संकुल नवप्रवर्तन केंद्र, दिल्ली विश्वविद्यालय
CLUSTER INNOVATION CENTRE, UNIVERSITY OF DELHI

Music Analysis and Compression using Signal Processing

Major Project Report

VII.1 Understanding Signals: The Mathematical and Computational Way

Mentor

Ms. Nirmal Yadav
Assistant Professor, CIC
University of Delhi

Submitted By

Rahul Yadav

ACKNOWLEDGEMENT

I would like to thank my mentor Ms. Nirmal Yadav for her constant encouragement and stimulating advice. Without her help the project would not have come to its present state. I thank her for sparing her valuable time to attend my petty doubts. I would like to extend my thanks to Ir. Abhijeet Parmar who introduced me to multimedia compression and various standards followed in multimedia at present. I would also like to thank my friends and family for their unconditional support and love.

-Author

Table of Contents

Introduction

Music is vocal or instrumental sounds or both combined in such a way as to produce beauty of form, harmony, and expression of emotion. Musical creations and performances are among the most complex and intricate of our cultural artefacts, and the emotional power of music can touch us in surprising and profound ways. In this project, I have tried to understand musical signals and how they can be processed using computational resources.

Now to understand music we must understand sound first. Sound is a vibration that propagates as a typically audible mechanical wave of pressure and displacement, through a medium such as air or water. In psychology, sound is the reception of such waves and their perception by the brain. All things that make sound move, and in some very metaphysical sense, all things that move make sound. As things move, they push and pull at the surrounding air or water or whatever medium they occupy, causing pressure variations: compressions and rarefactions. Those pressure variations, or sound waves, are what we hear as sound. Noise is a term often used to refer to an unwanted sound. Noise is generally an undesirable component that obscures a wanted signal. Sound is processed and recorded as explained in **Figure1**. The pressure variation from the source pass through the microphone and then processed in analog to digital converter.

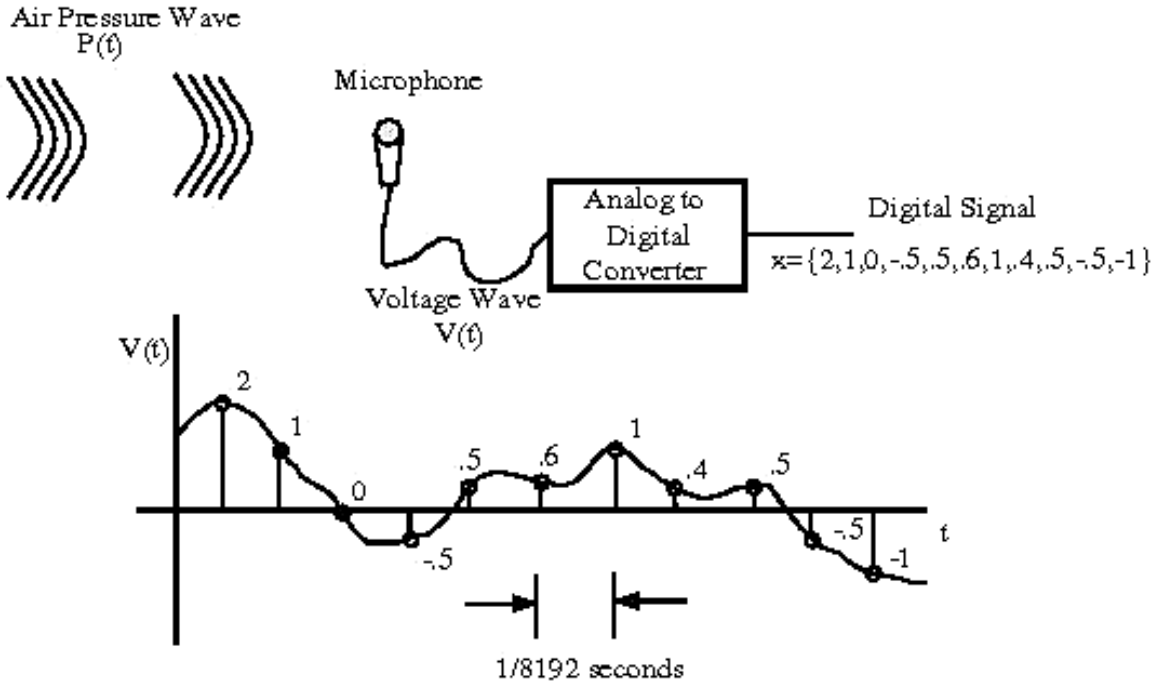

Figure1: Digital Sound Data

When we hear something it is the result of a very complicated sequence of events in our brain that was initiated by vibrations of our eardrum. The vibrations are caused by air molecules hitting the eardrum.

The picture of a sound wave as with amplitudes in time provides a nice visual for the idea of sound as a continuous sequence of pressure variations, see **Figure2** below. In signal processing this graph of pressure versus time becomes a picture of a list of numbers plotted against time. Relative signal strength denotes the amplitude of the sound waveform.

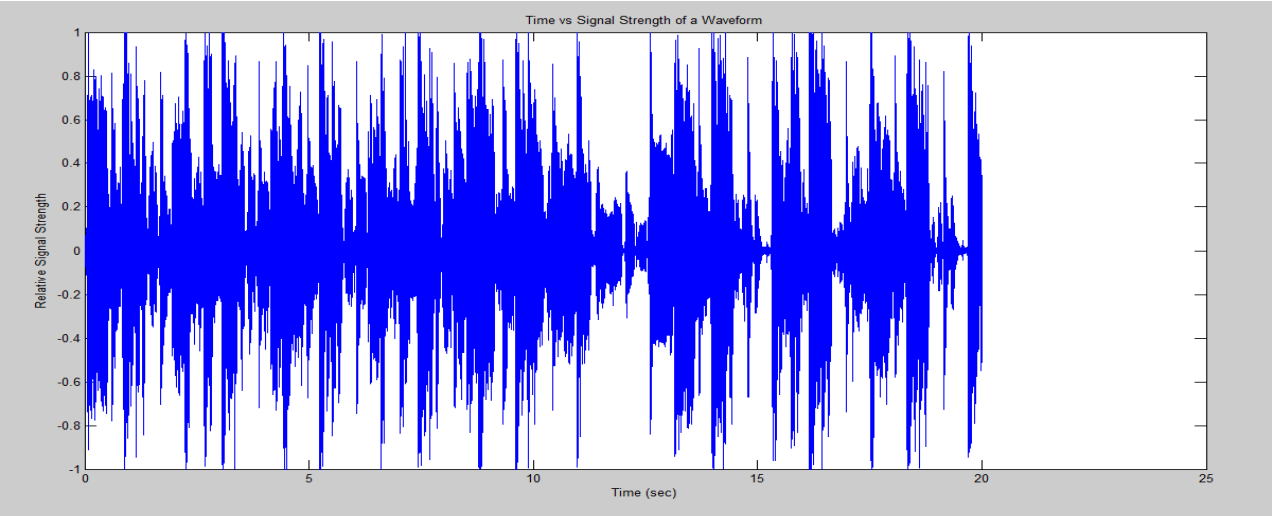

Figure2: Time vs Signal Strength of a Waveform

Digital Sound

How can we represent sound as a finite collection of numbers that can be stored efficiently in a finite amount of space, on a computer, and played back and manipulated at will? In short, how do we represent sound digitally? We have to compose a finite list of numbers that does a job of representing our waveform. This is done by sampling the original function at some sampling rate and recording the value of the function at those moments (see **Figure3**). For sound often use a special device that grabs instantaneous amplitudes at rapid audio rates is an ADC. In analog to digital conversions, continuous functions for example, air pressure are sampled and stored as numerical values. In digital to analog conversions, numerical values are interpolated by the converter to force some continuous system into a continuous vibration.

The interpolated waveforms will always be approximations as it is impossible to store truly continuous data digitally. However, by increasing the sampling rate an extremely accurate recording can be made. So, how often do we need to sample a waveform in order to achieve a good representation of it is told by the Nyquist-Shannon Sampling Theorem, which states that to well represent a signal, the sampling rate needs to be at least twice the highest frequency contained in the signal.

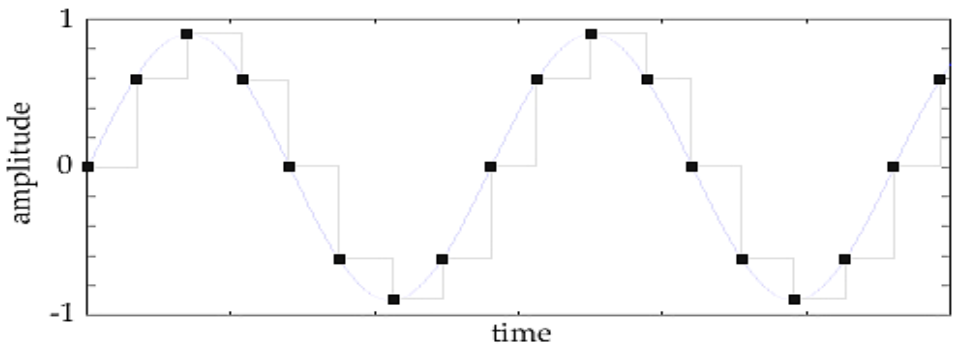

Figure3: An Analog Waveform and its Digital Representation

The hearing range of human ears is roughly 20 Hz to 20,000 Hz, and via the Nyquist–Shannon sampling theorem, the sampling rate therefore had to be greater than 40 kHz. In addition to this, signals must be low-pass filtered before sampling, otherwise aliasing occurs, and, while an ideal low-pass filter would perfectly pass frequencies below 20 kHz without attenuating them and perfectly cut off frequencies above 20 kHz, in practice a transition band is necessary, where frequencies are partly attenuated. The wider this transition band is, the easier and more economical it is to make an anti-aliasing filter. The 44.1 kHz sampling frequency allows for a 2.05 kHz transition band.

Compression

Suppose we are working with a stereo signal and 16-bit samples with sampling rate of 44.1 kHz. One 16-bit sample takes 2 bytes of storage space and in stereo, we need to double that number. For each second of sound, we will record 44,100 four-byte stereo samples, giving us a data rate of 176.4 kilobytes (176,400 bytes) per second! It is too many bits for most purposes. While it is not too wasteful if we want an hour of high-quality sound. Even though high-quality sound data are not anywhere near as large as image or video data, they are still too big to be practical.

What can we do to reduce the data explosion? If we keep in mind that we are representing sound as a kind of list of symbols, we just need to find ways to express the same information in a shorter string of symbols. That is called data compression, and it's a rapidly growing field dealing with the problems involved in moving around large quantities of bits quickly and accurately.

The goal is to store the most information in the smallest amount of space, without compromising the quality of the signal or at least, compromising it as little as possible. Compression techniques and research are not limited to digital sound- data compression plays an essential part in the storage and transmission of all types of digital information, from word-processing documents to digital photographs to full-screen, full-motion videos. As the amount of information in a medium increases, so does the importance of data compression. There are many different approaches for data compression. MP3 is the current standard for data compression of sound.

Karaoke

Karaoke is a form of interactive entertainment or video game in which amateur singers sing along with recorded music (a music video) using a microphone and public address system. The music is typically a well-known popular song minus the lead vocal. Lyrics are usually displayed on a video screen, along with a moving symbol, changing color, or music video images, to guide the singer. It is also a term used by recording engineers translated as "empty track" meaning there is no vocal track. [6]

We would stick to the definition of karaoke termed by the recording engineers. In most popular music recordings, the vocal track is the same on the left and right channels or very similar. The volume of the various instruments are more unevenly distributed between the two channels. Since the voice is the same on both channels, if we subtract one channel from the other and listen to the result, we would create a karaoke! In my experiments, some vocal can be heard as the song uses a stereo reverberation effect and the echo moves back and forth between left and right channels. This makes the voice unequal from left to right channels. But it still does a decent job.

Methodology

- Convert the music file to .wav format.
- Read the stereo music file: `[road,fs]=audioread('TarkeebeinA.wav');`

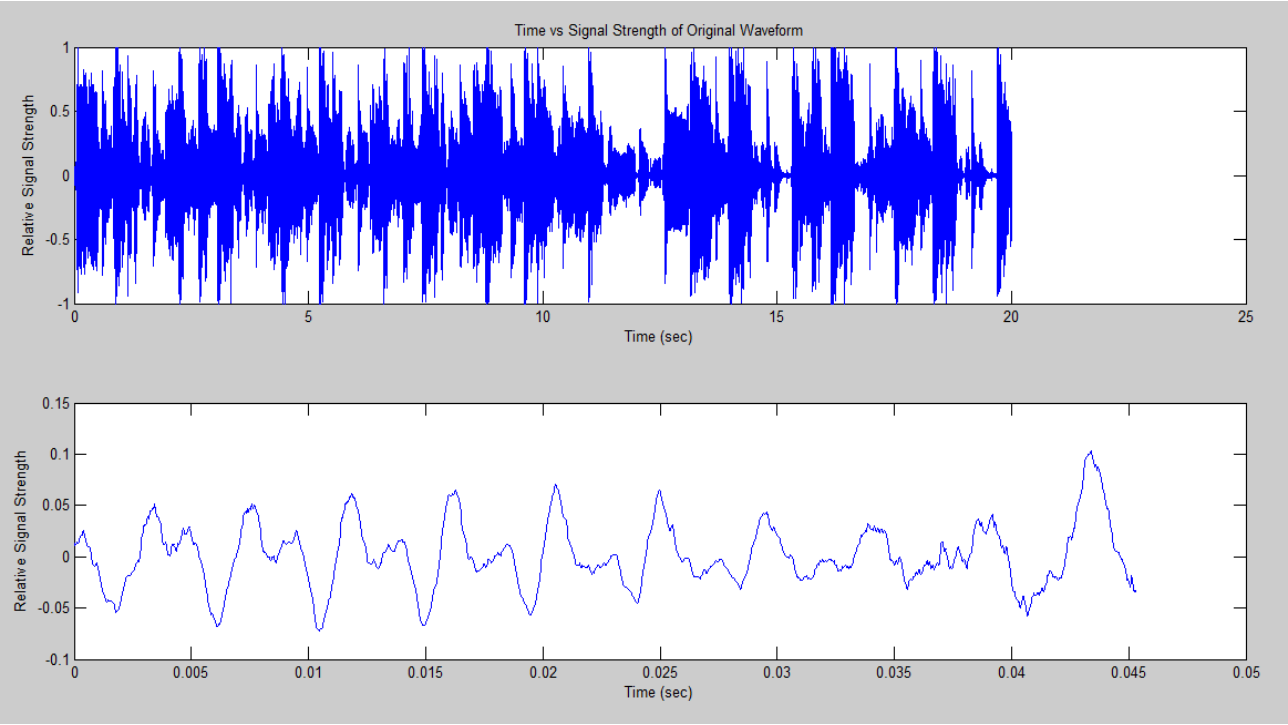

Figure4: Time vs Signal Strength of the Waveform

- Subtract right component of the stereo music file from the left component:
```
road=road(:,1)-road(:,2);
```

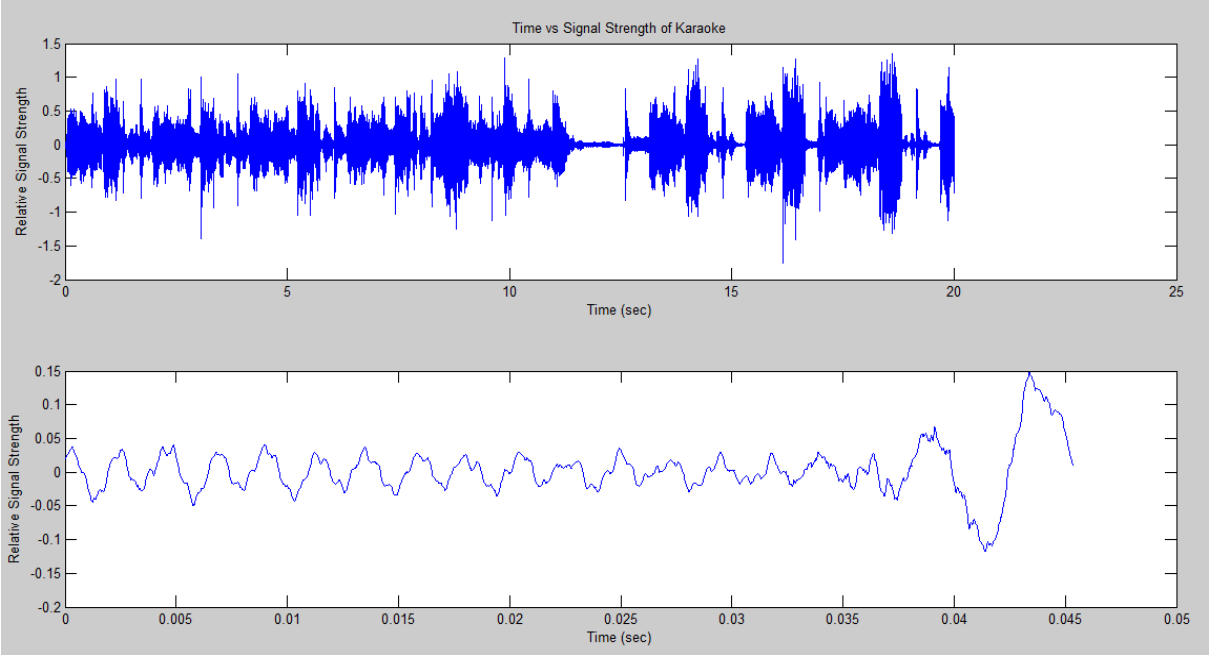

Figure5: Time vs Signal Strength of Karaoke

- Perform compression using DCT on karaoke by choosing appropriate window size:

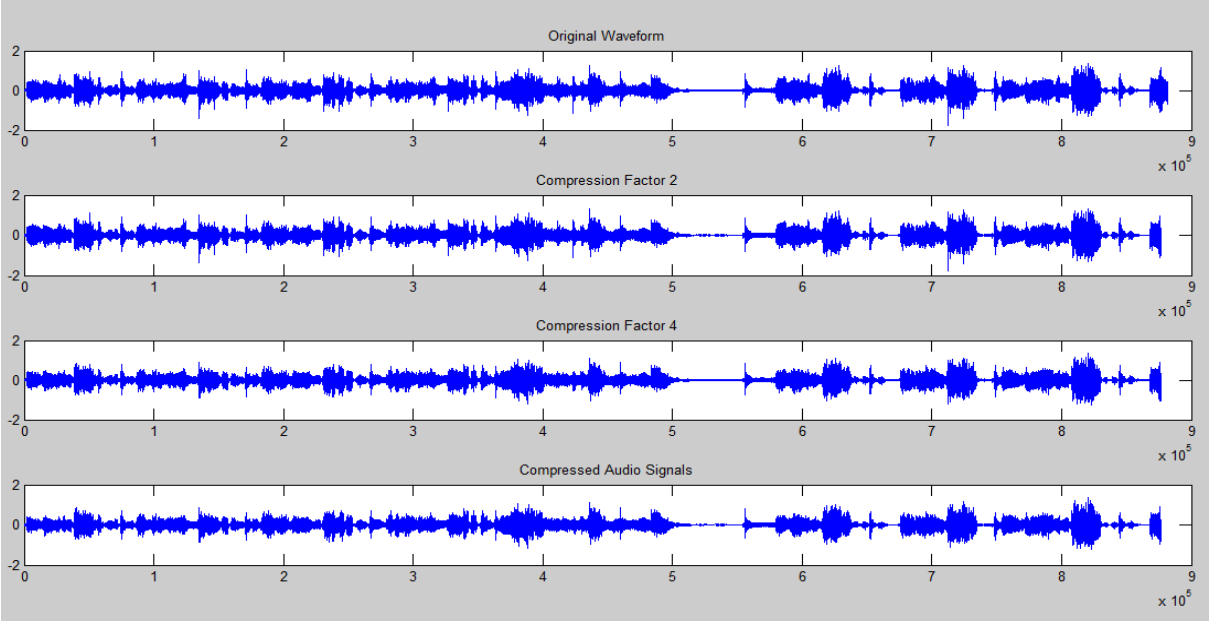

Figure6: Time vs Signal Strength of Compressed Karaoke

Spectrogram

The visual representation of the spectrum of frequencies in a sound could be done via spectrograph. `S = spectrogram(x)` returns `S`, the short time Fourier transform of the input signal vector `x`. The sound sample is divided into multiple blocks in time domain. The amplitude of `fft` for each time domain block is represented by colour and intensity in spectrogram. The x-axis is time and the y-axis is frequency. The amplitude of `fft` for each time domain block is represented by colour and intensity. The vertical axis represents frequency, the horizontal axis shows positive time toward the right, and the colours represent the most important acoustic peaks for a given time frame, with red representing the highest energies, then in decreasing order of importance, orange, yellow, green, cyan and blue areas having even less energy.

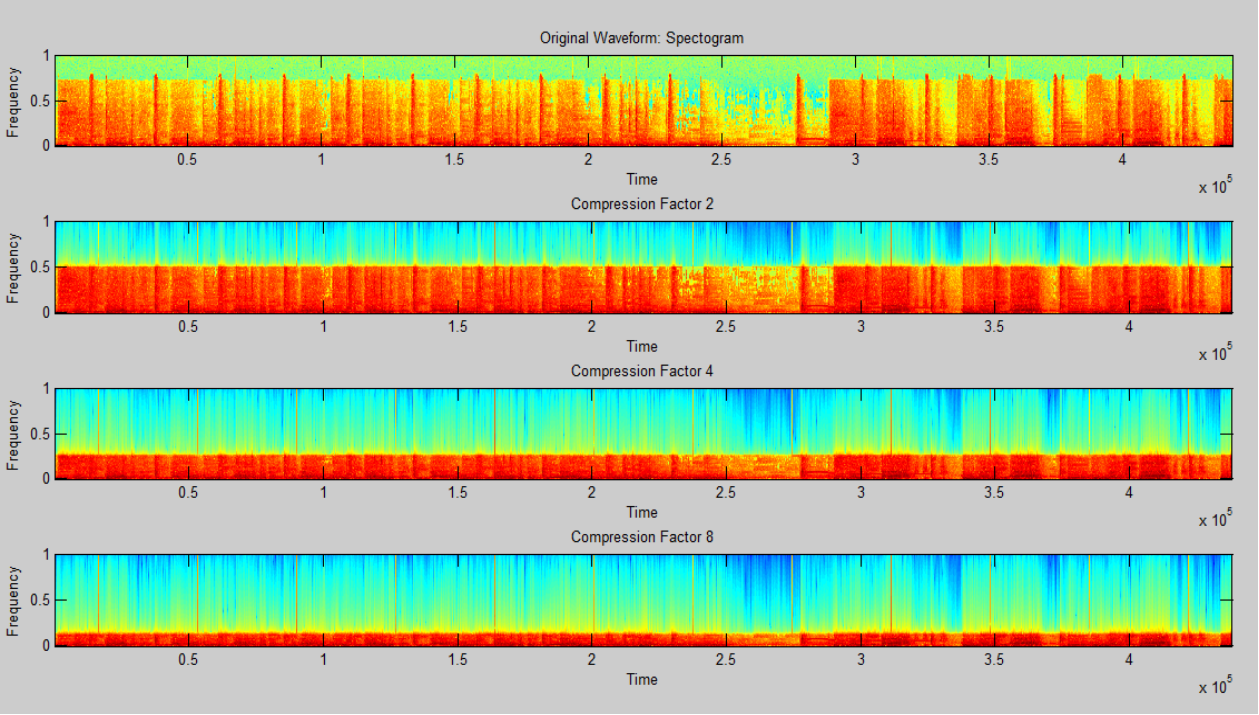

Figure7: Spectrogram of various waveforms

Conclusions

Music is carefully constructed sound signal and extracting information of relevance to listeners therefore requires various kinds of specialized methods. Karaoke, song minus lead vocal could be made better if there is low reverberation. 0.93% compression in .wav format could be achieved which is of course a very low compression, but converting it to .mp3 gives us a compression ratio of about 81.95%.

References

1. Meinard Müller, Daniel P. W. Ellis, Anssi Klapuri and Gaël Richard- "*Signal Processing for Music Analysis*", 2011.
2. Audio Processing Lecture: http://www.aquaphoenix.com/lecture/matlab10/page4.html
3. Audio Signal Processing Basics: http://www.cs.tut.fi/sgn/arg/intro/basics.html
4. Sound Processing in MATLAB: http://homepages.udayton.edu/~hardierc/ece203/sound.htm
5. MATLAB Demonstration - Basic Signal Manipulation using Audio Signals: https://www.youtube.com/watch?v=ie7iREcYBPU
6. Karaoke Wiki: http://en.wikipedia.org/wiki/Karaoke
7. Spectrogram Reading: http://www.cslu.ogi.edu/tutordemos/SpectrogramReading/spectrogram.html

Appendix

```
%% VII.1 Understanding Signals: The Mathematical and Computational Way
% Project: Music Analysis and Compression using Signal Processing
% Submitted by- Rahul Yadav (1387)
%--------------------------------------------------------------------------

%% Part1: Processing the Music Signal: Making a Karaoke [MP3 Song- Lead
Vocal]
clc;
clear;
[road,fs]=audioread('TarkeebeinA.wav');
%road=road(44100*1:44100*20); % Segment of the audio file (Range)
%sound(road,fs) % Play audio file
size(road) %Size of song
left=road(:,1);
right=road(:,2); %Left and right channel signals are the two columns of the
road array
%--------------------------------------------------------------------------
%% Plot of data vs time. This plot will show where the signal is strong and
weak over time.
figure(1)
h1=subplot(2,1,1);
time=(1/44100)*length(left);
t=linspace(0,time,length(left));
plot(t,left)
xlabel('Time (sec)');
ylabel('Relative Signal Strength')
title('Time vs Signal Strength of Original Waveform')

% Plot of a small portion to do detailed analysis
subplot(2,1,2);
time=(1/44100)*2000;
t=linspace(0,time,2000);
plot(t,left(1:2000))
xlabel('Time (sec)');
ylabel('Relative Signal Strength')
% --------------------------------------------------------------------------
%% To listen audio with modifications
sound(left,fs)      %Left Channel
sound(right,fs)     %Right Channel
sound(road,fs)      %Stereo

%Reverse Playing: To play the sound backwards, we simply reverse the order of
the numbers in the arrays.
left2=flipud(left);   %flipud stands for flip upside-down which flips your
array y and stores the inverted array in a new array called y2
sound(left2,fs)
right2=flipud(left2); %Reversed Again [Original]
sound(right2,fs)

% Changing the Speed The sampling frequency fs tells us how much time goes
between each sample (T=1/fs).
% If we play the song with more or less time between samples than was
originally there when recorded,
% the speed produces interesting effects:
```

```
sound(road,fs/1.5) %Slow Music
sound(road,fs*1.5) %Fast Music- Remix!
audiowrite('TarkeebeinF.wav',road,fs*1.5);
audiowrite('TarkeebeinS.wav',road,fs/1.5);
%--------------------------------------------------------------------------
%% Removing or minimizing lead vocals

% In most popular music recordings, the vocal track is the same on the left
and right channels (or very similar). The volume of the various instruments
are more unevenly distributed between the two channels.  Since the voice is
the same on both channels, what would happen if we subtract one channel from
the other and listen to the result?

sound(left,fs);        % Original left channel
sound(left-right,fs);  % Virtually no vocal. Voice is eliminated.
sound(road(:,1)-road(:,2),fs); % Reduced vocal
% Some vocal can be heard as the song uses a stereo reverberation effect and
the echo moves back and forth between left and right channels. This makes the
voice unequal from left to right channels.
%%-------------------------------------------------------------------------

%% Part2: Compressing the music signal
road=road(:,1)-road(:,2);
left=road(:,1);
%choosing a block size
windowSize = 8192;

%changing compression  percentages
samplesHalf = windowSize / 2;
samplesQuarter = windowSize / 4;
samplesEighth = windowSize / 8;

%initializing compressed matrix
roadCompressed2 = [];
roadCompressed4 = [];
roadCompressed8 = [];

%actual compression
for i=1:windowSize:length(road)-windowSize
    windowDCT = dct(road(i:i+windowSize-1));
    roadCompressed2(i:i+windowSize-1) = idct(windowDCT(1:samplesHalf),
windowSize);
    roadCompressed4(i:i+windowSize-1) = idct(windowDCT(1:samplesQuarter),
windowSize);
    roadCompressed8(i:i+windowSize-1) = idct(windowDCT(1:samplesEighth),
windowSize);
end

%% plotting audio signals
figure(2)
h1 = subplot(4,1,1);
plot(road), title('Original Waveform');
subplot(4,1,2)
plot(roadCompressed2), title('Compression Factor 2'), axis(axis(h1));
subplot(4,1,3)
```

```
plot(roadCompressed4), title('Compression Factor 4'), axis(axis(h1));
subplot(4,1,4)
plot(roadCompressed8), title('Compression Factor 8'), axis(axis(h1));
title('Compressed Audio Signals')

%expanded view of audio signals
figure(3)
h1 = subplot(4,1,1); plot(road(100000:120000)), title('Portion of Original
Waveform');
subplot(4,1,2)
plot(roadCompressed2(100000:120000)), title('Portion of Compression Factor
2'),
subplot(4,1,3)
plot(roadCompressed4(100000:120000)), title('Portion of Compression Factor
4'), axis(axis(h1));
subplot(4,1,4)
plot(roadCompressed8(100000:120000)), title('Portion of Compression Factor
8'), axis(axis(h1));
title('Expanded View of compressed Audio Signals')

%spectrogram of audio signals
figure(4)
subplot(4,1,1)
spectrogram(left), title('Original Waveform');
subplot(4,1,2)
spectrogram(roadCompressed2), title('Compression Factor 2');
subplot(4,1,3)
spectrogram(roadCompressed4), title('Compression Factor 4');
subplot(4,1,4)
spectrogram(roadCompressed8), title('Compression Factor 8');
title('Spectrogram of audio signals')

%% Saving compressed files as audio files
audiowrite('TarkeebeinK.m4a',road,fs);
audiowrite('TarkeebeinK2.wav',roadCompressed2,fs);
audiowrite('TarkeebeinK4.wav',roadCompressed4,fs);
audiowrite('TarkeebeinK8.wav',roadCompressed8,fs);

% End of the program-------------------------------------------------------
```