SIGNAL ANALYSIS/SYNTHESIS USING THE DFT

Chinmay Sahu

Communications, Signal Processing, Networking Lab
Department of Electrical and Computer Engineering
Clarkson Univeristy,Potsdam,13699

ABSTRACT

This paper presents a discrete Fourier transform (DFT) based audio compression technique.This involves a three-step process. First, the audio signal is transformed in terms of frequency components by using DFT. Next, the audio signal is compressed by selectively picking peaks in DFT magnitude spectrum while removing redundant elements. Peak picking is performed either by selecting n dominant components(Method 1) or by choosing first n components (Method 2). Finally, the resultant frequency domain results are transformed back to time domain using inverse DFT. Sliding rectangular window and sliding triangular window are multiplied to the audio signal for different values of N-point FFT to evaluate the quality of reconstructed audio. Using reconstructed audio signal, overall and segmental signal to noise ratio(SNR) is estimated for different values of N(64,128,256). The results are tabulated, analyzed based on different values of peak picking(n).

Index Terms— DFT, peak picking, audio compression

1. INTRODUCTION

Data compression is one of the needs of the modern day[1]. For example, with the explosive growth of the Internet, there is an increasing demand for audio compression, or data compression in general purpose of such compressions is to minimize the storage space[2] .Data compression using transformations such as the DCT and the DFT are the basis for many coding standards such as JPEG, MP3, and AC-3[3, 4].

The Fast Fourier Transform (FFT) is one of such data compression algorithms. The FFT is a DFT algorithm which lessens the number of computations from something on the order of N^2 to N*log (N)[5, 6]. The Fast Fourier Transform greatly simplifies the computations for large values of N, where N is the number of samples in the sequence. The idea behind the FFT is the divide and conquer approach, to break up the original N point sample into two (N/2) sequences. This is because a set of smaller problems are simpler to solve than one large one. The DFT requires $(N-1)^2$ complex multiplications and N (N-1) complex additions as opposed to the FFTs approach of splitting it down into a series of 2 point samples which only require 1 multiplication and 2 additions and the recombination of the points which is minimal[7, 8].

In this project, FFT (IFFT) is used for the compression (decompression) of a speech signal. This data compression plan is simulated using Matlab. Simulations are performed for different FFT sizes and a different number of components chosen. Two different methods used for the plan are:

- By retaining dominant n-components (Method-1)
- By retaining the first n-components (Method-2)

The speech data is compressed using both sliding data window and sliding triangular windowing technique. The SNRs (signal to noise ratios) are estimated for all the simulations and used to study the performance of the compression scheme using FFT. Also, the noise introduced in the signal (for various cases) is studied both by listening to the recovered signal and by the calculated SNR's.

The rest of paper is organized as follows.The Section 2 discusses about implementation of FFT algorithm along with MATLAB code. The Section 3 discusses about the figures and results followed by concluding remarks in Section 4.

2. MATLAB IMPLEMENTATION

First,The audio file "cleanspeech" is loaded into Matlab and saved in the speech data vector "audio_signal". The data set (in the vector) to be compressed is then segmented into N-point segments (or frames) using a sliding rectangular/triangular window.Then, the data is compressed using two methods as discussed in Subsection 2.1 and Subsection 2.2.

2.1. By retaining dominant n-components (Method-1)

In this method, The FFT of each segment is taken one by one by passing it into the loop N (the number of frames) times as shown in matlab code[line:1-3] listed in next page. Thus the magnitude spectrum of the signal is obtained. The result of the N point FFT has (1+N/2) independent components. This is due to the symmetry property of the DFT. These (1+N/2)

This project acknowledges the input of Prof. Mahesh Banavar for his valuable guidance.

points are retained as they are adequate to get back the all the information from the primary audio signal.From this set of about half the points, dominant "n" points are chosen to reconstruct the original signal i.e., the points with maximum magnitude[line:4-11].In the simulation, this is done for all possible n values from 1 to (1+N/2). The rest of the (1+N/2-n)

```
1  for i = 1:N:length(audio_signal)
2      Xt=audio_signal(i:i+(N-1));
3      N_frame_FFT=fft(Xt);
4      [sortedValues,sortIndex]=
5      sort(abs(N_frame_FFT(1:1+(N/2))),'descend');
6      dominant_peaks = sortIndex(1:n);
7      peak_keepers=zeros(1+(N/2),1);
8      for q=1:1:n
9      peak_keepers(dominant_peaks(q))=
10     N_frame_FFT(dominant_peaks(q));
11     end
12     Dom_FlipSig=flipud(conj(peak_keepers));
13     Dom_Sym_Sig=...
14     [peak_keepers;Dom_FlipSig(2:N/2)];
15     Dom_N_frame_IFFT=ifft(Dom_Sym_Sig,N);
16     Dom_recon_array=
17     [Dom_recon_array;Dom_N_frame_IFFT];
18   end
```

points are padded with zeros.Special care is taken to make the chosen dominant n points lie at the indices they previously were (in the signal with 1+N/2 components).Then, before taking the IFFT, The symmetry of speech data is maintained. In order to reconstruct the symmetry in the signal, the conjugate of the dominant n-component vector is taken.The first and last components in this new vector are discarded as they are dc values which do not take part in the symmetry building before taking IFFT. The conjugate vector is flipped and added to the original dominant n-component vector[line:12-18]. Hence, The signal with symmetrical properties is obtained and taking the IFFT results in all real values.

2.2. By retaining the first n-components (Method-2)

This method is very similar to the one we have discussed above.The matlab code relevant to this method is listed below. The only difference is in the way we select the "n" points for

```
1  for i = 1:N:length(audio_signal)
2      Xt=audio_signal(i:i+(N-1));
3      N_frame_FFT=fft(Xt);
4      first_n_picks=N_frame_FFT(1:n);
5      first_n_picks=
6      [first_n_picks;zeros((1+(N/2)-n),1)];
7      FlipSig=flipud(conj(first_n_picks));
8      Sym_Sig=[first_n_picks;FlipSig(2:N/2)];
9      N_frame_IFFT=ifft(Sym_Sig,N);
10     Reconstructed_array=
11     [Reconstructed_array;N_frame_IFFT];
12 end
```

the signal. In the previous case, the dominant n components are chosen and set the rest to zero.However, In this case, the first n points are chosen[line:4]. The rest of the (1+N/2-n) points are set to zero[line:5-6]. Then symmetry of speech data is maintained similarly as discussed in method-1.

3. RESULTS

The performance of each approach is evaluated by plotting overall SNR and segmental SNR[9] against percentage of component selected(n/N). The overall SNR is defined by [10].

$$SNR_{overall} = 10\, log_{10} \frac{\sum_{l=1}^{L} {x_l}^T x_l}{\sum_{l=1}^{L} {e_l}^T e_l} \quad (1)$$

The segmental SNR is defined by [11],where x_l represents

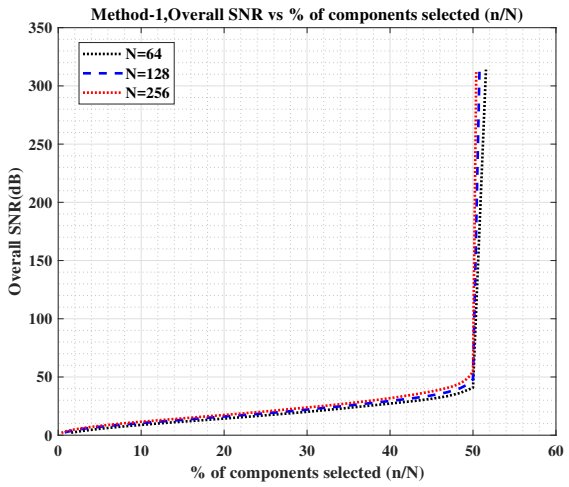
$$SNR_{segm} = \frac{10}{L} \sum_{l=1}^{L} log_{10} \frac{{x_l}^T x_l}{{e_l}^T e_l} \quad (2)$$

the lth frame of original signal.e_l represents the error vector and L represents the total number of frames.The overall and Segmental SNR for Method-1and Method-2 using rectangular window are listed next page in Figure 1 and Figure 2. Similarly,The overall and Segmental SNR for Method-1and Method-2 using sliding triangular window are listed last page in Figure 3 and Figure 4.In figures,The x-axis represents the percentage of components selected while y-axis represents the corresponding SNR values.The observations from figures and tables(listed in Appendix) are discussed in following subsections.

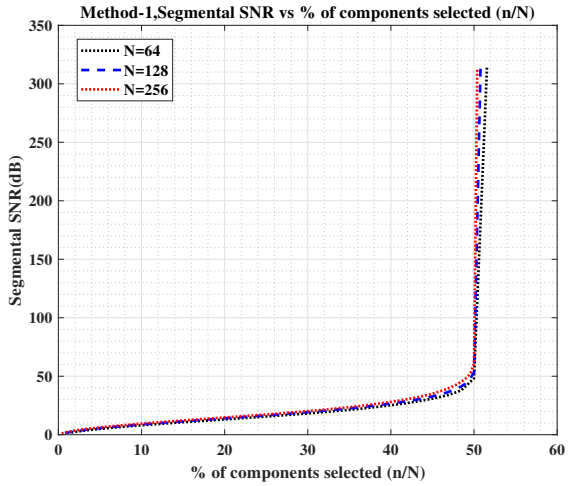
3.1. Effect of n / N on SNR

N corresponds to the number of components of the signal chosen. This implies by increasing value of n, the resolution of the signal can be improved. When n is (1+ N/2), the reconstructed audio matches with the original speech signal. This insinuates that there should be an enhancement in the quality of sound as the value of n increases, both in the case of first n and dominant n methods. This is indeed the case and is supported by the audio signal created by the process. A signal with increased n gives a better quality audio signal. This can also be observed from the SNR values. The SNR values increase as we increase the n value from 1 to (1+ N/2). The drawback of choosing large n is that the size of the file starts getting bigger as we increase n.

Here, N refers to the size of FFT used. This is also the length of the window used for the simulation of a particular N sized FFT. In this paper, simulations are carried out for N= 64, 128, and 256. As value N increases, It is expected that the resolution of signal increases as the number of samples considered increases. This improves the quality of the audio signal. In this case, the quality of the audio signal concurrently depends on N and n values. So if N value is increased but n value is chosen to be very low, the overall signal will not be a high-quality signal.This can be validated both SNR and

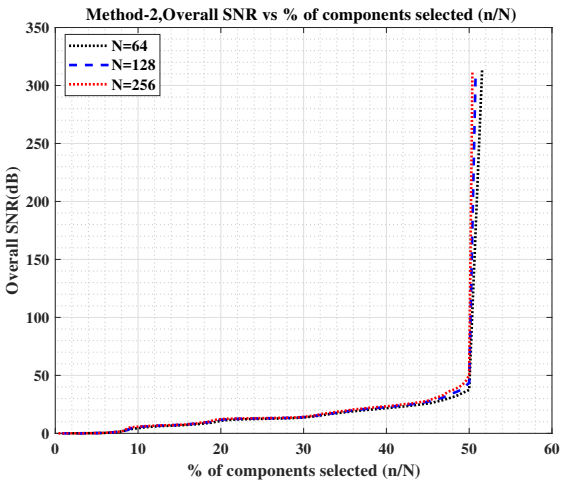


(a) Overall SNR

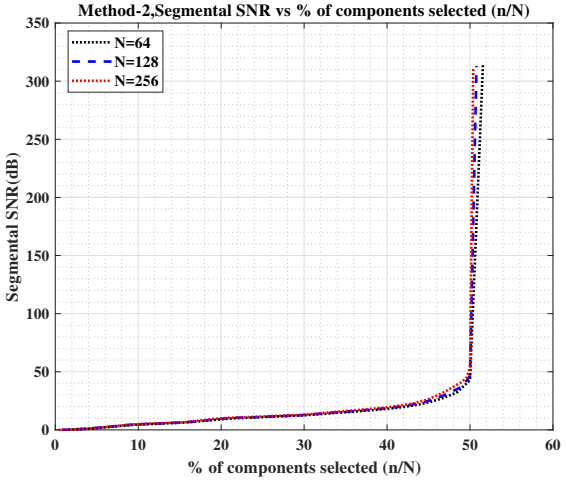


(c) Segmental SNR

Fig. 1: Overall and Segmental SNR's for Rectangular windowing technique using Method-1.



(a) Overall SNR



(b) Segmental SNR

Fig. 2: Overall and Segmental SNR's for Rectangular windowing technique using Method-2.

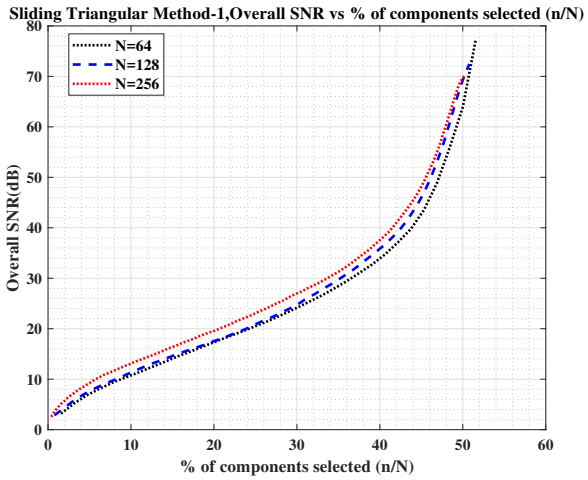
by the listening to the audio clip for n=1 and N=64,128,256. Picking N to be 64,128,256, the best quality compressed signal will be composed of 33,65,129 non-zero components respectively.Here, best quality means picking n=1+(N/2).

3.2. Difference between Method-1 and Method-2

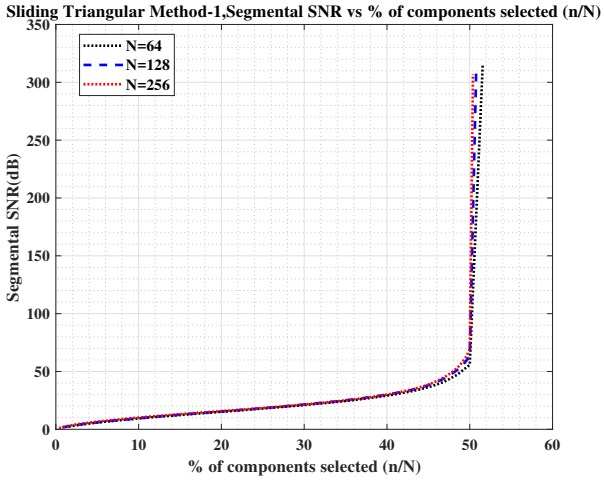
In Method-1, first n components are selected which usually gives a relatively poor result compared to the results produced by the method of choosing n-dominant components. This can be seen from the SNR plots. Picking n dominant peaks(Method-2) results in a better audio signal as the audio peaks with distinct magnitude are chosen. Choosing the first n components(Method-1) might result in not useful information as it might be removing the significant part of the signal, which is unlikely in Method-1. Also, the SNR values turn out to be the same for both Method-1 and 2 when n=1+(N/2).

3.3. Effect of triangular windows

In DFT, the signal is split into N-point frames. This results in the discontinuity in phase from frame to frame. So while reconstructing the audio signal using rectangular window, there is a loss of phase from frame to frame. Hence, there are clicking sounds in the reconstructed audio file. This can be avoided using sliding triangular method as sliding triangular window smoothen out the discontinuity in phase from frame to frame. Hence, the overall SNR is also a smooth curve unlike overall SNR of rectangular window method.Also, For given values of n, the SNR values are better in case of triangular window-

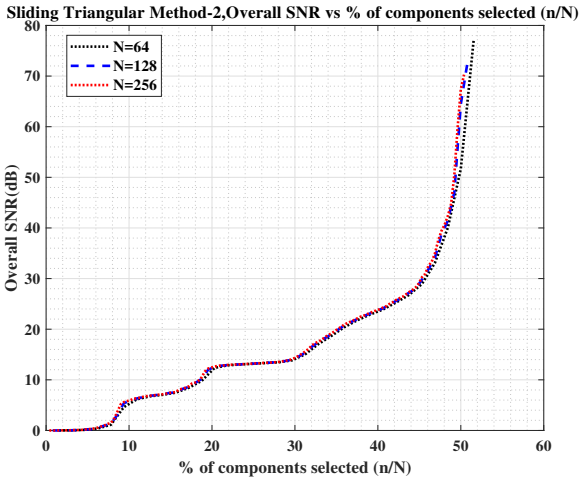


(a) Overall SNR

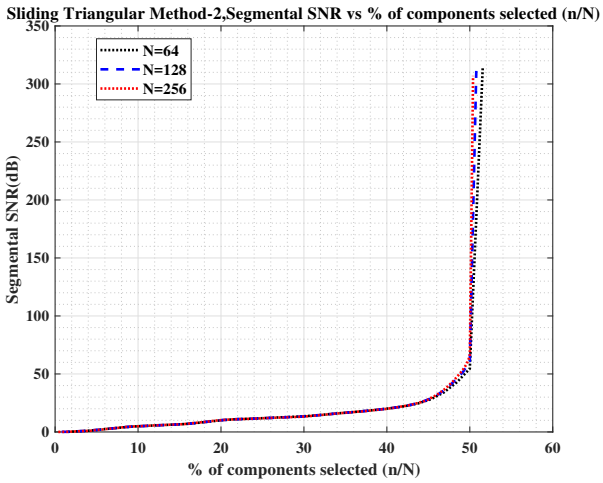


(c) Segmental SNR

Fig. 3: Overall and Segmental SNR's for Sliding triangular windowing technique using Method-1.



(a) Overall SNR



(b) Segmental SNR

Fig. 4: Overall and Segmental SNR's for Sliding triangular windowing technique using Method-2.

ing technique compared to rectangular windowing plan except for n=1+(N/2).This is because first and last (N/2) terms are multiplied by the single triangular window instead of sliding triangular window. so overall SNR at n=1+(N/2) is less corresponding to the overall SNR of the rectangular windowing plan.

4. CONCLUSION

In this paper, a DFT based speech compression algorithm is presented.The algorithm is implemented by two methods i.e. a. By retaining n dominant component and b. By retaining first n components. Both methods are implemented using rectangular windowing technique as well as sliding triangular windowing technique.The overall and segmental SNR values are estimated, plotted and analyzed for n=1 to 1=1+(N/2).

For low values of n, keeping the N constant, the quality of voice obtained with method-1 is much better. However, In the case of method-2, it is hard to differentiate between noise and voice. It seemed that the voice signal obtained by method-1, and method-2 can be compared to FM and AM radio respectively. When picking n to be in the mid-range values, the method-2 produced a voice signal with noise having clipping sound more than method-1. But with sliding triangular windowing, it sounded smooth. This is because it eliminates the discontinuity in phase. At high values of n, the voice signals generated by using the two different methods sounded almost identical.This can be validated by comparing overall and segmental SNR for n=1+(N/2) for both methods. The tables are listed in Appendix due to restrictions on space.

5. REFERENCES

[1] David Salomon, *Data compression: the complete reference*, Springer Science & Business Media, 2004.

[2] Khalid Sayood, *Introduction to data compression*, Newnes, 2012.

[3] Gregory K Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[4] William B Pennebaker and Joan L Mitchell, *JPEG: Still image data compression standard*, Springer Science & Business Media, 1992.

[5] Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, and Takashi Miyata, "Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform," *Nucleic acids research*, vol. 30, no. 14, pp. 3059–3066, 2002.

[6] Johan Schoukens, Rik Pintelon, and Hugo Van Hamme, "The interpolated fast fourier transform: A comparative study," *IEEE Transactions on Instrumentation and Measurement*, vol. 41, no. 2, pp. 226–232, 1992.

[7] Martin Vetterli, Henri J Nussbaumer, et al., "Simple fft and dct algorithms with reduced number of operations," *Signal processing*, vol. 6, no. 4, pp. 267–278, 1984.

[8] B Srinivasa Reddy and Biswanath N Chatterji, "An fft-based technique for translation, rotation, and scale-invariant image registration," *IEEE transactions on image processing*, vol. 5, no. 8, pp. 1266–1271, 1996.

[9] Andreas S Spanias, *Digital signal processing: An interactive approach*, J-DSP Editor, 2007.

[10] Mazen O Hasna and M-S Alouini, "End-to-end performance of transmission systems with relays over rayleigh-fading channels," *IEEE transactions on Wireless Communications*, vol. 2, no. 6, pp. 1126–1131, 2003.

[11] Israel Cohen and Baruch Berdugo, "Noise estimation by minima controlled recursive averaging for robust speech enhancement," *IEEE signal processing letters*, vol. 9, no. 1, pp. 12–15, 2002.

Appendix

Matlab Code For Sliding Rectangular Window

```
% Author: Chinmay Sahu
% Class: EE 501 - Digital Signal Processing
% Project: Signal Analysis/Synthesis using the DFT
% email: sahuc@clarkson.edu
% Clarkson Univeristy
% Nov 2017; Last revision: 19-Nov-2017
clc
clear
% close all
filename = 'cleanspeech.wav';% Loading the sound file

Reconstructed_array=[]; % initializing reconstructed audio array for method-2
Dom_recon_array=[];  % initializing reconstructed audio array for method-1
first_n_picks=[]; %initializing array to keep first n picks for method-2
peak_keepers=[]; %initializing array to keep first n peaks for method-1


SNR_64_I=[];
SNR_64_II=[];
Snr_I=0;
Snr_II=0;
Snr_Seg_I=0;
Snr_Seg_II=0;

for fftPoints= 1 : 3

switch fftPoints
case 1
N=64;% Framerate
case 2
N=128;
case 3
N=256;
end


[y,Fs] = audioread('cleanspeech.wav');% read the sound file

%N=64; % decideing the  N=64,128,256
L=length(y)/N;
y1=reshape(y,[N,L]);% reshaping the matrix to N-point frames of Lth columns
first_n_picks=zeros(1+(N/2),1);
 % 1+(N/2) independent components, only n independent components are retained for data ...
     reconstruction (n<1+(N/2))
for n=1:1:(1+(N/2)) % n varies from 1: 1+(N/2) example: for N=64, n=1:33

Reconstructed_array=[];
Dom_recon_array=[];
Snr_I_Num=0;
Snr_I_Den=0;
Snr_II_Num=0;
Snr_II_Den=0;

Snr_II=0;
Snr_Seg_I=0;
Snr_Seg_II=0;

for l=1:1:L % Lth Frame

N_frame_FFT=fft(y1(:,l),N); % N frame FFT

```

```
% Method 1 dominant signal pick
[sortedValues,sortIndex] = sort(abs(N_frame_FFT(1:1+(N/2))),'descend');  %# Sort the values in ...
    descending order
dominant_peaks = sortIndex(1:n);  %Careful of symmetry Z= noumber of dominant peaks
peak_keepers=zeros(1+(N/2),1);
for q=1:1:n
peak_keepers(dominant_peaks(q))=N_frame_FFT(dominant_peaks(q)); % Keeping dominant peaks at their ...
    indices
end


Dom_FlipSig=flipud(conj(peak_keepers)); % flip
Dom_Sym_Sig=[peak_keepers;Dom_FlipSig(2:N/2)]; % Joining flipped signal
Dom_N_frame_IFFT=ifft(Dom_Sym_Sig,N); % Takinf IFFT of symmetric signal
Dom_recon_array=[Dom_recon_array;Dom_N_frame_IFFT]; % reconstructing audio data



Xt=y1(:,l); % Lth frame
et=(Xt-Dom_N_frame_IFFT);% Lth frame - estimated Lth frame

Snr_I_Num=Snr_I_Num+(Xt.'*Xt);
Snr_I_Den=Snr_I_Den+(et.'*et);

Snr_Seg_I=Snr_Seg_I+(10/L)*log10((Xt.'*Xt)/(et.'*et));

%Method-2 the first n components are retained

first_n_picks=N_frame_FFT(1:n); % picking first n picks
first_n_picks=[first_n_picks;zeros((1+(N/2)-n),1)]; % padding zeros
FlipSig=flipud(conj(first_n_picks)); % fliping after taking conjugate
Sym_Sig=[first_n_picks;FlipSig(2:N/2)]; % Joining to get symmtry
N_frame_IFFT=ifft(Sym_Sig,N); % IFFT
Reconstructed_array=[Reconstructed_array;N_frame_IFFT]; % reconstruction

Xt=y1(:,l);
et=(Xt-N_frame_IFFT);

Snr_II_Num=Snr_II_Num+(Xt.'*Xt);
Snr_II_Den=Snr_II_Den+(et.'*et);
Snr_Seg_II=Snr_Seg_II+((10/L)*log10(((Xt).'*Xt)/(et.'*et)));

end


Reconstructed_audio_array_I{n}=Dom_recon_array; % recontructed audio for n method 1
Reconstructed_audio_array_II{n}=Reconstructed_array; % recontructed audio for n method 2


% Overall SNR
SNR_I(n)=10*log10(Snr_I_Num/Snr_I_Den);
SNR_II(n)=10*log10(Snr_II_Num/Snr_II_Den);

% Segmental SNR Dom Pick
SNR_SEGMENTAL_I(n)=Snr_Seg_I;
% Segmental SNR
SNR_SEGMENTAL_II(n)=Snr_Seg_II;
end % end of for loop where we move n using Method-1 or Method-2 to find SNR ,audios



switch fftPoints
case 1 % retrive audios and SNR for 64 frame

%Retrived Audio Signals
Compressed_Audio_Signals_64_1=Reconstructed_audio_array_I;
Compressed_Audio_Signals_64_2=Reconstructed_audio_array_II;
% Overall SNR
```

```
SNR_64_1=SNR_I;
SNR_64_2=SNR_II;
% Segmental
SNR_SEGMENTAL_64_1=SNR_SEGMENTAL_I;
SNR_SEGMENTAL_64_2=SNR_SEGMENTAL_II;

case 2

%Retrived Audio Signals
Compressed_Audio_Signals_128_1=Reconstructed_audio_array_I;
Compressed_Audio_Signals_128_2=Reconstructed_audio_array_II;
% Overall SNR
SNR_128_1=SNR_I;
SNR_128_2=SNR_II;
% Segmental
SNR_SEGMENTAL_128_1=SNR_SEGMENTAL_I;
SNR_SEGMENTAL_128_2=SNR_SEGMENTAL_II;

case 3

%Retrived Audio Signals
Compressed_Audio_Signals_256_1=Reconstructed_audio_array_I;
Compressed_Audio_Signals_256_2=Reconstructed_audio_array_II;
% Overall SNR
SNR_256_1=SNR_I;
SNR_256_2=SNR_II;
% Segmental
SNR_SEGMENTAL_256_1=SNR_SEGMENTAL_I;
SNR_SEGMENTAL_256_2=SNR_SEGMENTAL_II;
end

end



% Overall SNR method 1
figure
grid on
box on
grid minor
hold on
plot(100*((1:33)/64),SNR_64_1,':k','LineWidth',2);
hold on
plot(100*((1:65)/128),SNR_128_1,'--b','LineWidth',2);
hold on
plot(100*((1:129)/256),SNR_256_1,':r','LineWidth',2);
hold on
title('Method-1,Overall SNR vs % of components selected (n/N)','FontName','Times New ...
    Roman','FontSize',12,'FontWeight','bold');
xlabel('% of components selected (n/N)','FontName','Times New ...
    Roman','FontSize',12,'FontWeight','bold');
ylabel('Overall SNR(dB)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
set(gca,'FontName','Times New Roman','FontSize',12,'FontWeight','bold');

legend({'N=64','N=128','N=256'},'FontName','Times New Roman','FontSize',12,'FontWeight','bold');


% Segmental SNR method 1
figure
grid on
box on
grid minor
hold on
plot(100*((1:33)/64),SNR_SEGMENTAL_64_1,':k','LineWidth',2);
hold on
plot(100*((1:65)/128),SNR_SEGMENTAL_128_1,'--b','LineWidth',2);
hold on
plot(100*((1:129)/256),SNR_SEGMENTAL_256_1,':r','LineWidth',2);
```

```
193
194
195 title('Method-1,Segmental SNR vs % of components selected (n/N)','FontName','Times New ...
        Roman','FontSize',12,'FontWeight','bold');
196 xlabel('% of components selected (n/N)','FontName','Times New ...
        Roman','FontSize',12,'FontWeight','bold');
197 ylabel('Segmental SNR(dB)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
198 set(gca,'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
199 legend({'N=64','N=128','N=256'},'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
200
201 % Overall SNR method 2
202 figure
203 grid on
204 box on
205 grid minor
206 hold on
207 plot(100*((1:33)/64),SNR_64_2,':k','LineWidth',2);
208 hold on
209 plot(100*((1:65)/128),SNR_128_2,'--b','LineWidth',2);
210 hold on
211 plot(100*((1:129)/256),SNR_256_2,':r','LineWidth',2);
212
213
214 title('Method-2,Overall SNR vs % of components selected (n/N) ','FontName','Times New ...
        Roman','FontSize',12,'FontWeight','bold');
215 xlabel('% of components selected (n/N)','FontName','Times New ...
        Roman','FontSize',12,'FontWeight','bold');
216 ylabel('Overall SNR(dB)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
217 set(gca,'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
218 legend({'N=64','N=128','N=256'},'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
219
220
221 % Segmental SNR method 2
222 figure
223 grid on
224 box on
225 grid minor
226 hold on
227 plot(100*((1:33)/64),SNR_SEGMENTAL_64_2,':k','LineWidth',2);
228 hold on
229 plot(100*((1:65)/128),SNR_SEGMENTAL_128_2,'--b','LineWidth',2);
230 hold on
231 plot(100*((1:129)/256),SNR_SEGMENTAL_256_2,':r','LineWidth',2);
232
233 title('Method-2,Segmental SNR vs % of components selected (n/N)','FontName','Times New ...
        Roman','FontSize',12,'FontWeight','bold');
234 xlabel('% of components selected (n/N)','FontName','Times New ...
        Roman','FontSize',12,'FontWeight','bold');
235 ylabel('Segmental SNR(dB)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
236 set(gca,'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
237 legend({'N=64','N=128','N=256'},'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
238 % filename = 'taylor.wav';
239 % audiowrite(filename,Reconstructed_array,Fs);
240 % filename = 'Dominque.wav';
241 % audiowrite(filename,Dom_recon_array,Fs);
```

Matlab Code For Sliding Triangular Window

```
% Author: Chinmay Sahu
% Class: EE 501 - Digital Signal Processing
% Project: Signal Analysis/Synthesis using the DFT
% email: sahuc@clarkson.edu
% Clarkson Univeristy
% Nov 2017; Last revision: 19-Nov-2017
clc
clear
close all
filename = 'cleanspeech.wav';% Loading the sound file

Reconstructed_array=[]; % initializing reconstructed audio array for method-2
Dom_recon_array=[];  % initializing reconstructed audio array for method-1
first_n_picks=[]; %initializing array to keep first n picks for method-2
peak_keepers=[]; %initializing array to keep first n peaks for method-1


for fftPoints= 1 : 3

switch fftPoints
case 1
N=64;% Framerate
case 2
N=128;
case 3
N=256;
end


[audio_signal,Fs] = audioread('cleanspeech.wav');% Load the sound file


first_n_picks=zeros(1+(N/2),1);

for n=1:1:(1+(N/2)) % n varies from 1: 1+(N/2) example: for N=64, n=1:33

Reconstructed_array=[]; % clearing the reconstructed array for method-2 to start over again for ...
    each n
Dom_recon_array=[]; % clearing the Dom_recon_array for method-2 to start over again for each n

Snr_I_Num=0; % Snr_I_Num reset to zero to start over again for each n
Snr_I_Den=0;  % Snr_I_Den reset to zero to start over again for each n
Snr_II_Num=0;  % Snr_II_Num reset to zero to start over again for each n
Snr_II_Den=0;  % Snr_II_Den reset to zero to start over again for each n

Snr_Seg_I=0; % segmental SNR reset to zero to start over again for each n
Snr_Seg_II=0; % segmental SNR reset to zero to start over again for each n

 %
 %%
 % *Signal Synthesis:* For Sliding triangular windowing,audio signal
 % increments in (N/2)interval till {length of audio-(N/2)}
for li=1:N/2:(32768-(N/2)) % li: increments at N/2 intervals

Xt=audio_signal(li:li+(N-1)); % Signal that is selected from 1:64, 33:96, 65:127,
Y_1=Xt.*triang(N); % multiplying triangular window with each frame

N_frame_FFT=fft(Y_1); % N frame FFT of audio signal

% Method 1 dominant signal pick
[sortedValues,sortIndex] = sort(abs(N_frame_FFT(1:1+(N/2))),'descend');  %# Sort the values in ...
    descending order
dominant_peaks = sortIndex(1:n); % n= number of dominant peaks
peak_keepers=zeros(1+(N/2),1); %initializing the peak_keepers to zeros
```

```
63
64  for q=1:1:n
65  peak_keepers(dominant_peaks(q))=N_frame_FFT(dominant_peaks(q)); %
66  end
67
68  Dom_FlipSig=flipud(conj(peak_keepers));
69  Dom_Sym_Sig=[peak_keepers;Dom_FlipSig(2:N/2)];
70  Dom_N_frame_IFFT=ifft(Dom_Sym_Sig,N);
71
72
73  if li==1
74  Dom_recon_array=[Dom_recon_array;Dom_N_frame_IFFT];
75  elseif li ≠1
76  Pad_Dom_N_frame_IFFT=vertcat(zeros(li-1,1),Dom_N_frame_IFFT); %li=33,65,97
77  Dom_recon_array=vertcat(Dom_recon_array,zeros(N/2,1));
78  Dom_recon_array=Dom_recon_array+Pad_Dom_N_frame_IFFT;
79  end
80
81
82  %Method-2 the first n components are retained
83
84  first_n_picks=N_frame_FFT(1:n);
85  first_n_picks=[first_n_picks;zeros((1+(N/2)-n),1)];
86  FlipSig=flipud(conj(first_n_picks));
87  Sym_Sig=[first_n_picks;FlipSig(2:N/2)];
88  N_frame_IFFT=ifft(Sym_Sig,N);
89
90  if li==1
91  Reconstructed_array=[Reconstructed_array;N_frame_IFFT];
92  elseif li ≠1
93  Pad_N_frame_IFFT=vertcat(zeros(li-1,1),N_frame_IFFT); %li=33,65,97
94  Reconstructed_array=vertcat(Reconstructed_array,zeros(N/2,1));
95  Reconstructed_array=Reconstructed_array+Pad_N_frame_IFFT;
96  end
97
98  end
99
100
101 Reconstructed_audio_array_I{n}=Dom_recon_array; % capturing audio signals for method-1
102 Reconstructed_audio_array_II{n}=Reconstructed_array;% capturing audio signals for method-2
103
104
105 %SNR_New
106 [SNR_I(n),SNR_SEGMENTAL_I(n),SNR_II(n),SNR_SEGMENTAL_II(n)] = ...
        CalculateSNR(audio_signal,Reconstructed_audio_array_I{n},Reconstructed_audio_array_II{n},N);
107
108 end % end of for loop where we move n using Method-1 or Method-2 to find SNR ,audios
109
110
111
112 switch fftPoints
113 case 1 % retrive audios and SNR for 64 frame
114
115 %Retrived Audio Signals
116 T_Compressed_Audio_Signals_64_1=Reconstructed_audio_array_I;
117 T_Compressed_Audio_Signals_64_2=Reconstructed_audio_array_II;
118 % Overall SNR
119 T_SNR_64_1=SNR_I;
120 T_SNR_64_2=SNR_II;
121 % Segmental
122 T_SNR_SEGMENTAL_64_1=SNR_SEGMENTAL_I;
123 T_SNR_SEGMENTAL_64_2=SNR_SEGMENTAL_II;
124
125 case 2
126
127 %Retrived Audio Signals
128 T_Compressed_Audio_Signals_128_1=Reconstructed_audio_array_I;
129 T_Compressed_Audio_Signals_128_2=Reconstructed_audio_array_II;
```

```
% Overall SNR
T_SNR_128_1=SNR_I;
T_SNR_128_2=SNR_II;
% Segmental
T_SNR_SEGMENTAL_128_1=SNR_SEGMENTAL_I;
T_SNR_SEGMENTAL_128_2=SNR_SEGMENTAL_II;

case 3

%Retrived Audio Signals
T_Compressed_Audio_Signals_256_1=Reconstructed_audio_array_I;
T_Compressed_Audio_Signals_256_2=Reconstructed_audio_array_II;
% Overall SNR
T_SNR_256_1=SNR_I;
T_SNR_256_2=SNR_II;
% Segmental
T_SNR_SEGMENTAL_256_1=SNR_SEGMENTAL_I;
T_SNR_SEGMENTAL_256_2=SNR_SEGMENTAL_II;
end

end


% Overall SNR method 1
figure
grid on
box on
grid minor
hold on
plot(100*((1:33)/64),T_SNR_64_1,':k','LineWidth',2);
hold on
plot(100*((1:65)/128),T_SNR_128_1,'--b','LineWidth',2);
hold on
plot(100*((1:129)/256),T_SNR_256_1,':r','LineWidth',2);
hold on
title('Sliding Triangular Method-1,Overall SNR vs % of components selected ...
    (n/N)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
xlabel('% of components selected (n/N)','FontName','Times New ...
    Roman','FontSize',12,'FontWeight','bold');
ylabel('Overall SNR(dB)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
set(gca,'FontName','Times New Roman','FontSize',12,'FontWeight','bold');

legend({'N=64','N=128','N=256'},'FontName','Times New Roman','FontSize',12,'FontWeight','bold');


% Segmental SNR method 1
figure
grid on
box on
grid minor
hold on
plot(100*((1:33)/64),T_SNR_SEGMENTAL_64_1,':k','LineWidth',2);
hold on
plot(100*((1:65)/128),T_SNR_SEGMENTAL_128_1,'--b','LineWidth',2);
hold on
plot(100*((1:129)/256),T_SNR_SEGMENTAL_256_1,':r','LineWidth',2);


title('Sliding Triangular Method-1,Segmental SNR vs % of components selected ...
    (n/N)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
xlabel('% of components selected (n/N)','FontName','Times New ...
    Roman','FontSize',12,'FontWeight','bold');
ylabel('Segmental SNR(dB)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
set(gca,'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
legend({'N=64','N=128','N=256'},'FontName','Times New Roman','FontSize',12,'FontWeight','bold');

% Overall SNR method 2
figure
```

```
grid on
box on
grid minor
hold on
plot(100*((1:33)/64),T_SNR_64_2,':k','LineWidth',2);
hold on
plot(100*((1:65)/128),T_SNR_128_2,'--b','LineWidth',2);
hold on
plot(100*((1:129)/256),T_SNR_256_2,':r','LineWidth',2);


title('Sliding Triangular Method-2,Overall SNR vs % of components selected (n/N) ...
    ','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
xlabel('% of components selected (n/N)','FontName','Times New ...
    Roman','FontSize',12,'FontWeight','bold');
ylabel('Overall SNR(dB)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
set(gca,'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
legend({'N=64','N=128','N=256'},'FontName','Times New Roman','FontSize',12,'FontWeight','bold');


% Segmental SNR method 2
figure
grid on
box on
grid minor
hold on
plot(100*((1:33)/64),T_SNR_SEGMENTAL_64_2,':k','LineWidth',2);
hold on
plot(100*((1:65)/128),T_SNR_SEGMENTAL_128_2,'--b','LineWidth',2);
hold on
plot(100*((1:129)/256),T_SNR_SEGMENTAL_256_2,':r','LineWidth',2);

title('Sliding Triangular Method-2,Segmental SNR vs % of components selected ...
    (n/N)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
xlabel('% of components selected (n/N)','FontName','Times New ...
    Roman','FontSize',12,'FontWeight','bold');
ylabel('Segmental SNR(dB)','FontName','Times New Roman','FontSize',12,'FontWeight','bold');
set(gca,'FontName','Times New Roman','FontSize',12,'FontWeight','bold');
legend({'N=64','N=128','N=256'},'FontName','Times New Roman','FontSize',12,'FontWeight','bold');

% filename = 'taylor.wav';
% audiowrite(filename,Reconstructed_array,Fs);
% filename = 'Dominque.wav';
% audiowrite(filename,Dom_recon_array,Fs);
```

Matlab Code For Calculate SNR

```
% function that calculates SNR given original audio, estimated audio and framelength N=(64,128,256)
function ...
    [Overall_SNR_1,Segmental_SNR_1,Overall_SNR_2,Segmental_SNR_2]=CalculateSNR(audio_signal,estimated_audio_signa

Snr_I_Num=0;
Snr_I_Den=0;
Snr_Seg_I=0;

Snr_II_Num=0;
Snr_II_Den=0;
Snr_Seg_II=0;

L=length(audio_signal)/N;

for j=1:N:length(audio_signal) % 1:64:32768

  Xt=audio_signal(j:j+N-1); % Lth frame
  estimatedXt_1=estimated_audio_signal_1(j:j+N-1);% estimated Xt method_1
  estimatedXt_2=estimated_audio_signal_2(j:j+N-1);% estimated Xt method_2

% Method 1
et=(Xt-estimatedXt_1);% Lth frame - estimated Lth frame
Snr_I_Num=Snr_I_Num+(Xt.'*Xt);
Snr_I_Den=Snr_I_Den+(et.'*et);

Snr_Seg_I=Snr_Seg_I+(10/L)*log10((Xt.'*Xt)/(et.'*et));

%Method-2
et=(Xt-estimatedXt_2);

Snr_II_Num=Snr_II_Num+(Xt.'*Xt);
Snr_II_Den=Snr_II_Den+(et.'*et);
Snr_Seg_II=Snr_Seg_II+((10/L)*log10(((Xt).'*Xt)/(et.'*et)));

end

% Overall SNR
Overall_SNR_1=10*log10(Snr_I_Num/Snr_I_Den);
Overall_SNR_2=10*log10(Snr_II_Num/Snr_II_Den);

% Segmental SNR Dom Pick
Segmental_SNR_1=Snr_Seg_I;
% Segmental SNR Method-2
Segmental_SNR_2=Snr_Seg_II;

end
```

n	Method 1		Method 2	
	Overall	Segmental	Overall	Segmental
1	2.20	1.9391	0.0135	0.1516
2	3.87	3.484	0.0673	0.6952
...	...	...	...	...
N/2	40.71	48.175	37.52	42.82
1+(N/2)	313.99	314.6384	313.99	314.6384

Table 1: SNR for a Rectangular Window with N = 64.

n	Method 1		Method 2	
	Overall	Segmental	Overall	Segmental
1	2.391	1.387	0.0032	0.0905
2	3.7275	2.436	0.0108	0.1702
...	...	...	...	...
N/2	47.914	53.21	43.69	46.71
1+(N/2)	312.54	313.09	312.54	313.09

Table 2: SNR for a Rectangular Window with N = 128.

n	Method 1		Method 2	
	Overall	Segmental	Overall	Segmental
1	2.009	1.076	7.38e-04	0.0481
2	3.087	1.8301	0.0023	0.0893
...	...	...	...	...
N/2	54.767	61.100	50.215	53.866
1+(N/2)	311.833	312.164	311.833	312.164

Table 3: SNR for a Rectangular Window with N = 256.

n	Method 1		Method 2	
	Overall	Segmental	Overall	Segmental
1	3.17	2.41	0.0012	0.1395
2	5.17	4.15	0.0291	0.7072
...	...	...	...	...
N/2	63.92	55.80	51.69	54.58
1+(N/2)	77.31	314.83	77.31	314.83

Table 4: SNR for a Sliding Triangular Window with N = 64.

n	Method 1		Method 2	
	Overall	Segmental	Overall	Segmental
1	2.855	1.610	1.8e-05	0.0932
2	3.817	2.672	0.0034	0.1589
...	...	...	...	...
N/2	69.31	63.25	64.45	61.07
1+(N/2)	72.45	311.24	72.45	311.24

Table 5: SNR for a Sliding Triangular Window with N = 128.

n	Method 1		Method 2	
	Overall	Segmental	Overall	Segmental
1	2.54	0.9693	5.71e-05	0.0462
2	3.54	1.8631	2.79e-04	0.080
...	...	...	...	...
N/2	69.71	69.38	67.38	65.29
1+(N/2)	70.28	307	70.28	307

Table 6: SNR for a Sliding Triangular Window with N = 256.