal_syn_FE,2^10,2^9,[],Fs);
        imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
        colorbar

        axis xy
        set(gca, 'FontSize', 10)

        xlabel('Time (s)')

        ylabel('Frequency (Hz)')

        title('Spectrogram for "Female" Utterance Synthesized in Channel
Vocoder','FontSize', 15)

        ylim([0 Fs/2])
    end
end

function PlaybackPitchless(hObject,eventdata)
    if rec~=1
        % ERROR CHECKING

        set(Info, 'BackgroundColor','red')
        %Background changed to red to emphasize error
        set(Info, 'String', 'Data must be recorded before playback')
        %if else used to tell user to record first
    else
        %NO ERROR

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')

        set(Info, 'BackgroundColor','white')
        %Background changed back
        set(Info, 'String', 'Recoding is being played back')
        sound(signal_syn_PI*0.6)
        set(Info, 'String', 'Playback is done')
```

```
        %plots

        %time plot
        subplot('Position',TR)
        plot((0:length(signal_o)-1)/8000, signal_o/norm(signal_o,inf), 'b-',
'Linewidth', 2)
        hold on

        plot((0:length(signal_syn_PI)-1)/8000, signal_syn_PI, 'Color',
[0.302 0.745 0.933], 'Linewidth', 2)

        legend('Recorded', 'Synthesized in Channel Vocoder',...
            'Location',[.55,.03,.4,.1]);

        set(gca, 'FontSize', 10)
        xlabel('Time (s)', 'FontSize', 15)
        ylabel('Normalized Amplitude', 'FontSize', 15)
        str = sprintf('Time Domain of Recorded Utterance:\nNormalized
Amplitude v. Time');
        title(str,'FontSize', 15)

        hold off
        %Frequency Plot
        subplot('Position',BR)

        [S_syn,F_syn,T_syn] = spectrogram(signal_syn_PI,2^10,2^9,[],Fs);
        imagesc(T_syn,F_syn,20*log10(abs(S_syn)),[-126 34])
        colorbar

        axis xy

        xlabel('Time (s)')

        ylabel('Frequency (Hz)')

        title('Spectrogram for "Pitchless" Utterance Synthesized in Channel
Vocoder','FontSize', 15)

        ylim([0 Fs/2])
    end
end
end
```

FILT_BANK.m

```
% BE491 Group Project Filter Bank Generator
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function bank = filt_bank(N, L, varargin)
% FILT_BANK Filter bank generator
%   BANK = FILT_BANK(N,L,Fs,B) generates a bank of filters where
```

```
%   N is the number of filter bands
%   L is the length of each FIR filter
%   Fs is the sampling frequency in Hz, default 8kHz
%   B is the width of each band in Hz
%   BANK is an LxN matrix, where each of the N columns of BANK contains an
L-point FIR
%   filter.
%
%   BANK = FILT_BANK(N,L,Fs) automatically selects the bandwidth B so that
the N
%   filters span the spectrum from 0 Hz to 3600 Hz.
%
%   BANK = FILT_BANK(N,L) sets Fs to 8000 Hz, and automatically selects the
%   bandwidth B so that the N filters span the spectrum from 0 Hz to 3600
Hz.

%% Process input
if nargin < 4
    B = 3600/N;  % set default width of each band in Hz
else
    B = varargin{2};
end
if nargin < 3
    Fs = 8000;   % set default sampling frequency in Hz
else
    Fs = varargin{1};
end
start = B/2;      % First center freq. in Hz
FL = B;           % Bandwidth in Hz

% Preallocate output for speed
bank = zeros(L,N);

% Determine Nyquist frequency
Fny = Fs/2;

%% Prototype LPF
    % Cutoff frequency chosen to obtain a bandwidth of B
    % Kaiser window with beta = 3
lpf = fir1(L-1,B/Fny,kaiser(L,3));
lpf = lpf(:); % Make LPF into a column vector

%% Create bandpass filters
    % By shifting the lowpass filter into a series of bandpass filters
% (i) Create a discrete-time column vector n for argument to cosines
        % with length of the lowpass filter (L)
        % with spacing between the samples 1/Fs
n = ([0:L-1]/Fs)';

% (ii) Design filters for the remaining bands by looping
for i = 1:N
 % Compute desired center frequency from i, B, start, and Fs
 cf = i*B-start;
 % Shift lowpass prototype to center frequency
    if i==1
```

```
        bank(:,i) = lpf;
    else
        bank(:,i) = lpf .* cos(2*pi*cf*n)*2;
    % Default calculations in MATLAB for cos are done in radians
    end
end

%% Extra code for plotting the frequency response of the filter bank
%{
F = 1:Fny;

figure
Band = abs(freqz(bank(:,1),1,F,Fs));
plot(F,Band,'r','Linewidth', 2)
xlabel('Frequency (Hz)', 'FontSize', 40)
ylabel('Amplitude', 'FontSize', 40)
title('Fundamental Frequency Response of the Filter Bank ','FontSize', 40)
axis([-100 4100 -0.1 1.1])
set(gca, 'FontSize', 25)

figure
hold on
plot(F,Band,'r--','Linewidth', 2)
for z=2:N
    Band = abs(freqz(bank(:,z),1,F,Fs));
    plot(F,Band,'b-','Linewidth', 2);
end
hold
xlabel('Frequency (Hz)', 'FontSize', 40)
ylabel('Amplitude', 'FontSize', 40)
str = sprintf('Frequency Response of the Filter Bank\nfor an 18-band 65th-
order LPF');
title(str,'FontSize', 40)
legend('Fundamental Response','Response of Banked LPFs')
axis([-100 4100 -0.1 1.1])
set(gca, 'FontSize', 25)
%}
```

ISTFT.m

```
% BE491 Group
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function x = istft(d, ftsize, w, h)
% X = istft(D, F, W, H)                   Inverse short-time Fourier
transform.
%   Performs overlap-add resynthesis from the short-time Fourier transform
%   data in D.  Each column of D is taken as the result of an F-point
%   fft; each successive frame was offset by H points (default
%   W/2, or F/2 if W==0). Data is hann-windowed at W pts, or
%       W = 0 gives a rectangular window (default);
%       W as a vector uses that as window.
%       This version scales the output so the loop gain is 1.0 for
%       either hann-win an-syn with 25% overlap, or hann-win on
```

```
%       analysis and rect-win (W=0) on synthesis with 50% overlap.

if nargin < 2; ftsize = 2*(size(d,1)-1); end
if nargin < 3; w = 0; end
if nargin < 4; h = 0; end  % will become winlen/2 later

s = size(d);
if s(1) ~= (ftsize/2)+1
  error('number of rows should be fftsize/2+1')
end
cols = s(2);

if length(w) == 1
  if w == 0
    % special case: rectangular window
    win = ones(1,ftsize);
  else
    if rem(w, 2) == 0   % force window to be odd-len
      w = w + 1;
    end
    halflen = (w-1)/2;
    halff = ftsize/2;
    halfwin = 0.5 * ( 1 + cos( pi * (0:halflen)/halflen));
    win = zeros(1, ftsize);
    acthalflen = min(halff, halflen);
    win((halff+1):(halff+acthalflen)) = halfwin(1:acthalflen);
    win((halff+1):-1:(halff-acthalflen+2)) = halfwin(1:acthalflen);
    % 2009-01-06: Make stft-istft loop be identity for 25% hop
    win = 2/3*win;
  end
else
  win = w;
end

w = length(win);
% now can set default hop
if h == 0
  h = floor(w/2);
end

xlen = ftsize + (cols-1)*h;
x = zeros(1,xlen);

for b = 0:h:(h*(cols-1))
  ft = d(:,1+b/h)';
  ft = [ft, conj(ft([((ftsize/2)):-1:2]))];
  px = real(ifft(ft));
  x((b+1):(b+ftsize)) = x((b+1):(b+ftsize))+px.*win;
end;
```

PEAK.m

```
% BE491 Group Project Find Fundamental Peak in Autocorrelation
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
```

```
% Lab Section B3

function [peakval,peakindex] = peak(x)
% PEAK Detects autocorrelation fundamental peak
%   [PEAKVAL, PEAKINDEX] = PEAK(X) locates the value and index of the
largest
%   peak in the vector X other than Rx[0].  X must be an autocorrelation
%   function with maximum value Rx[0] as its first element.

%% Check input arguments
if nargin == 0
    error('You must enter an autocorrelation function as input.');
end
if (size(x,1) > 1) && (size(x,2) > 1)
    error ('The input signal must be a vector')
end

%% Processing
x = x(:);
if x(1) ~= max(x)
    error('Input must be an autocorrelation function with Rx[0] as the first
element')
end;

positive_slope = find(diff(x)>0);
start_index = positive_slope(1);

[peakval, peakindex] = max(x(start_index:end)); % find max value
peakindex = peakindex + start_index - 1;        % adjust for start index
offset
```

PITCH_DETECT.m

```
% BE491 Group Project Autocorrelation Algorithm
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

% Algorithm to determine the fundamental frequency of the voice
function pitch = pitch_detect(x, varargin)
% x         vector containing frame of speech data sampled at Fs
% varargin
%      clip_thresh:a scaling factor of the max/min values, 0.75 (75%)
default
%      unvoiced_thresh: the minimum allowable relative pitch peak amplitude,
%         below which the segment is considered unvoiced, 0.25 (25%) default
%      Fs: sampling frequency, default 8 kHz
% pitch: a scalar containing pitch of frame in Hz or 0 if unvoiced

%% Check input arguments
if nargin == 0
    error('You must enter a filtered speech signal.');
elseif nargin == 1
    clip_thresh = 0.75;
    unvoiced_thresh = 0.25;
```

```
    Fs = 8000; %Hz
elseif nargin == 2
    clip_thresh = varargin{1};
    unvoiced_thresh = 0.25;
    Fs = 8000; %Hz
elseif nargin == 3
    clip_thresh = varargin{1};
    unvoiced_thresh = varargin{2};
    Fs = 8000; %Hz
elseif nargin == 4
    clip_thresh = varargin{1};
    unvoiced_thresh = varargin{2};
    Fs = varargin{3}; %Hz
else
    error('Improper input format.');
end

% Remove DC offset
x = x - mean(x);

% Find min and max samples, account for thresholds, and center clip using
cclip function
x = cclip(x, min(x)*clip_thresh, max(x)*clip_thresh);
%{
Center clips the signal x setting the lower and upper clipping thresholds
from the MINVAL and MAXVAL respectively. Signal components between MINVAL
and MAXVAL are 'center clipped', while components below MINVAL are shifted
up and compoents above MAXVAL are shifted down. MINVAL must be negative and
MAXVAL must be positive. Each elements of X is processed as follows:
If X(i) > MAXVAL, then Y(i) = X(i) ? MAXVAL;
If MINVAL < X(i) < MAXVAL, then Y(i) = 0;
If X(i) < MINVAL, then Y(i) = X(i) - MINVAL;
Motivation:
In order to use the autocorrelation function for automatic pitch detection,
it is helpful to suppress the peaks due to the vocal tract transfer
function. This center clipping will accomplish the suppression.
%}

% Compute the autocorrelation of the frame
Rx = xcorr(x,'coeff');
% Calculates the autocorrelation and also normalizes to 1
% Note that the zeroth lag of the correlation, Rx[0], is in the middle of
the output sequence.

% Find the maximum peak following Rx[0] by calling peak function
% To find the index of the maximum value; should be at Rx[0]
max_index = find(Rx == max(Rx));
% Extract the positive part of the correlation (e.g. on x axis)
Rx_pos = Rx(max_index: length(Rx));
% Find the maximum peak following Rx[0]
[peakVAL, index] = peak(Rx_pos);
%PEAK Detects autocorrelation fundamental peak
%   [PEAKVAL, PEAKINDEX] = PEAK(X) locates the value and index of the
largest
%   peak in the vector X other than Rx[0].  X must be an autocorrelation
%   function with maximum value Rx[0] as its first element.
```

```
peaktime = index/Fs; % converting from index of sample to seconds

% Determine if the segment is unvoiced based on the 'voicing strength' (the
% ratio of the autocorrelation function at the peak pitch lag to the
% autocorrelation function at lag = 0)...
% If voicing strength is less than unvoiced_thresh, call it unvoiced and set
% pitch = 0, otherwise compute the pitch.
if peakVAL < unvoiced_thresh % segment is unvoiced
    pitch = 0;
else % segment is voiced
    pitch = 1/peaktime;
end
end
```

PULSE_TRAIN.m

```
% BE491 Group Pulse Train Generator
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function [p, next_delay] = pulse_train(period, len, init_delay)
%PULSE_TRAIN Generate a discrete impulse train
%   [PULSE, NEXT_DELAY] = PULSE_TRAIN(PERIOD,LENGTH,INIT_DELAY) generates a
%   discrete impulse pulse train. The period in samples is specified by
PERIOD,
%   the length of the pulse train is specified by LENGTH, and the sample
delay
%   from the first sample to the first impulse (i.e. the number of leading
%   zeros) is set by INIT_DELAY.
%
%   The output pulse train is returned in vector PULSE, and the delay to the
%   first pulse in the next frame is returned in NEXT_DELAY. To create two
%   consecutive pulse trains with aligned pulse periods, the second pulse
train
%   should be created by specifying an INIT_DELAY corresponding to the
%   NEXT_DELAY of the previous pulse train.
%
%   [PULSE, NEXT_DELAY] = PULSE_TRAIN(PERIOD,LENGTH) uses a default
INIT_DELAY
%   of zero.

%% Check input argument
if nargin < 3
    init_delay = 0;
end

% Create impulse train
p = zeros(len,1);     % Initialize impulse train output
pulse_times = init_delay+1:period:len;
if ~isempty(pulse_times),
    p(pulse_times) = 1;
    next_delay = max(pulse_times) + period - len - 1; % find delay to next
pulse
else
    next_delay = init_delay - len;  % accounts for init delays > length
```

```
end
```

STFT.m

```
% BE491 Group
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function D = stft(x, f, w, h, sr)
% D = stft(X, F, W, H, SR)                       Short-time Fourier
transform.
%   Returns some frames of short-term Fourier transform of x.  Each
%   column of the result is one F-point fft (default 256); each
%   successive frame is offset by H points (W/2) until X is exhausted.
%       Data is hann-windowed at W pts (F), or rectangular if W=0, or
%       with W if it is a vector.
%       Without output arguments, will plot like sgram (SR will get
%       axes right, defaults to 8000).

if nargin < 2;  f = 256; end
if nargin < 3;  w = f; end
if nargin < 4;  h = 0; end
if nargin < 5;  sr = 8000; end

% expect x as a row
if size(x,1) > 1
  x = x';
end

s = length(x);

if length(w) == 1
  if w == 0
    % special case: rectangular window
    win = ones(1,f);
  else
    if rem(w, 2) == 0   % force window to be odd-len
      w = w + 1;
    end
    halflen = (w-1)/2;
    halff = f/2;   % midpoint of win
    halfwin = 0.5 * ( 1 + cos( pi * (0:halflen)/halflen));
    win = zeros(1, f);
    acthalflen = min(halff, halflen);
    win((halff+1):(halff+acthalflen)) = halfwin(1:acthalflen);
    win((halff+1):-1:(halff-acthalflen+2)) = halfwin(1:acthalflen);
  end
else
  win = w;
end

w = length(win);
% now can set default hop
if h == 0
```

```
  h = floor(w/2);
end

c = 1;

% pre-allocate output array
d = zeros((1+f/2),1+fix((s-f)/h));

for b = 0:h:(s-f)
  u = win.*x((b+1):(b+f));
  t = fft(u);
  d(:,c) = t(1:(1+f/2))';
  c = c+1;
end;

% If no output arguments, plot a spectrogram
if nargout == 0
  tt = [0:size(d,2)]*h/sr;
  ff = [0:size(d,1)]*sr/f;
  imagesc(tt,ff,20*log10(abs(d)));
  axis('xy');
  xlabel('time / sec');
  ylabel('freq / Hz')
  % leave output variable D undefined
else
  % Otherwise, no plot, but return STFT
  D = d;
end
```

SW_SOURCE.m

```
% BE491 Group Source Signal Generator
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3



function [out,next_delay] = sw_source(pitch,Fs,frlen,init_delay)
%SW_SOURCE  Creates source signal from pitch/voicing data
%   [SOURCE,NEXT_DELAY] = SW_SOURCE(PITCH,FS,FR_LEN,INIT_DELAY) generates
speech
%   source signal SOURCE from the vector of pitch values PITCH (in Hz). FS
is
%   the sampling frequency in Hz, FR_LEN is the frame length in samples, and
%   INIT_DELAY is the initial delay to the first pulse in samples. If
%   INIT_DELAY is not specified, the default value is zero.
%
%   For nonzero values of PITCH, SOURCE contains a pulse train at the pitch
%   rate; when PITCH is zero, SOURCE contains white noise.

% Input argument checking
if nargin  < 3
    error('SW_SOURCE: Improper function call.');
end;
if nargin < 4
```

```
    init_delay = 0;
end;

nframes = length(pitch);
out = zeros(frlen*nframes,1);

next_delay = init_delay;
for i = 1:length(pitch)
    if pitch(i) > 0
        pitchPeriod = floor(Fs/pitch(i));
        [source,next_delay] = pulse_train(pitchPeriod,frlen,next_delay);
    else
        source = randn(frlen,1);
    end;
    if norm(source) ~= 0
        source = source/norm(source);  % ensure unit energy per frame
        % optional: norm(source,inf): normalizes amplitude versus energy
    end;
    out((i-1)*frlen+1:i*frlen) = source;
end;
```

VOC_P.m

```
% BE491 Group
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function y = voc_p(x, r, n)
% y = voc_p(x, r, n)  Time-scale a signal to r times faster with phase
vocoder
%      x is an input sound. n is the FFT size, defaults to 1024.
%      Calculate the 25%-overlapped STFT, squeeze it by a factor of r,
%      inverse spegram.

if nargin < 3
  n = 1024;
end

% With hann windowing on both input and output,
% we need 25% window overlap for smooth reconstruction
hop = n/4;
% Effect of hanns at both ends is a cumulated cos^2 window (for
% r = 1 anyway); need to scale magnitudes by 2/3 for
% identity input/output
scf = 1.0;

% Calculate the basic STFT, magnitude scaled
X = scf * stft(x', n, n, hop);

% Calculate the new timebase samples
[rows, cols] = size(X);
t = 0:r:(cols-2);
% Have to stay two cols off end because (a) counting from zero, and
% (b) need col n AND col n+1 to interpolate
```

```
% Generate the new spectrogram
X2 = voc_p_interp(X, t, hop);
% Invert to a waveform
y = istft(X2, n, n, hop)';
```

VOC_P_INTERP.m

```
% BE491 Group
% Echo: A Voice Recognition and Playback System
% Davy Huang, Blake Oberfeld, Arjun Patel, Allison Ramsey, and Kate Ryan
% Lab Section B3

function c = voc_p_interp(b, t, hop)
% c = voc_p_interp(b, t, hop)   Interpolate an STFT array according to the
'phase vocoder'
%     b is an STFT array, of the form generated by 'specgram'.
%     t is a vector of (real) time-samples, which specifies a path through
%     the time-base defined by the columns of b.  For each value of t,
%     the spectral magnitudes in the columns of b are interpolated, and
%     the phase difference between the successive columns of b is
%     calculated; a new column is created in the output array c that
%     preserves this per-step phase advance in each bin.
%     hop is the STFT hop size, defaults to N/2, where N is the FFT size
%     and b has N/2+1 rows.  hop is needed to calculate the 'null' phase
%     advance expected in each bin.
%     Note: t is defined relative to a zero origin, so 0.1 is 90% of
%     the first column of b, plus 10% of the second.


if nargin < 3
  hop = 0;
end

[rows,cols] = size(b);

N = 2*(rows-1);

if hop == 0
  % default value
  hop = N/2;
end

% Empty output array
c = zeros(rows, length(t));

% Expected phase advance in each bin
dphi = zeros(1,N/2+1);
dphi(2:(1 + N/2)) = (2*pi*hop)./(N./(1:(N/2)));
% Phase accumulator
% Preset to phase of first frame for perfect reconstruction
% in case of 1:1 time scaling
ph = angle(b(:,1));
b = [b,zeros(rows,1)];

ocol = 1;
```

```
for tt = t
  % Grab the two columns of b
  bcols = b(:,floor(tt)+[1 2]);
  tf = tt - floor(tt);

  bmag = (1-tf)*abs(bcols(:,1)) + tf*(abs(bcols(:,2)));

  % calculate phase advance
  dp = angle(bcols(:,2)) - angle(bcols(:,1)) - dphi';
  % Reduce to -pi:pi range
  dp = dp - 2 * pi * round(dp/(2*pi));
  % Save the column
  c(:,ocol) = bmag .* exp(j*ph);

  % Cumulate phase, ready for next frame
  ph = ph + dphi' + dp;
  ocol = ocol+1;
end
```