DSP Design
Short-Time Fourier Transform

Shannon L. Sabino

November 2, 2017 (Original)
February 14, 2018 (Edits)

Contents

Introduction

In this project, we continue the previous project's theme of pursuing real-time processing methods. As most signals' content varies greatly with time, we are motivated to view these time-changes and filter them accordingly. From this, we find our motivation for the Short-Time Fourier Transform (STFT). To take the STFT, we break our input, x, into sections of length N, called x_m, and make sure that our sections overlap such that we can later reconstruct our signal as if there had been no sectioning done at all. We call the time lag between our sections R. Additionally, we apply a window, $w(n)$, also of length N, to each section to soften the harsh cutoff on the edges, which would create oddities in our frequency processing.

Let's clarify notation before we move on. The input is broken into $M+1$ segments, x_m, where $m = 0, 1, ..., M$. Each x_m is defined over N samples, $x_m(n)$, where $n = 0, 1, ..., N-1$. In short, we have divided our input into $M+1$ length-N pieces. If we apply our length-N window, $w(n)$, to each piece, we can think of this as a $N \times (M+1)$ matrix, which can be more easily passed into matrix-optimized MATLAB functions. Finally, we can describe taking the N-point DFT of each section in a neat equation:

$$X_{k,m} = \sum_{n=0}^{N-1} x(mR+n)w(n)e^{-j\omega_k n} = \sum_{n=0}^{N-1} x_m(n)e^{-j\omega_k n}, \tag{1}$$

where $x_m(n) = x(mR+n)w(n)$. This describes the general STFT. From this, it logically follows that the inverse STFT, the ISTFT, makes use of the N-point inverse DFT, such that

$$x(mR+n)w(n) = x_m(n) = \frac{1}{N}\sum_{k=0}^{N-1} X_{k,m}e^{j\omega_k n} \tag{2}$$

would represent the ISTFT. This solution presents a problem though — we still have the window, $w(n)$, multiplied into the output, which is non-ideal. Thus our final step in taking the ISTFT is a reach-back to our previous project, using the overlap-add (OLA) method of reconstruction, where our final output is

$$y(n) = \sum_{m=-\infty}^{\infty} x_m(n - mR). \tag{3}$$

This is the final expression for the ISTFT used herein.

In practice, the gap R used in the STFT and the R used in the ISTFT can be different — the former is called the analysis hop, and the latter is the synthesis hop size. The effects of having $R_a \neq R_s$ will be described shortly.

Section I: Phase Vocoder

In this section, we create a phase vocoder. Phase vocoders preserve the magnitude of the input signal while altering the phase after taking its STFT and before taking its ISTFT. One application of this is to speed up or slow down the play time of a signal without modulating the audio into a higher or lower pitch. To accomplish this, we can resample each section of the signal that we took for our STFTs, and then place these sections back in

the "appropriate" location by changing the R for our ISTFT. A neat way to describe this is by the factor of speed up, r, where $r = \frac{Ra}{Rs}$. Therefore, given our analysis hop R_a and our desired speed change r, we can find a way to change our time-domain signal without changing our frequency-domain "content".

Part I: `phvoc.m`

To accomplish this, we write a phase vocoder function in MATLAB, **phvoc**, which takes in our entire original signal, x, the factor of speed up, r, the initial chunk displacement size, R_a, and the length of each section, N. The output, y, is the appropriately slowed down or sped up version of x.

We break the implementation of this function into three parts, the first of which is taking the STFT of x. We choose a hanning window, of the form,

$$w(n) = 0.5 - 0.5\cos(\frac{2\pi n}{N-1}). \tag{4}$$

We break x up with the function **buffer**, which politely breaks x into an $N \times (M+1)$ matrix, including any necessary zero-padding at the end to keep it even. We additionally multiply each row of this matrix by our window. Finally, we can pass this entire matrix into the built-in function **fft**, and take an N-point DFT of each row.

Next, we do the phase mapping on these sections. Each sample's phase is shifted with respect to the analysis hop size and scaled with respect to the synthesis hop size.

Finally, we take the ISTFT, using the synthesis hop size.

Test I: Speed Up

To test our function, we try both slowing down and speeding up a piece of audio. I chose a .wav file that was used a few projects ago because I have no other .wav files on my computer, and ran it through the **phvoc** function with $r = 1.4$, $R_a = 50$ and $N = 3000$. After normalizing the signal (since the speed up also pushed the volume up), the output is indeed a faster version of the input, but with the same pitch. This is saved to *phvocFast.wav*.

Test II: Slow Down

Next, I used $r = 0.7$ with the same R_a and N. The pitch is again unchanged although the signal is much longer. This audio is saved to *phvocSlow.wav*. Examining the magnitude and phase spectrums of each signal shows us that the magnitude is the same between the two signals, while the phase is appropriately stretched on the slower signal.

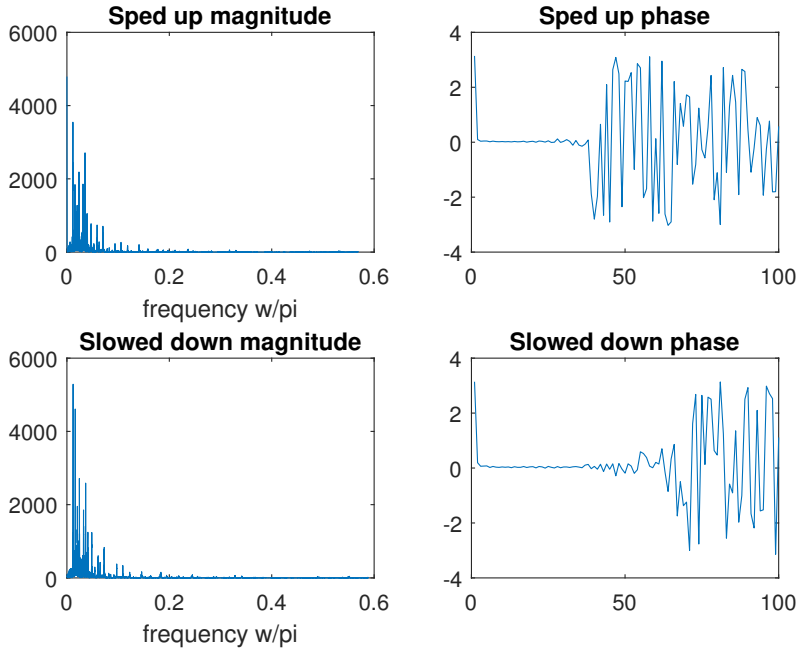


Figure 1: Comparison of slowed and sped up audio after the phase vocoder.

Part II: `pitchmod.m`

In this section, we intentionally modify the pitch of the audio of a flute without changing its speed, using a function we make called **pitchmod**. This function resamples the input by r, which changes the length of the signal, before passing it into **phvoc** with $r = \frac{1}{r}$, which has the overall result of the output being around the same length as the input, but with its pitch shifted up or down. Examine a section of the frequency of the original audio below.

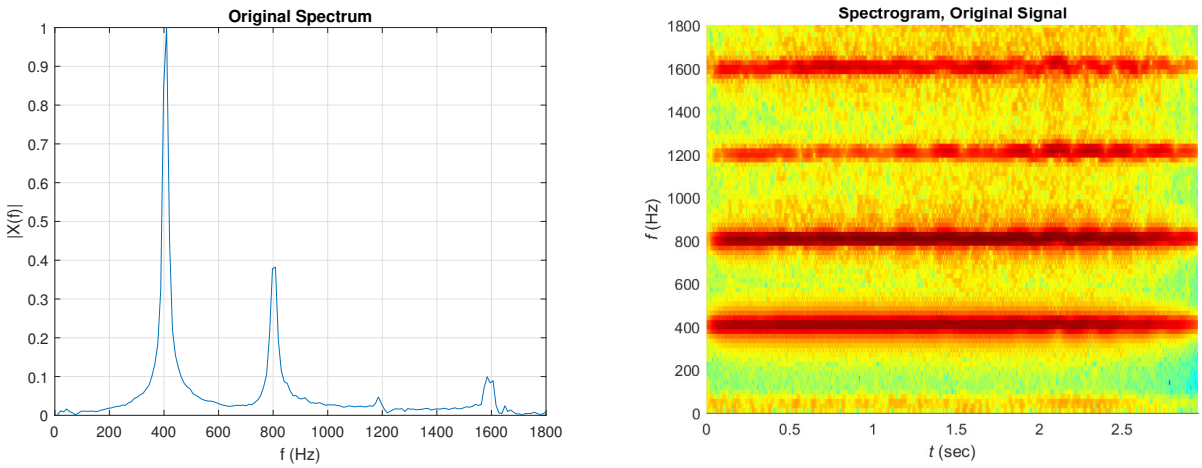


Figure 2: Original signal, notice the peak at 400 Hz.

Notice that the fundamental harmonic in the original audio is at 400 Hz. When we use $r = 2$ to shift these frequencies down, we find the fundamental frequency shifts to $400 \times r = 800$ Hz. Similarly, using $r = \frac{1}{2}$, we see the fundamental frequency shifted to $400 \times r = 200$ Hz. The audio in each case plays for about the same amount of time, but the pitch is altered.

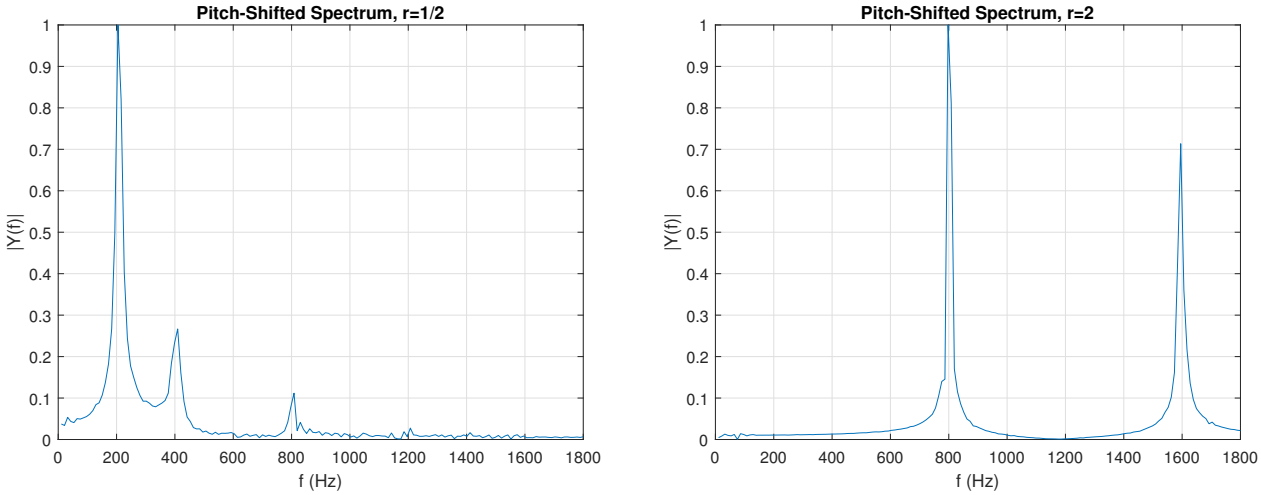


Figure 3: The altered spectrums. (1)

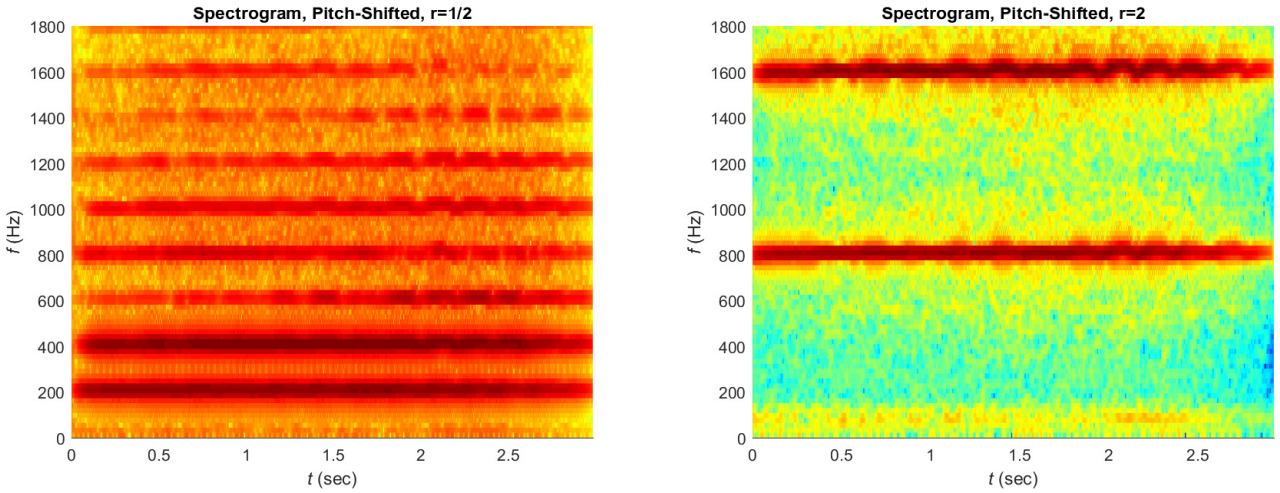
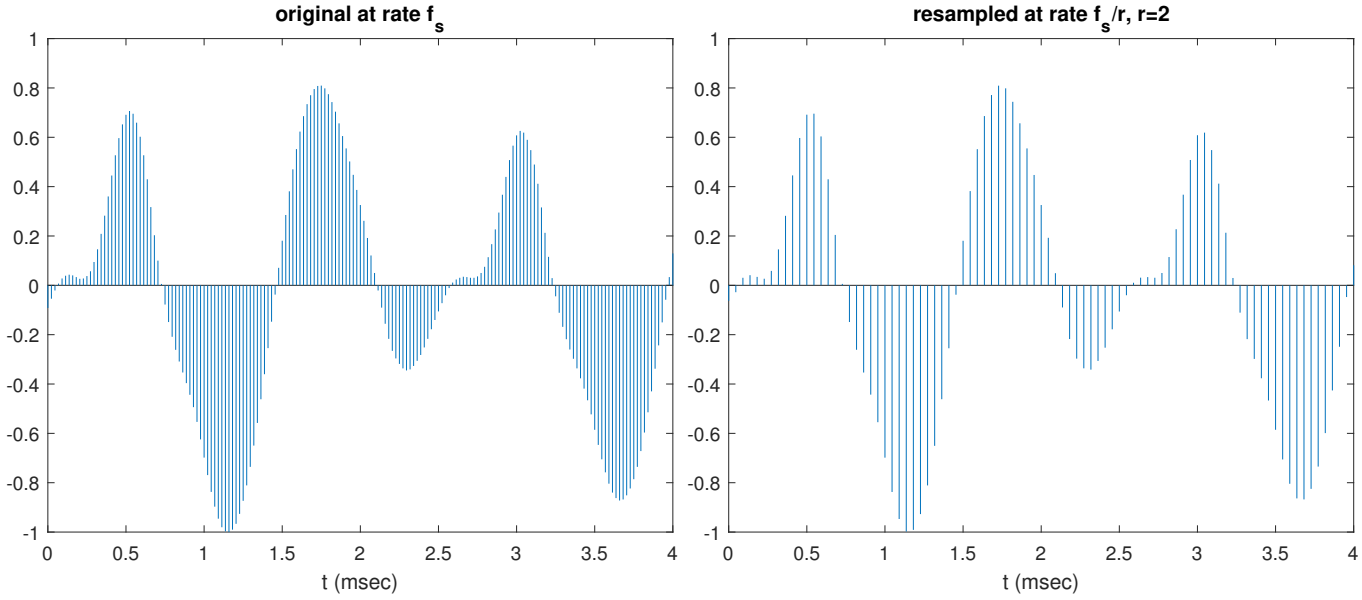


Figure 4: The altered spectrums. (2)

Visualization

Let's explore what happens in the pitch modulation further. Below is 4 milliseconds of the flute audio, unaltered, in the time-domain.



Next, we pass the original signal into the **resample** function, for $r = 2$. As we can see on the right, the envelope of the signal is unchanged, but with the sample rate cut in half, every other sample is just missing.

If we play the resampled signal at half the original sampling frequency, it sounds nearly the same. That being said, if you replay the resampled signal at its original sampling frequency, you get the "chipmunked" audio effect. The voice actors for the Chipmunk movies have said in interviews that they have to read the lines very slowly so that their audio can be sped up and made to sound mousier! Pretty weird approach considering that the figure below on the right shows a way that we can end up with chipmunked audio using only the previously described phase vocoder - you can see the added higher frequencies, although the signal is still 4 milliseconds long!

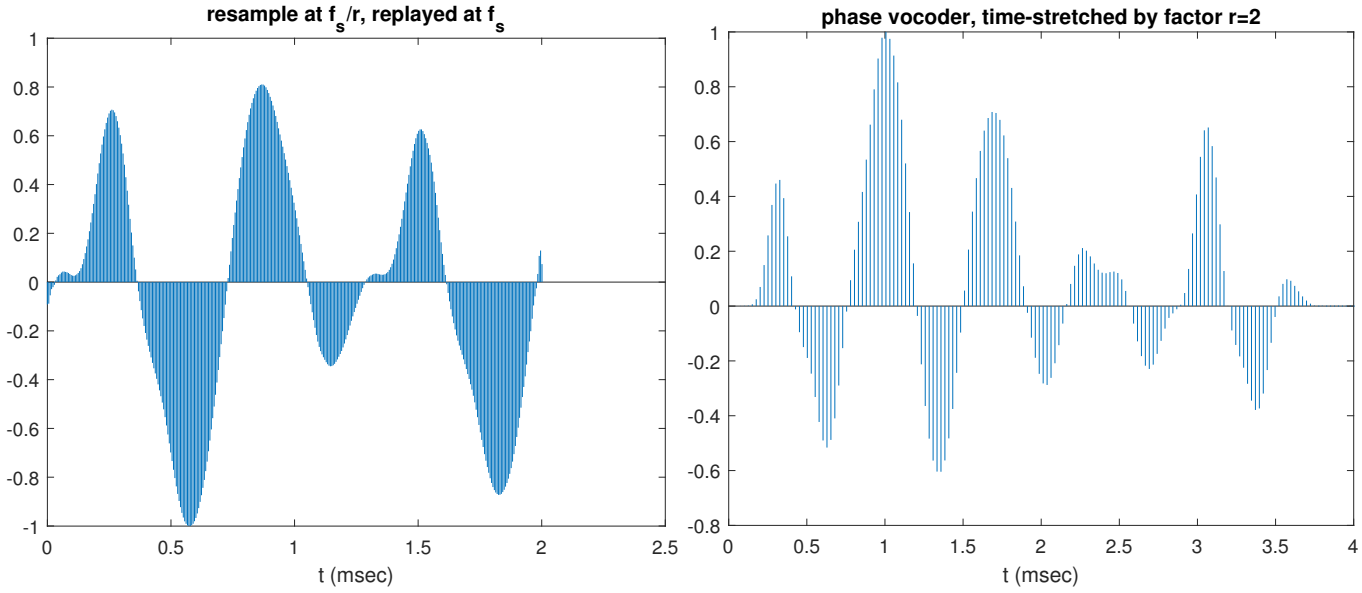


Figure 5: Chipmunked audios.

Part III: COLA Property Proof

The Constant Overlap-Add (COLA) property refers to the property that allows us to use Overlap-Add (OLA) reconstruction to account for our windowing. If this property is fulfilled at a particular R, then our STFT will properly represent the whole signal's frequency domain content.

For the COLA property to be satisfied, a DFT taken at R points should only be nonzero at the constant frequency, $\omega_r = 0$. What this effectively means, is that the windows end up behaving similar to a constant multiplier across all of the chunks in the time domain, and an impulse in the frequency domain. Consider the length-128 hanning window we used previously, $w(n) = 0.5 - 0.5\cos(\frac{2\pi n}{N-1})$, and an overlapped hanning window with $M = 10$, $\tilde{w}(n) = \sum_{m=0}^{M} w(n - mR)$. Let's examine how different hop sizes, R, affect the windowing.

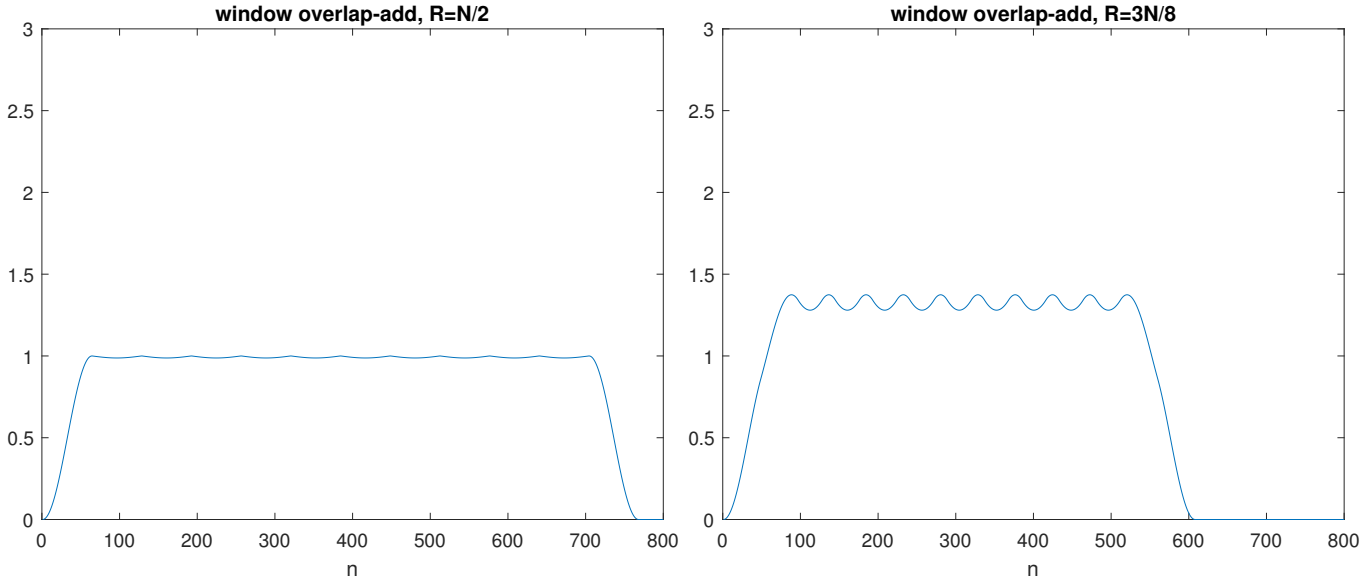


Figure 6: Windows in the time-domain. (1)

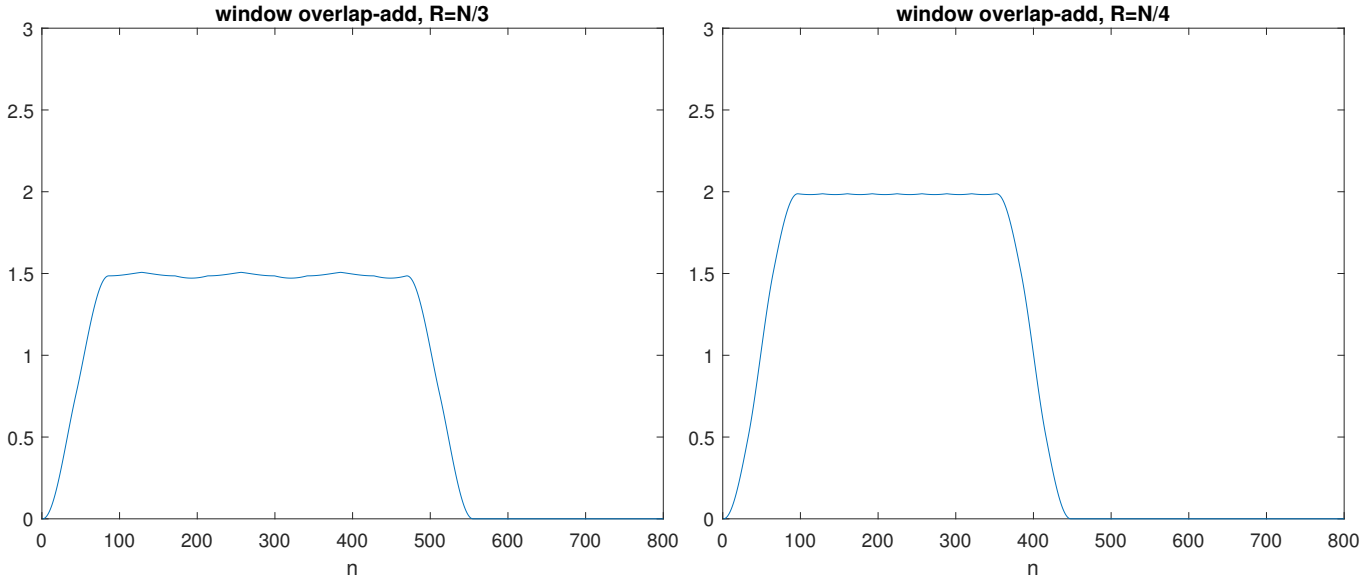


Figure 7: Windows in the time-domain. (2)

Note that in both the biggest and the smallest hop sizes, $R = \frac{N}{2}$ and $R = \frac{N}{4}$, the windows plateau into a tabletop. This can be observed in the frequency domain too.

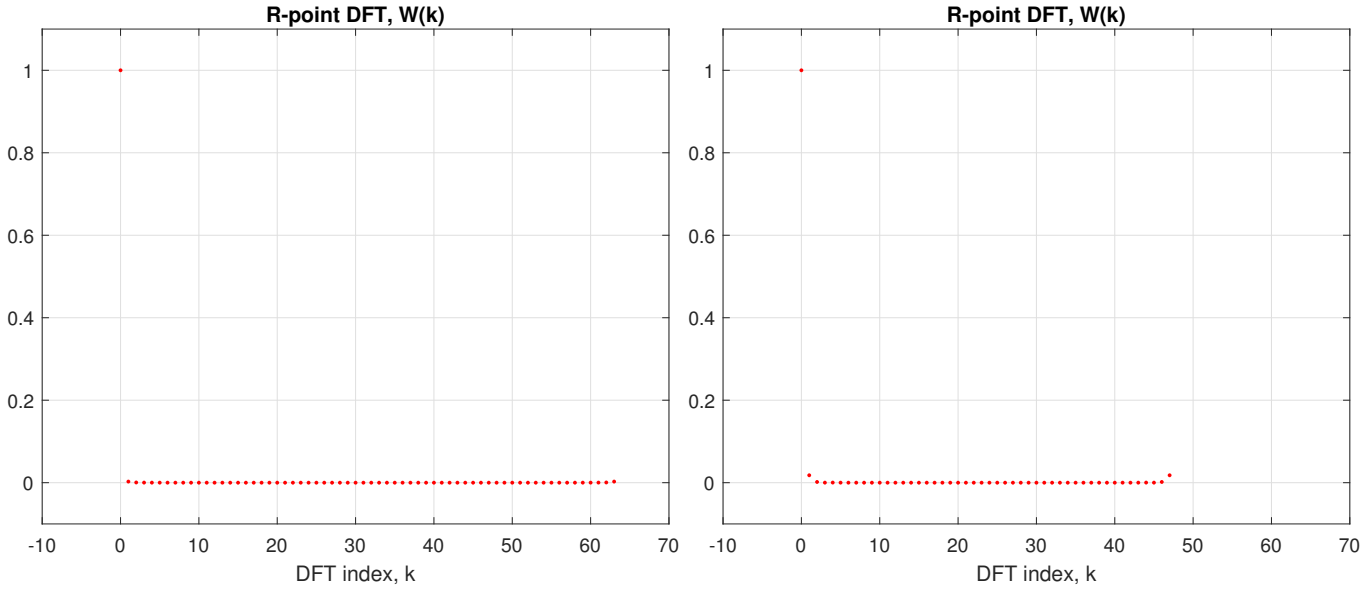


Figure 8: Windows in the frequency-domain for R=N/2 (left) and R=3N/8 (right). (1)

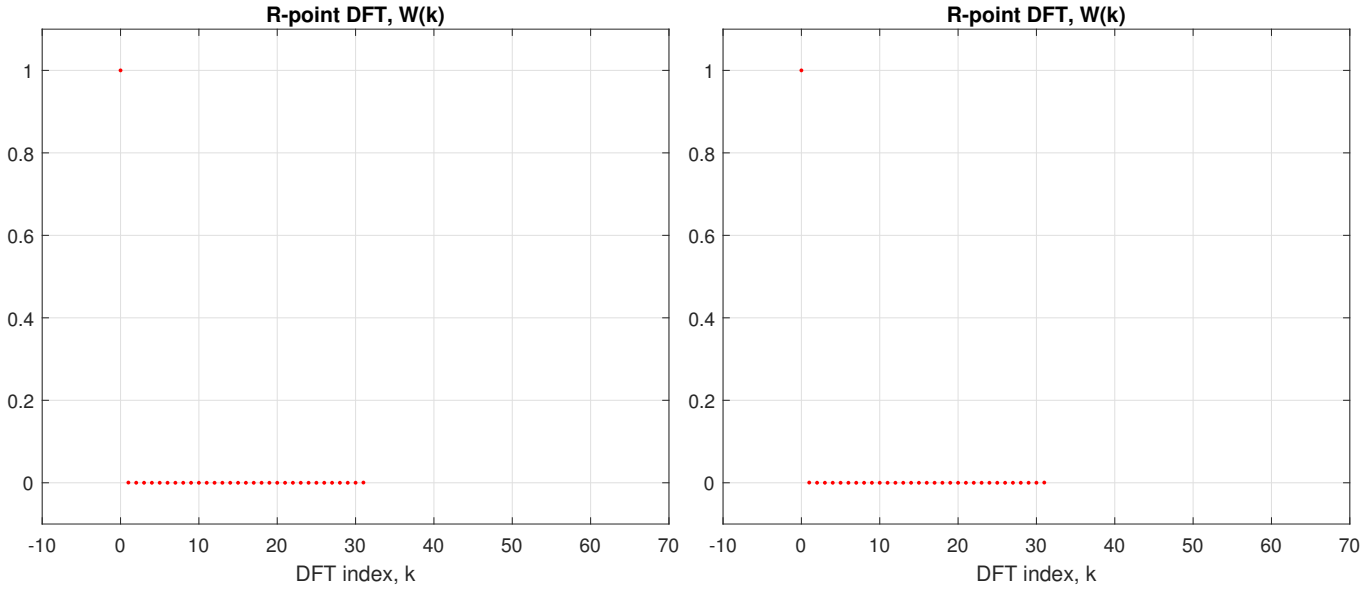


Figure 9: Windows in the frequency-domain for R=N/3 (left) and R=N/4 (right). (2)

Note that almost all of the frequency content is in the DC frequency — satisfying the COLA property.

Appendix

Project5.m

```
%3.1
addpath('C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files');

[x,Fs]=audioread('C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-1-files\original.wav');
%% 3.1.a and 3.1.b----------------------------------------------------------
%phvoc(x,r,Ra,N)
yfast=phvoc(x,1.4,50,3000);
yfast=yfast./max(yfast); %normalize
sound(yfast,Fs);
audiowrite('C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\phvocFast.wav',...
    [x; zeros(Fs,1); yfast], Fs);

yslow=phvoc(x,.7,50,3000);
yslow=yslow./max(yslow);
sound(yslow,Fs);
audiowrite('C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\phvocSlow.wav',...
    [x; zeros(Fs,1); yslow], Fs);

figure; plot((1:length(x))./Fs,x)
xlabel('time (seconds)'); title('Original Audio')
%----
fastfreq=fft(yfast);
fastmag=abs(fastfreq); fastph=angle(fastfreq);
slowfreq=fft(yslow);
slowmag=abs(slowfreq); slowph=angle(slowfreq);
figure;
subplot(2,2,1);plot((1:round(length(fastmag)/4))./3000./17,fastmag(1:round(end/4)));
xlabel('frequency w/pi'); title('Sped up magnitude')
subplot(2,2,3);plot((1:round(length(slowmag)/8))./3000./16,slowmag(2:2:round(end/4)));
xlabel('frequency w/pi'); title('Slowed down magnitude')
subplot(2,2,2);plot(1:100,fastph(1:100));
title('Sped up phase')
subplot(2,2,4);plot(1:100,slowph(1:100));
title('Slowed down phase')

print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\phvoc', '-depsc')

%phase vocoder changes time stuff without changing frequency content
%% 3.2.a

[x,FS] = audioread('flute2.wav'); x = x(:,1).';

% y = pitchmod(x, r, Ra, N)
yhigh=pitchmod(x,2,256,2048);
%STILL need to take real part >:(
ylow=pitchmod(x,1/2,256,2048);
%should these boys changing how long they are in time?
soundsc(x,FS)
soundsc(yhigh,FS)
soundsc(ylow,FS)
```

```
freq=abs(fft(x,4096));
freq=freq./max(freq);
figure;
plot((1:(round(1800/(FS)*4096)))./334*1800*2,freq(1:(round(1800/(FS)*4096))))
xlabel('f (Hz)'); ylabel('|X(f)|'); title('Original Spectrum')
grid on;
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\
    OriginalSpectrum2', '-depsc')
freq=abs(fft(yhigh,4096));
freq=freq./max(freq);
figure;
plot((1:(round(1800/(FS)*4096)))./334*1800*2,freq(1:(round(1800/(FS)*4096))))
xlabel('f (Hz)'); ylabel('|Y(f)|'); title('Pitch-Shifted Spectrum, r=2')
grid on;
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\
    HighSpectrum2', '-depsc')
freq=abs(fft(ylow,4096));
freq=freq./max(freq);
figure;
plot((1:(round(1800/(FS)*4096)))./334*1800*2,freq(1:(round(1800/(FS)*4096))))
xlabel('f (Hz)'); ylabel('|Y(f)|'); title('Pitch-Shifted Spectrum, r=1/2')
grid on;
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\
    LowSpectrum2', '-depsc')

%% 3.2.c
load x4.mat;
fs=44100;
figure;
stem((0:1/fs:(.004)).*1000,x4,'marker','none')
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\
    COrig', '-depsc')

r=2;
[p,q]=rat(1/r);
x=resample(x4,p,q);
figure; stem((0:1/2/fs:(.004+1/2/fs)).*1000,x,'marker','none')
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\Cr2'
    , '-depsc')

figure; stem((0:1/4/fs:(.002+1/4/fs)).*1000,x,'marker','none')
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\Cr1
    -2', '-depsc')

y=phvoc(x,1/r,4,32);
figure;
stem(linspace(0,4,length(y)),y./max(y),'marker','none')
title('phase vocoder, time-stretched by factor r=2')
xlabel('t (msec)')
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\
    Cphvoc', '-depsc')

%figure 2 and this method changes the frequency content and pushes the time
%back to normal

%% 3
N=128; M=10;
% n=1:N;

% w=.5-.5.*cos(2*pi*(1:N)./(N-1))';
```

```
w=@(n) .5-.5.*cos(2*pi*(n)./(127))';
wn=w(0:(N-1));
wprime=zeros(800,1);
R=N/2;
for m=0:M
    %n=n+1
    wprime(((R*m+1):(R*m+128)))=wprime(((R*m+1):(R*m+128))) + w(0:(N-1));
    %hold on; plot(wprime,'color',[(m+1)/19 .5 (m+1)/11])
end
figure;
plot(wprime); title('window overlap-add, R=N/2')
xlabel('n'); axis([0 800 0 3])
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\winN
    -2', '-depsc')
%get dft from equation 7
r=0:(R-1);
wr=2*pi.*r./R;
W=zeros(R,1);
for n=0:(N-1)
    W=W+w(n).*exp(-1j.*wr.*n)';
end
plot(abs(W)./max(abs(W)),'r.'); title('R-point DFT, W(k)')
grid on; xlabel('DFT index, k'); axis([-10 70 -.1 1.1])
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\dftN
    -2', '-depsc')

%-----
wprime=zeros(800,1);
R=N*3/8;
for m=0:M
    wprime(((R*m+1):(R*m+128)))=wprime(((R*m+1):(R*m+128))) + w(0:(N-1));
end
figure;
plot(wprime); title('window overlap-add, R=3N/8')
xlabel('n'); axis([0 800 0 3])
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\
    winN3-8', '-depsc')
%get dft from equation 7
r=0:(R-1);
wr=2*pi.*r./R;
W=zeros(R,1);
for n=0:(N-1)
    W=W+w(n).*exp(-1j.*wr.*n)';
end
plot(r,abs(W)./max(abs(W)),'r.'); title('R-point DFT, W(k)')
grid on; xlabel('DFT index, k'); axis([-10 70 -.1 1.1])
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\
    dftN3-8', '-depsc')

%-----
wprime=zeros(800,1);
R=N/3;
for m=0:M
    wprime(((R*m+1):(R*m+128)))=wprime(((R*m+1):(R*m+128))) + w(0:(N-1));
end
figure;
plot(wprime); title('window overlap-add, R=N/3')
xlabel('n'); axis([0 800 0 3])
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\winN
    -3', '-depsc')
```

```
%get dft from equation 7
r=0:(round(R)-1);
wr=2*pi.*r./R;
W=zeros(round(R),1);
for n=0:(N-1)
    W=W+w(n).*exp(-1j.*wr.*n)';
end
plot(r, abs(W)./max(abs(W)),'r.'); title('R-point DFT, W(k)')
grid on; xlabel('DFT index, k'); axis([-10 70 -.1 1.1])
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\dftN
    -3', '-depsc')

%------
wprime=zeros(800,1);
R=N/4;
for m=0:M
    wprime(((R*m+1):(R*m+128)))=wprime(((R*m+1):(R*m+128))) + w(0:(N-1));
end
figure;
plot(wprime); title('window overlap-add, R=N/4')
xlabel('n'); axis([0 800 0 3])
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\winN
    -4', '-depsc')
%get dft from equation 7
r=0:(round(R)-1);
wr=2*pi.*r./R;
W=zeros(round(R),1);
for n=0:(N-1)
    W=W+w(n).*exp(-1j.*wr.*n)';
end
plot(r,abs(W)./max(abs(W)),'r.'); title('R-point DFT, W(k)')
grid on; xlabel('DFT index, k'); axis([-10 70 -.1 1.1])
print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\dftN
    -4', '-depsc')
```

phvoc.m

```
function y = phvoc(x,r,Ra,N)
% y = phvoc(x,r,Ra,N)
%
% x = input augio signal
% r = speed-up factor
% Ra = analysis hop size, synethesis hop size Rs = round(Ra/r)
% N = FFT frame length
%
% y = output signal

X = stft(x,Ra,N); % STFT of input, implementing Eq. (11)
%clear x
Y = phmap(X,r,Ra,N); % phase remapping, implementing Eqs. (15) and (16)
%clear X
Rs=round(Ra./r); %new speed
y = istft(Y,Rs,N); % ISTFT/OLA, implementing Eqs. (12) and (13)
clear Y

end
```

pitchmod.m

```
function y = pitchmod(x, r, Ra, N)
```

```
% x = input audio signal
% r = pitch scaling factor
% Ra = analysis hop
% N = FFT frame length
%
% y = output signal

[p,q]=rat(1/r);
xtemp=resample(x,p,q);
y = phvoc(xtemp,1/r,Ra,N);

end
```

stft.m

```
function X = stft(x,Ra,N)
% X = stft(x,Ra,N)
% Short Time Fourier Transform

w=.5-.5.*cos(2*pi*(1:N)./(N-1))'; %check 0/1 indexing

Xtemp=buffer(x,N,N-Ra,'nodelay'); %x chunks
[N1,M]=size(Xtemp); %grab M
W=repmat(w,1,M);   %window
X=fft(W.*Xtemp,N);

end
```

istft.m

```
function y = istft(Y,Rs,N)
% y = istft(Y,Rs,N)

w=.5-.5.*cos(2*pi*(1:N)./(N-1))'; %check 0/1 indexing

%y=zeros(size(Y));
ytemp=real(ifft(Y,N));

[N1,M]=size(Y); %N1=N?
y=zeros(M*Rs+N,1); %?

for m=0:(M-1) %?
    for n=1:(N)
        y(m*Rs+n)=y(m*Rs+n)+ytemp(n,m+1).*w(n);
    end
end
end
```

phmap.m

```
function Y = phmap(X,r,Ra,N)
%phase modification
%want |Y|=|X|

Ymag=abs(X);
%X(k,m)=xm(k)

Rs=round(Ra./r); %new speed
psi=zeros(size(X));
```

```
phi=angle(X);
[N1,M]=size(X); %check this, N1=N if right?
w=zeros(size(X));
dw=w;
mod2pi=@(x) mod(x+pi,2*pi)-pi;

for k=1:N %check 0/1 indexing

    wk=2*pi*(k-1)/N;
    psi(k,1)=phi(k,1); %check indexing
    for m=2:M %check
        dw(k,m)=1/Ra.*mod2pi(phi(k,m)-phi(k,m-1)-Ra*wk);
        w(k,m)=wk+dw(k,m);
        psi(k,m)=psi(k,m-1)+Rs.*w(k,m);
    end
end
Y=Ymag.*exp(1j.*psi);
end
```

spectrogram.m

```
R = 256; % hop size
N = 2048; % FFT length
[x,fs] = audioread('flute2.wav'); x = x(:,1).';
yhigh=pitchmod(x,2,256,2048);
%STILL need to take real part >:(
ylow=pitchmod(x,1/2,256,2048);
X = stft(ylow,R,N); % assumed to have size Nx(M+1)
M = size(X,2) - 1;
k = 0:N/2; % positive-frequency half of Nyquist interval
f = k*fs/N; % 0 <= f <= fs/2
t = (0:M)*R/fs; % M+1 time frames spaced at hop interval R*Ts = R/fs
Xmag = abs(X(k+1,:)); % extract positive-frequency half
Xmag = Xmag/max(max(Xmag)); % normalize to unity maximum
S = 20*log10(Xmag); % dB
surf(t,f,S,'edgecolor','none'); % spectrogram plot, t horizontal, f vertical
colormap(jet);
axis tight; view(0,90);
xlabel('{\itt} (sec)');
ylabel('{\itf} (Hz)');
ylim([0,1800]); % show range 0 <= f <= 1800 Hz

print( 'C:\Users\shsab\OneDrive\Documents\FALL 2017\DSPD\project-5-files\
    HighGram', '-depsc')
```