****

**Shahjalal University of Science and Technology**

**Akhalia, Sylhet**

**Assignment**

**Course Title:** Speech Processing

**Course Code:** CSE 517

**Submitted By:**

Amit Lal Das

M.Sc in CSE

1st Year 2nd Semester

Reg. No: 2021321001

**Submitted To:**

### Associate Professor Dr. Sadia Sultana

Department: CSE

SUST

Date of Submission: 24/02/2022

**Assignment 1**

**Title:** Load a speech signal (.wav file) and find out the sampling frequency. Plot the simple waveform, Linear Magnitude Spectrum, and Log Magnitude Spectrum of the signal.

**Code:**

import matplotlib.pyplot as plt

import numpy as np

import scipy.io.wavfile as wavfile

# Load the speech signal from a .wav file

filename = "abc.wav"

sampling\_freq, signal = wavfile.read(filename)

# Find the length of the signal in seconds

signal\_length = len(signal) / sampling\_freq

# Create a time vector for the signal

time = np.linspace(0, signal\_length, len(signal))

# Plot the waveform

plt.subplot(3, 1, 1)

plt.plot(time, signal)

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.title("Waveform")

sampling\_freq, signal\_data = wavfile.read(filename)

# Compute the linear magnitude spectrum using the FFT

magnitude\_spectrum = np.abs(np.fft.fft(signal\_data))

# Compute the corresponding frequency vector

freq\_vector = np.fft.fftfreq(len(signal\_data), 1/sampling\_freq)

# Plot the linear magnitude spectrum

plt.plot(freq\_vector[:len(freq\_vector)//2], magnitude\_spectrum[:len(magnitude\_spectrum)//2])

plt.xlabel("Frequency (Hz)")

plt.ylabel("Magnitude")

plt.title("Linear Magnitude Spectrum")

# Show the plot

plt.show()

sampling\_freq, signal\_data = wavfile.read(filename)

# Compute the linear magnitude spectrum using the FFT

magnitude\_spectrum = np.abs(np.fft.fft(signal\_data))

# Compute the corresponding frequency vector

freq\_vector = np.fft.fftfreq(len(signal\_data), 1/sampling\_freq)

# Convert the magnitude spectrum to decibels (dB)

log\_magnitude\_spectrum = 20\*np.log10(magnitude\_spectrum)

# Plot the log magnitude spectrum

plt.plot(freq\_vector[:len(freq\_vector)//2], log\_magnitude\_spectrum[:len(log\_magnitude\_spectrum)//2])

plt.xlabel("Frequency (Hz)")

plt.ylabel("Magnitude (dB)")

plt.title("Log Magnitude Spectrum")

# Show the plot

plt.show()

**Result:**

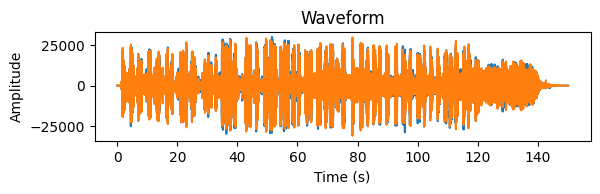
****

Fig-1: Waveform

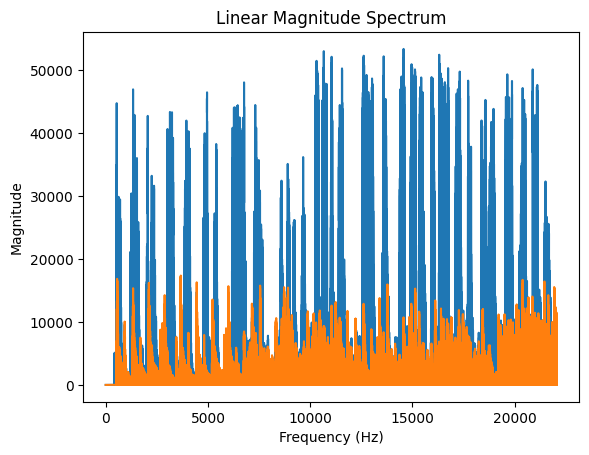


Fig-2: Linear magnitude spectrum

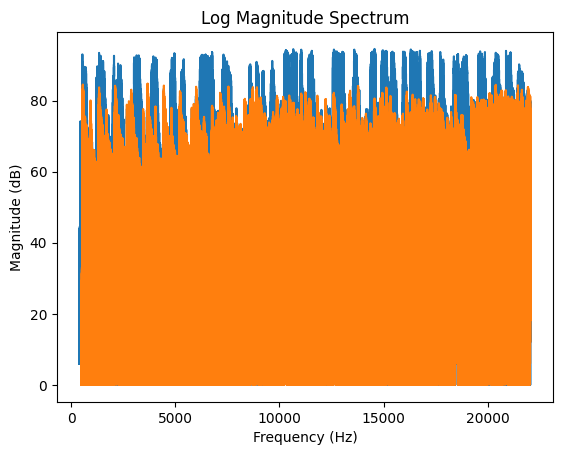


Fig-2: log magnitude spectrum

**Assignment 2**

**Title:** Applying framing of 25 ms with an overlapping of 10 ms. Apply hamming window on each frame. Reconstruct the signal and draw the plot.

**Code:**

import scipy.io.wavfile as wavfile

import scipy.signal as signal

import numpy as np

import matplotlib.pyplot as plt

# Load the audio signal from a .wav file

filename = "abc.wav"

sampling\_freq, signal\_data = wavfile.read(filename)

# Define the window length and overlap in samples

window\_length = int(0.025 \* sampling\_freq)

overlap = int(0.01 \* sampling\_freq)

# Segment the signal into frames with the given window length and overlap

num\_frames = int(np.ceil(len(signal\_data) / overlap))

pad\_length = num\_frames \* overlap - len(signal\_data)

padded\_signal = np.append(signal\_data, np.zeros(pad\_length))

indices = np.tile(np.arange(0, window\_length), (num\_frames, 1)) + np.tile(np.arange(0, num\_frames \* overlap, overlap), (window\_length, 1)).T

frames = padded\_signal[indices.astype(np.int32, copy=False)]

# Apply a Hamming window to each frame

hamming\_window = signal.hamming(window\_length)

hamming\_frames = frames \* hamming\_window

# Perform the overlap-add reconstruction

reconstructed\_signal = np.zeros\_like(padded\_signal)

for i in range(num\_frames):

start = i \* overlap

end = start + window\_length

reconstructed\_signal[start:end] += signal.fftconvolve(hamming\_frames[i], hamming\_window, mode="same")

# Trim the padding from the reconstructed signal

reconstructed\_signal = reconstructed\_signal[:len(signal\_data)]

# Plot the original and reconstructed signals

time\_vector = np.arange(len(signal\_data)) / sampling\_freq

plt.plot(time\_vector, signal\_data, label="Original Signal")

plt.plot(time\_vector, reconstructed\_signal, label="Reconstructed Signal")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.title("Original vs. Reconstructed Signal")

plt.legend()

# Show the plot

plt.show()

**Result:**

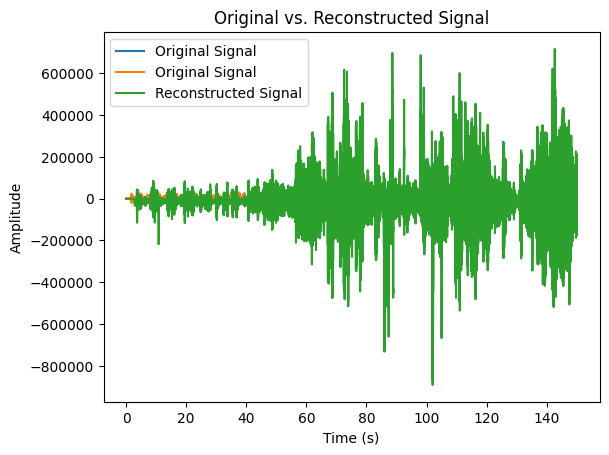
****

Fig-4: Original vs Reconstructed Signal

**Assignment 3**

**Title:** Apply FFT on a speech signal and draw the frequency spectrum.

**Code:**

import scipy.io.wavfile as wavfile

import numpy as np

import matplotlib.pyplot as plt

# Load the audio signal from a .wav file

filename = "abc.wav"

sampling\_freq, signal\_data = wavfile.read(filename)

# Compute the one-dimensional FFT of the signal

signal\_fft = np.fft.fft(signal\_data)

# Compute the two-sided spectrum

magnitude\_spectrum = np.abs(signal\_fft)

frequency\_axis = np.fft.fftfreq(len(signal\_data), 1/sampling\_freq)

frequency\_mask = frequency\_axis >= 0

# Plot the frequency spectrum

plt.plot(frequency\_axis[frequency\_mask], magnitude\_spectrum[frequency\_mask])

plt.xlabel("Frequency (Hz)")

plt.ylabel("Magnitude")

plt.title("Frequency Spectrum")

# Show the plot

plt.show()

**Result:**

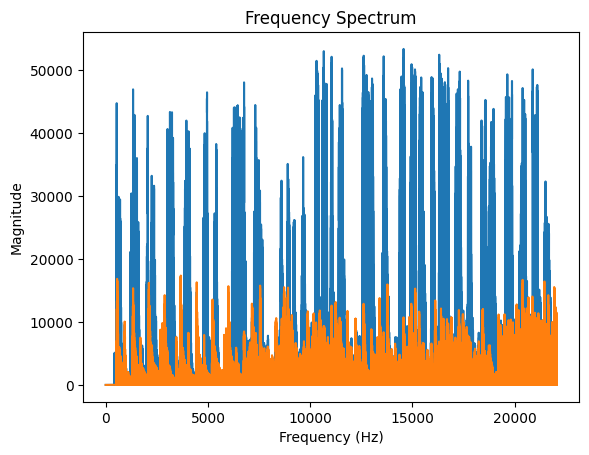
****

Fig-5: Frequency Spectrum

**Assignment 4**

**Title:** Find out the Pitch (F0) and formants F1, F2, and F3 from a speech signal.

**Code:**

**Result:**

**Assignment 5**

**Title:** Calculate the short-term energy, zero-crossing rate, and MFCCs(Including all steps to calculate MFCC : Theoretical explanation) of a speech signal.

**Code:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

from scipy.signal import stft, hamming

from python\_speech\_features import mfcc

# Load the speech signal

sample\_rate, signal = wavfile.read('abc.wav')

# Set the parameters

frame\_length = int(25 \* sample\_rate / 1000) # Frame length in samples

frame\_step = int(10 \* sample\_rate / 1000) # Frame step in samples

nfft = 512 # FFT size

num\_ceps = 12 # Number of MFCC coefficients to compute

# Calculate the short-term energy and zero-crossing rate

frames = np.array([signal[i:i+frame\_length] for i in range(0, len(signal)-frame\_length, frame\_step)])

energy = np.sum(frames\*\*2, axis=1)

zc\_rate = np.sum(np.abs(np.diff(np.sign(frames))), axis=1) / (2 \* frame\_length)

# Calculate the MFCCs

mfccs = mfcc(signal, sample\_rate, winlen=0.025, winstep=0.01, numcep=num\_ceps, nfilt=26, nfft=nfft, preemph=0.97)

# Plot the results

plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1)

plt.plot(signal)

plt.ylabel('Amplitude')

plt.title('Speech Signal')

plt.subplot(3, 1, 2)

plt.plot(energy)

plt.ylabel('Energy')

plt.title('Short-term Energy')

plt.subplot(3, 1, 3)

plt.plot(zc\_rate)

plt.ylabel('ZCR')

plt.title('Zero Crossing Rate')

plt.xlabel('Frame Index')

plt.tight\_layout()

plt.show()

plt.figure(figsize=(10, 6))

plt.imshow(np.abs(stft(signal, nperseg=frame\_length, noverlap=frame\_step, window=hamming(frame\_length), nfft=nfft)), aspect='auto', origin='lower', cmap='inferno')

plt.xlabel('Time (s)')

plt.ylabel('Frequency (Hz)')

plt.title('STFT')

plt.colorbar()

plt.show()

plt.figure(figsize=(10, 6))

plt.imshow(mfccs.T, aspect='auto', origin='lower', cmap='inferno')

plt.xlabel('Frame Index')

plt.ylabel('MFCC Coefficient Index')

plt.title('MFCCs')

plt.colorbar()

plt.show()

**Result:**

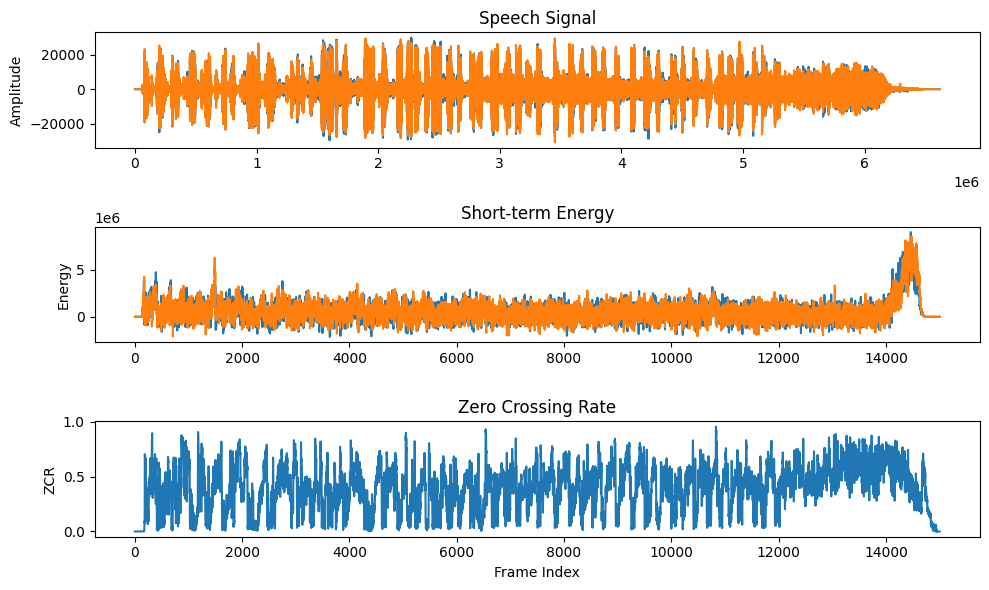
****

Fig-6: Speech signal, short term energy, zero crossing rate