Music Resampling and Synthesis

TuanKhai Nguyen

tran.tuan.khai.nguyen@vanderbilt.edu

EECE 5356 – Digital Signal Processing

A) Documentation

1) Data is broken into frames and each frame is FFT transformed using spectrogram()

```
% --------------------- 1) Break song into frames ---------------
x=x(:,1);
Twin = 0.050;
Nwin = round(Twin*Fs);
Noverlap = round( 0.75*Nwin);
NFFT = Fs;
S = spectrogram(x, hamming(Nwin), Noverlap, NFFT, Fs, 'twosided');
% break into frames - each col of S is fourier transform of each frame
% -------------------------------------------------------------
```

2) Use autocorrelation in the frequency domain to identify the peaks. Then obtain the window-normalized autocorrelation in the time-domain. Save in vector *R* for autocorrelation.

```
R = [];
frameenergy = []; % frameenergy = power-spectrum

% Compute the autocorrelations for each frame
for i=1:scols
    % USE FREQUENCY DOMAIN TO COMPUTE autocorrelation
    frameenergy = [frameenergy ; sum( abs( S(:,i) ).^2  )]; % freq domain
measure of autocorr = power spectrum
    %normally use Rxx = 1/N * xcorr(x) OR ---- Sxx = 1/N * sum( abs( S(:,i)
).^2
    stmp = ifft( abs( S(:,i) ).^2 )/Nwin; % to obtain auto correlation in
time domain
    stmp = stmp(1:Nwin); %
    stmp = real(stmp); % get real part of S only
    R = [R stmp];
end
```

3) Identify distance between 2 big peaks.
If the tallest peak is not big enough to correspond to an actual note being played, then indicate that no note was played in that frame. In music this is a rest.

```
periods = [];
energythreshold = 5000;
% Find periods and silent frames
for i=1:scols
    [pkvals, pklocs] = findpeaks( R(:,i) );  % loop through autocorrelation
vector R. pklocs = index of peaks
    [pkvsort , III] = sort(pkvals, 'descend'); % III - big peaks index of
pkvals
    ptmp = pklocs(III(2)) - pklocs(III(1));  % distance between 2 big peaks
    if ptmp < 0
```

```
        ptmp = 0;
    end
    if frameenergy(i) < energythreshold % NO note
        ptmp = 0;
    end
    periods = [periods ; ptmp];
end
```

4) Compute the Fundamental frequencies.
   F/Fs = f = 1/T --> F = Fs/T = sampling freq / period = fundamental freq
   Frequencies are stored in *notefrequencies.*

```
notefrequencies = zeros( size( periods) );
notebeingplayed = frameenergy > energythreshold;  % 0/1 vector for masking
periods = periods .* notebeingplayed;
for i=1:scols
    if notebeingplayed(i) > 0  %=1
        notefrequencies(i) = Fs / periods(i);
        % F/Fs = f = 1/T --> F = Fs/T = sampling freq / period = fundamental
freq
    end
end

% Convert notefrequencies to the nearest true note frequency values
m = 0:24;
fm = 110*2.^(m/12);  % True note frequencies 110Hz = A-note
for i=1:scols
    if notebeingplayed(i) > 0
        notefrequencies(i) = mynearestnumber( notefrequencies(i) , fm);  %
note close to fm will be classified as fm (Hz)
    end
end
```

5) Do some furnishing

```
% keep integer values only
notefrequencies = round(notefrequencies);
% one-dimensional median filter to the input vector, to get rid of
rizzling/bubling/radical steppings
notefrequencies = medfilt1(notefrequencies,5); % a 5th-order
notefrequencies(574) = 0;
```

6) Compute frequency/note/duration map

```
%% ----------- freq_length_map -------------

% L=17 0's at the start
% How many samples?
% Nhop = Nwin - Noverlap = 551
% Nsamples = (L-1)*Nhop + Nwin --- thus first part of song is Nsamples 0's

% L=32 frames at 330Hz second part of song
% Nsamples = (L-1)*Nhop + Nwin
```

```
% loop through note frequencies
% Note ---- Notelength/samples
% 0           (L-1)*Nhop-Nwin (in a loop)
% 0           11021
% 330         19286
% 0            ....
% 244          ....

freq_length_map = [];
Nhop = Nwin - Noverlap;
current_freq = 0;
current_notelength = 1;
for freq_loc = 2:length(notefrequencies)
    temp_freq = notefrequencies(freq_loc);
    if temp_freq == notefrequencies(freq_loc - 1)
        current_freq = temp_freq;
        current_notelength = current_notelength + 1;
    else
        numsamples = (current_notelength-1) * Nhop + Nwin;
        if isempty(freq_length_map) % first time
            freq_length_map = [current_freq, current_notelength,
numsamples];
            current_notelength = 1;
        else
            freq_length_map = cat(1,freq_length_map, [current_freq,
current_notelength, numsamples]);
            current_notelength = 1;
        end
    end
end
```

7) Furnish any rizzling in sounds / jump in frequencies.

```
%% furnishing the rizzling
[occurences, fundamentalfreq] = groupcounts(notefrequencies)
occurences_and_fundamentalfreq = [occurences, fundamentalfreq]
occurences_and_fundamentalfreq =
sortrows(occurences_and_fundamentalfreq,'descend')
% Remove rows with zeros fequencies
occurences_and_fundamentalfreq(~occurences_and_fundamentalfreq(:,2),:) = []
occurences_and_fundamentalfreq = occurences_and_fundamentalfreq(1:3,:)
fundamentalfreq = occurences_and_fundamentalfreq(:,2)

for i = 2:length(freq_length_map(:,1))
    if freq_length_map(i,1) > 0 && sum(freq_length_map(i,1) ~=
fundamentalfreq) == 3
        if freq_length_map(i-1,1) == freq_length_map(i+1,1)
            freq_length_map(i,1) = freq_length_map(i+1,1);
        elseif freq_length_map(i-1,1) ~= freq_length_map(i+1,1)
            freq_length_map(i,1) = 0;
        end
    end
end
```

```
freq_length_map(11,1) = 294;
```

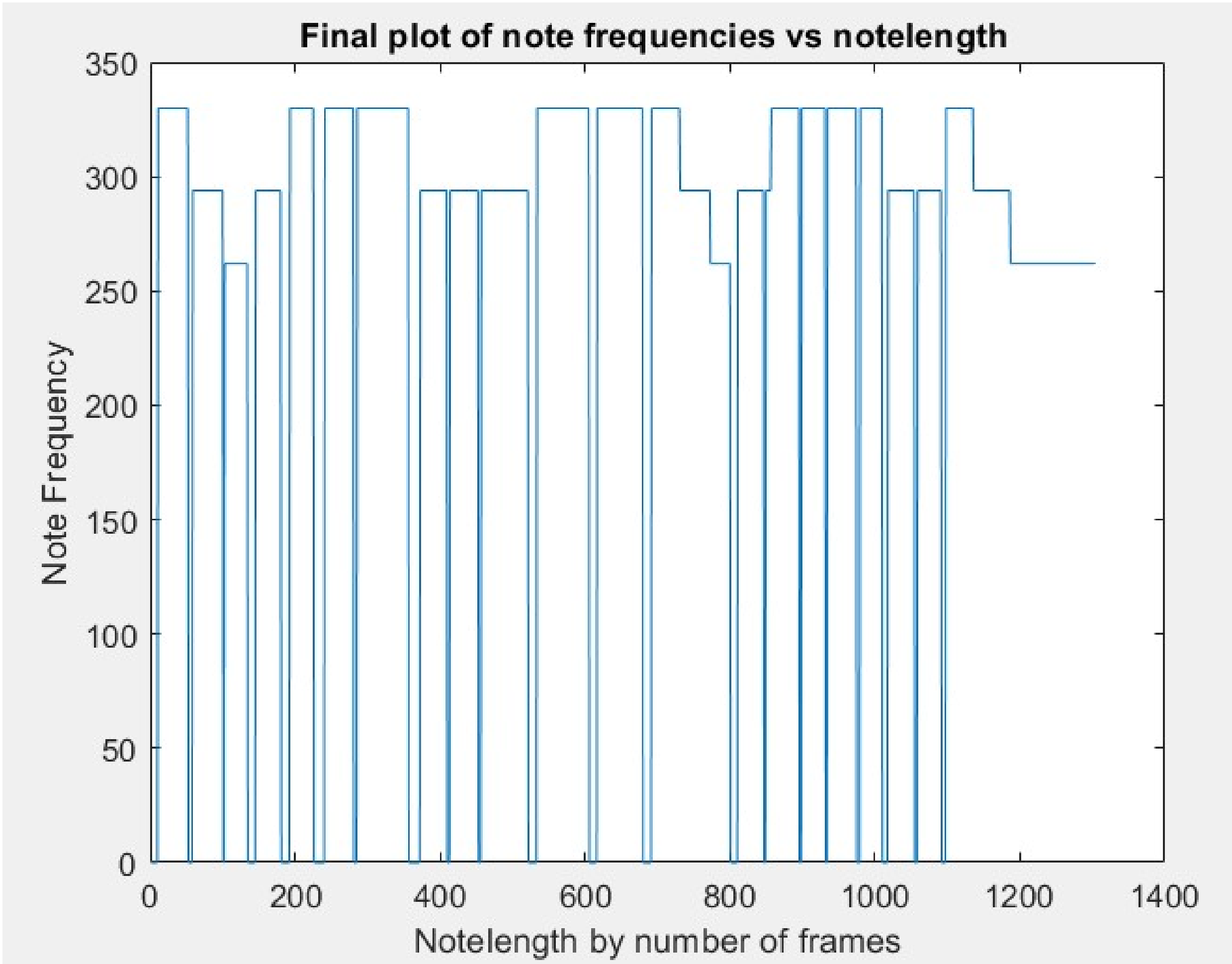
8) Resynthesize the sone using the trombone sound and *resample()* function. Write result to file.

```
%%
[xtrombone,Ft]=audioread('trombone44100.wav');

xtrombonefreq = 262;
% enote = resample(xtrombone, xtrombonefreq, 330);
% dnote = resample(xtrombone, xtrombonefreq, 294);
% cnote = xtrombone;
notes = freq_length_map(:,1);
notelength = freq_length_map(:,3);
%%
song = [];
for i=1:length(notes)
  tmp = notes(i);
  if tmp == 0
      tmpnote = zeros(notelength(i),1);
      song = [song; tmpnote];
  else
      tmpnote = resample(xtrombone, xtrombonefreq, notes(i));
      tmpnote = tmpnote(1:notelength(i));
      song = [song; tmpnote];
  end
end
%%
sound(song,Fs)
%%
% build new notefrequencies sequence after all processing
new_notefrequencies = [];
for i = 1:length(freq_length_map(:,1))
    new_notefrequencies = [new_notefrequencies;
repmat(freq_length_map(i,1),freq_length_map(i,2),1)];
end
figure(1), plot( new_notefrequencies )

filename = 'trombone_synthesized.wav';
audiowrite(filename,song,Fs);
```

9) Result



Another synthesis version for the violin sound is attached.

B) <u>**How to run:**</u>

1) For trombone: Run *trombone_synthesis_autocorrelationbased.m*
   Resulting synthesized sound will be played.

2) For violin: Run *violin_synthesis_autocorrelationbased.m.*
   Resulting synthesized sound will be played.