Introduction

Due to the growth of the internet and the social media, people are more connected than ever before. Any message can be shared across the globe within seconds. This ease and flexibility have also come with its negative outcomes related to the accuracy of the message shared. Today fake messages can be easily spread among people and can cause chaos. One such approach taken is **splicing** of audio signals. Splicing plays an important role in audio forensics**. Here any audio signals can be taken and cut into several sub signals**. These sub signals can be arranged in a different order **concatenated to give a different meaning**. For instance, the audio signal of any important personalities or celebrities can be taken, spliced and produce new fake signals. Detecting whether an audio signal is spliced or not is an important feature in determining the credibility of a signal. In this project a spliced signal is processed and analyzed on how many occasions it has been spliced using MATLAB.

Objective

To understand various signal processing tools and techniques in MATLAB and use that to ensure the credibility of a given audio signal.

The output of the project is in the form of a report having a minimum of 500 words that contains at least the following information:

1- A figure obtained using MATLAB showing the spectrogram of the audio signal along with discussion.

Discussion

Spectrogram represents the spectrum of frequencies over time. That is as the figure gets brighter in color the more frequent those frequencies are in that region and vice versa. Therefore, the color intensity level in the spectrogram gives us the energy content of the signal in frequency component thus making it a powerful tool in audio analysis. The spectrogram is plotted as follows.

1. The audio is split into overlapping chunks or windows.

   In our example we have split the signal into 1024 windows.

2. Short time Fourier transformation is performed in each window.
3. Each resulting window gives a vertical line which is magnitude vs frequency.
4. The resulting window is converted to decibels
5. Laying out the windows into length of the original audio clip.

CODE

```
clc
clear all
```

```
%Reads the data from the .wav file and returns sampled
data,AudioSemesterProject, and a sample rate, Fs
[AudioSemesterProject, Fs] =
audioread('AudioSemesterProject.wav');

%% Spectogram plotting of original audio
AudioSemesterProject = AudioSemesterProject(:, 1);
N =length(AudioSemesterProject);
t=(0:N-1)/Fs;

%plot time domain signal
figure(1);
plot(t, AudioSemesterProject)
grid on
xlabel('Time (s)')
ylabel('Amplitude')
title('Signal in time domain')


%plot spectrogram of the signal
window = 1024;  % divide the signal into segments and
perform windowing
noverlap = 512;   % samples of overlap between adjoining
segments
nfft = 1024;   % sampling points to calculate the
discrete Fourier transform
figure(2);
spectrogram(AudioSemesterProject, window, noverlap, nfft,
Fs, 'yaxis')
title('Spectrogram of the Audio Signal');
```

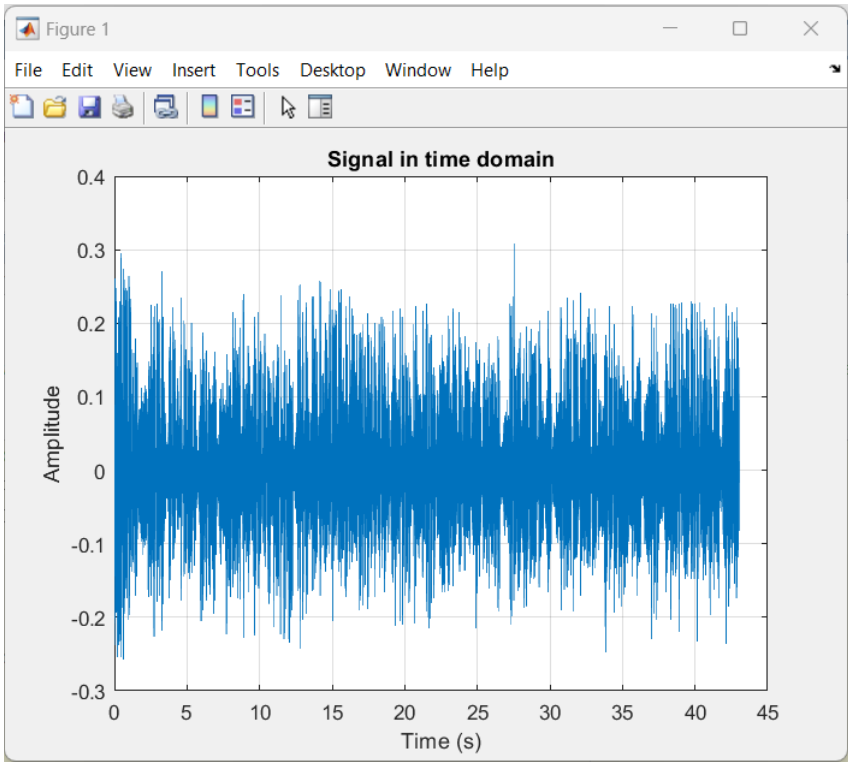
NOTE:

Here in our spectrogram plot

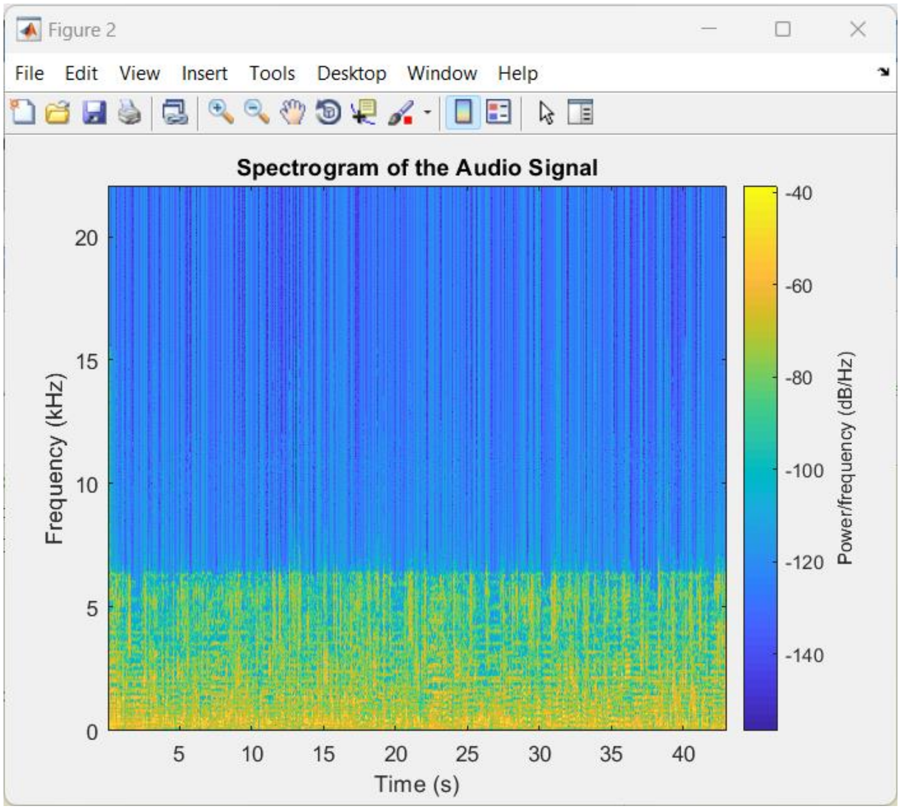
- The signal is divided into 1024 components(windows) and used the window type as hamming window to avoid side lobes of the signals.
- We have also used 512 distinct overlaps between our sample windows for more intensity.
- 1024 distinct points were used to calculate the Fourier transform.
- We have used the Fs as our sampling rate.

OUTPUT

Original Signal:



Spectrogram:



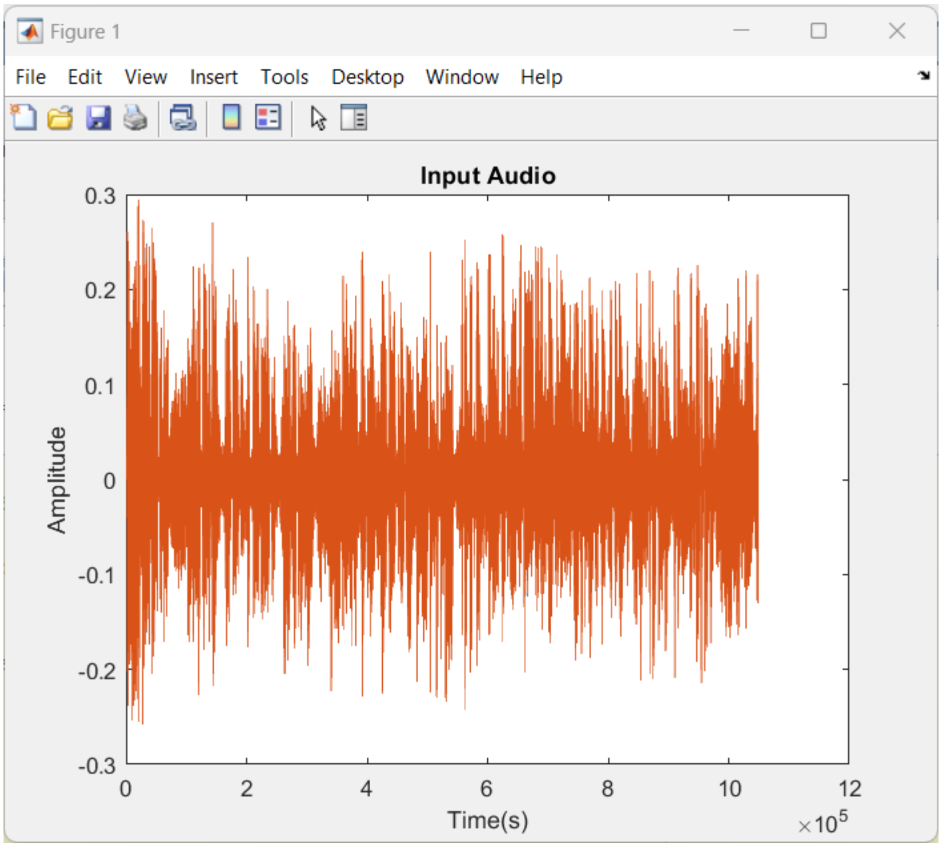
2- **MATLAB code that is used to obtain the time locations of the audios being spliced.**

NOISE CANCELLATION APPROACH

```
clc
clear all
[AudioSemesterProject,fs]=audioread('AudioSemesterProject
.wav');

%% Play original file
% pin = audioplayer(AudioSemesterProject,fs);
% pin.play;
b=AudioSemesterProject([1:1048576],:);
audiolen=length(b);
df=fs/audiolen;
audiofrequency=-fs/2:df:fs/2-df;

figure
plot(b)      % Input audio plot
title('Input Audio');
xlabel('Time(s)');
ylabel('Amplitude');

%% FFT of original file
FFT=fftshift(fft(b))/length(fft(b));
f4=FFT;
figure
plot(audiofrequency,abs(FFT));
title('FFT of Input Audio');
xlabel('Frequency(Hz)');
ylabel('Amplitude');

% Initializing zero matrix of same size as that of
original matrix

f2=zeros(1048576,2);    % f2 matrix is for background
music
f3=zeros(1048576,2);    % f3 matrix is for human voice

for i=1:1048576
    for j=1:2
       if  (i<=400000 || i>=648576)
        f2(i,j)=FFT(i,j);
      end
      if ((i>=472288&&i<=514288))||(i>=534288&&i<=576288)
        f3(i,j)=FFT(i,j);
      end
    end
end

%% FFT of Noise cancelled audio

fsample=(f3);
len1=length(fsample);
s=(ifft(ifftshift((fsample)*length(b))));
fs=44100;
de=fs/len1;
farray=-fs/2:de:fs/2-de;

figure
plot(farray,abs(fsample))
title('FFT of Noise Cancelled Audio');
xlabel('Frequency(Hz)');
ylabel('Amplitude');

noiselessout=(s+transpose(ctranspose(s)))*0.5;
audiowrite('NoiselessAudio.wav',noiselessout,fs);
[NoiselessAudio,fs]=audioread('NoiselessAudio.wav');
% p1 = audioplayer(NoiselessAudio,fs);
% p1.play;

figure
plot(noiselessout);     % Output plot for Noise Cancelled
Audio
title('Noise Cancelled Audio');
xlabel('time');
ylabel('Amplitude')
%% FFT of Background noise
fsample=(f2);
len1=length(fsample);
s=(ifft(ifftshift((fsample)*length(b))));
fs=44100;
de=fs/len1;
farray=-fs/2:de:fs/2-de;

figure
plot(farray,abs(fsample))
title('FFT of Background Noise');
xlabel('Frequency(Hz)');
ylabel('Amplitude');

backgroundout=(s+transpose(ctranspose(s)))*0.5;
audiowrite('BackgroundNoise.wav',backgroundout,fs);
[BackgroundNoise,fs]=audioread('BackgroundNoise.wav');
```

```
p2 = audioplayer(BackgroundNoise,fs);
p2.play;

figure
plot(backgroundout);      % Output plot for background
noise
title('Background Noise');
xlabel('time');
ylabel('Amplitude');
```

OUTPUTS



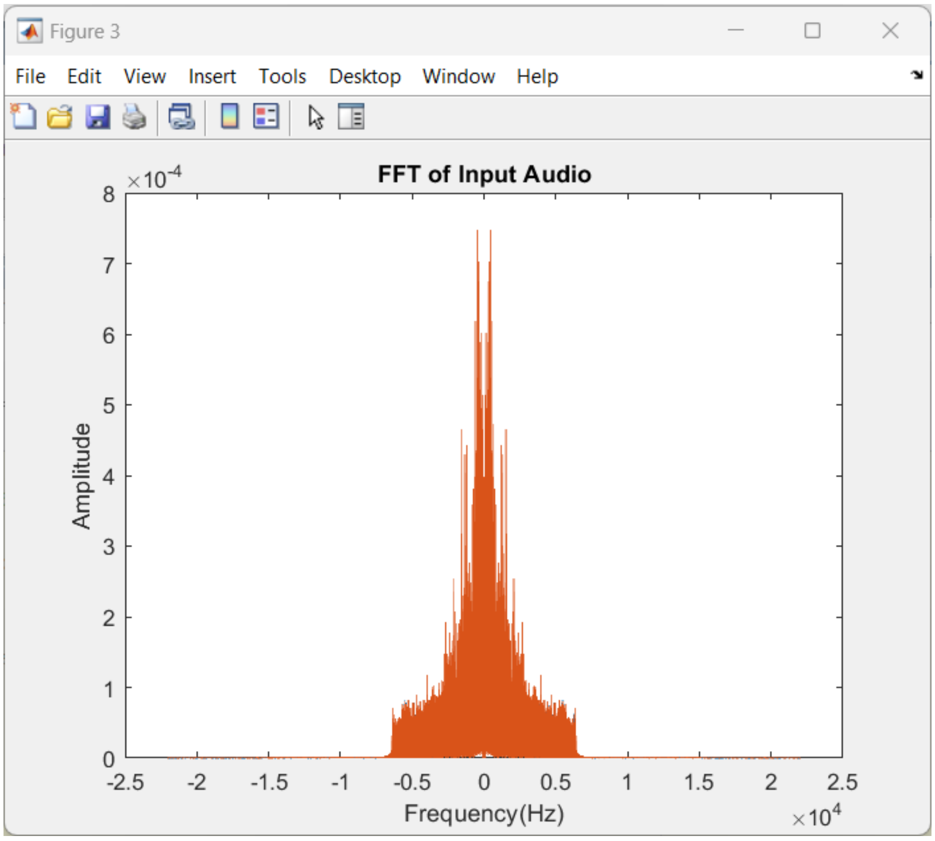


Figure 3
File
Edit
View
Insert
Tools
Desktop
Window
Help
FFT of Input Audio
×10^-4
Amplitude
Frequency(Hz)
×10^4
0
1
2
3
4
5
6
7
8
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5

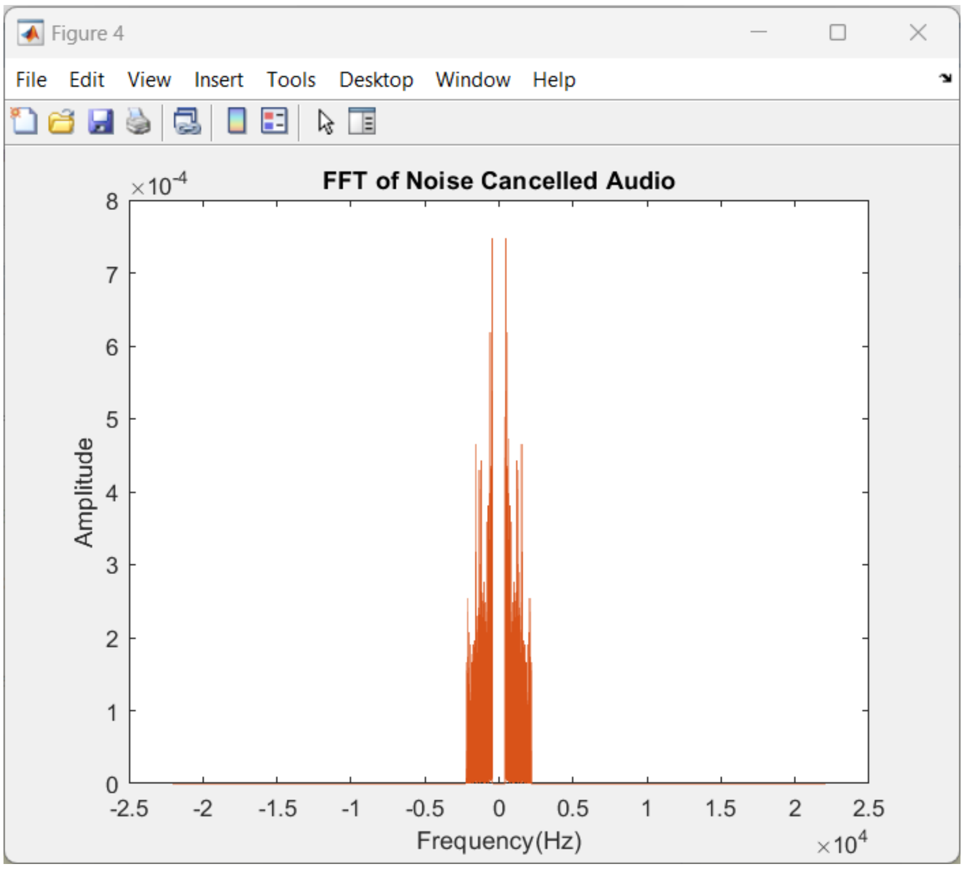


Figure 4
File
Edit
View
Insert
Tools
Desktop
Window
Help
FFT of Noise Cancelled Audio
×10^-4
Amplitude
Frequency(Hz)
×10^4
0
1
2
3
4
5
6
7
8
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5

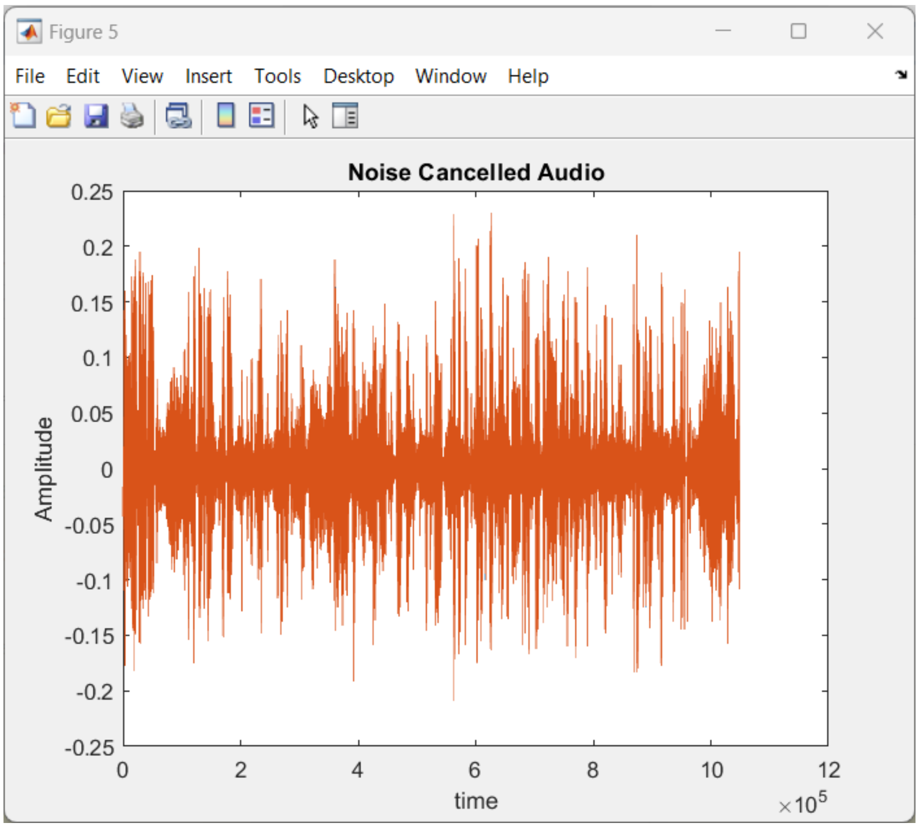


Figure 5
File Edit View Insert Tools Desktop Window Help
Noise Cancelled Audio
Amplitude
time
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0
2
4
6
8
10
12
$\times 10^5$

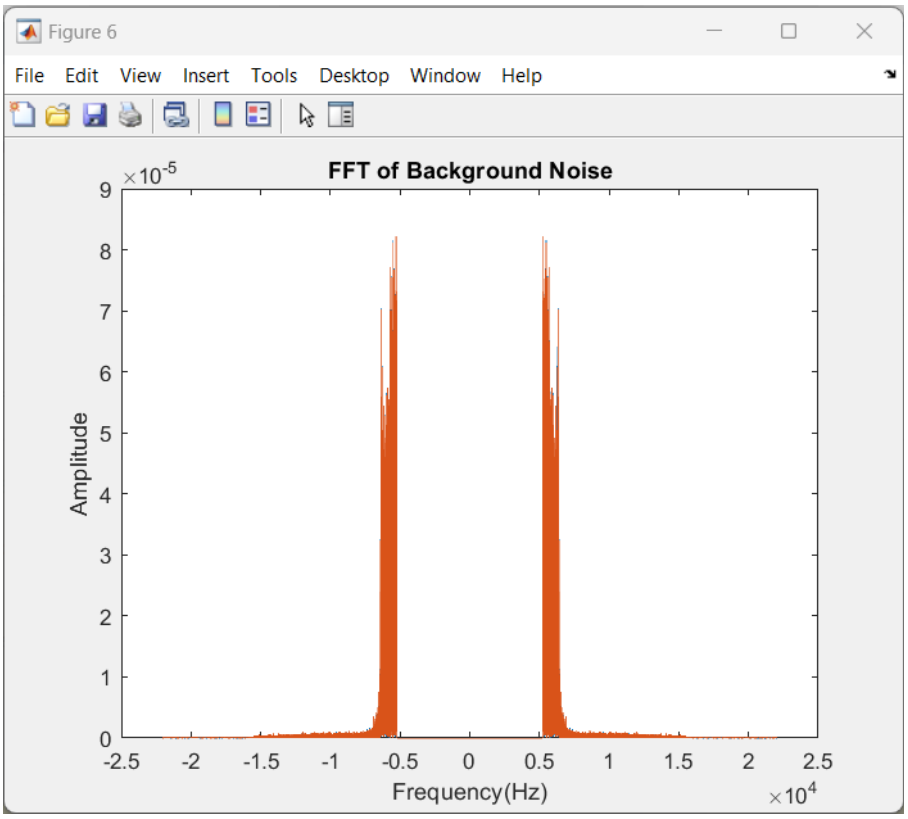
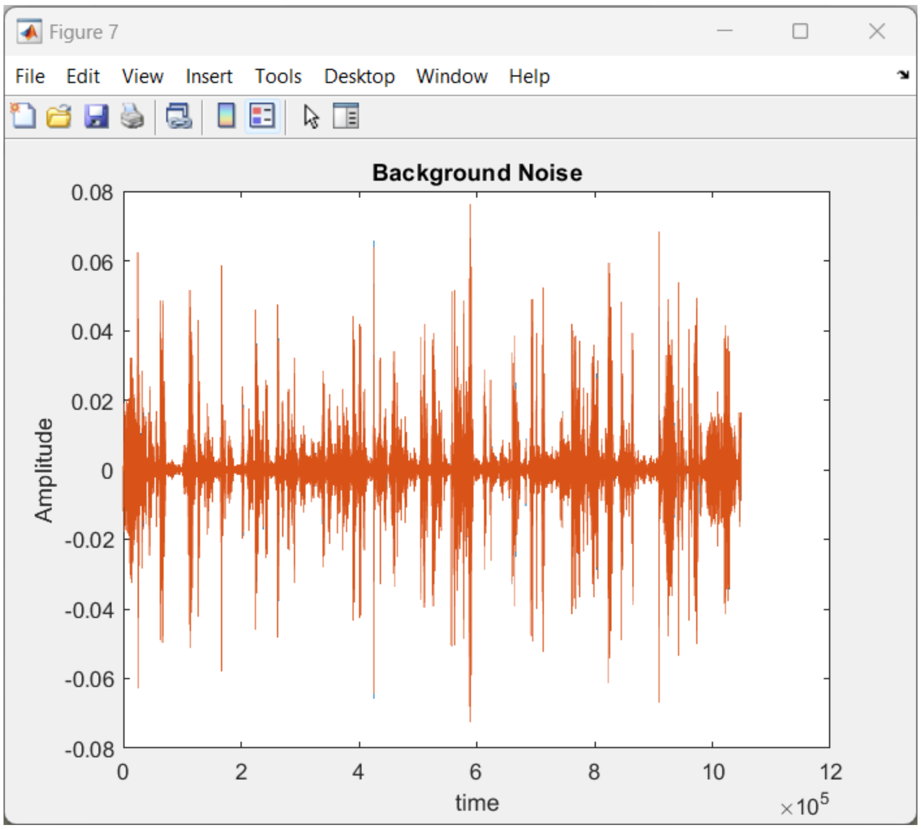
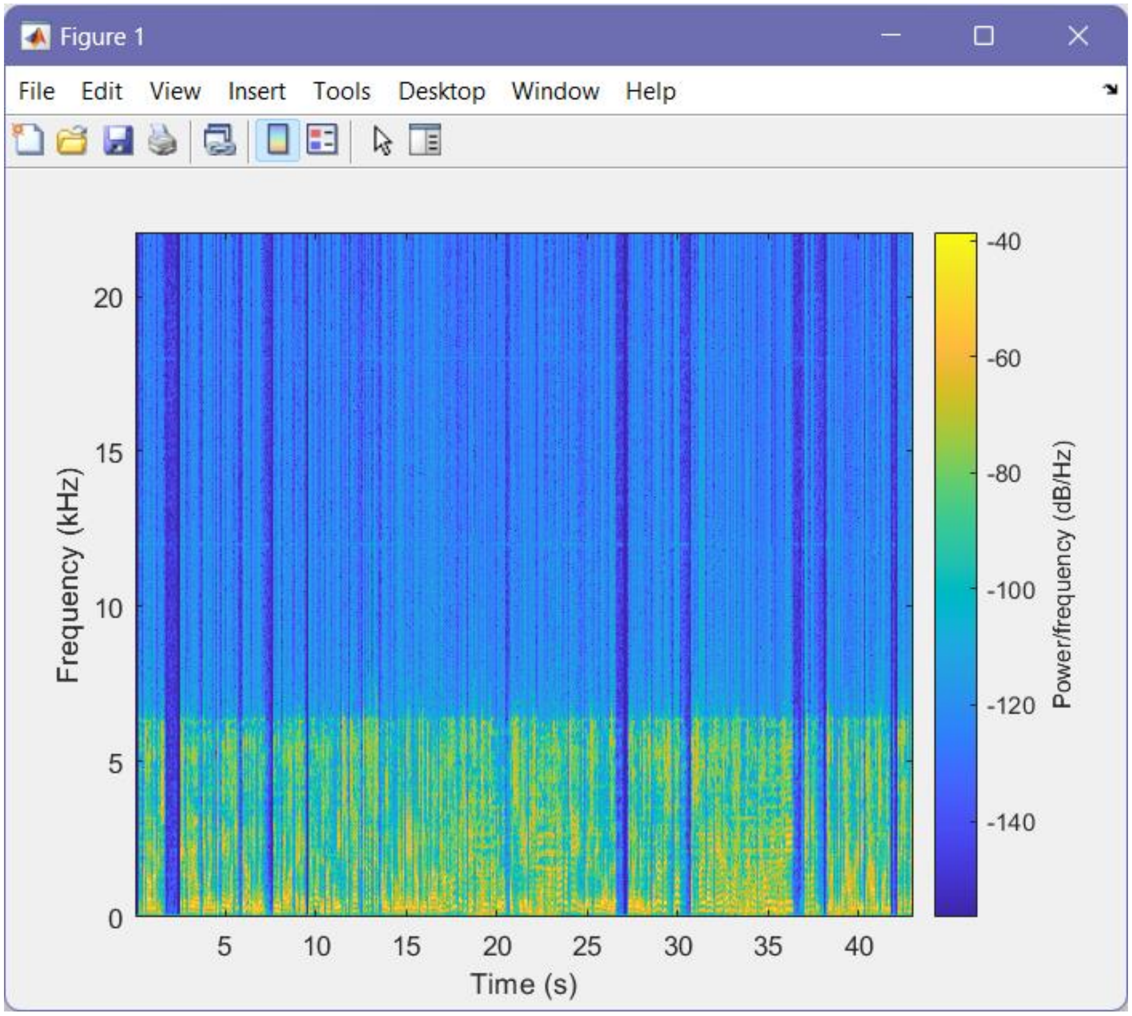


Figure 6
File Edit View Insert Tools Desktop Window Help
FFT of Background Noise
$\times 10^{-5}$
Amplitude
Frequency(Hz)
9
8
7
6
5
4
3
2
1
0
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
$\times 10^4$



Now we plot the spectrogram of the noiseless audio signal as follows.

```
clc
clear all
[NoiselessAudio, fs] = audioread ("NoiselessAudio.wav");

%% Spectogram plotting of noise cancelled new audio
NoiselessAudio = NoiselessAudio(:,1);
N = length (NoiselessAudio);
t = (0: N-1) /fs;

maxValue=max(NoiselessAudio)
minValue=min(NoiselessAudio)
meanValue=mean(NoiselessAudio)
stdValue=std(NoiselessAudio)

spectrogram (NoiselessAudio,1024,512,1024, fs, 'yaxis')
title('Spectrogram of Noise Cancelled New Audio');
```

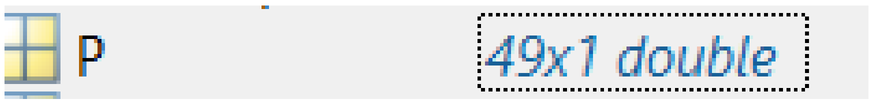


Next, we find the number of splicing points using the above spectrogram. For this we used the MATLAB inbuilt function findchangept(x).

```
clc
clear all
[NoiselessAudio, fs] = audioread ("NoiselessAudio.wav");

%% Plotting time function of the audio
NoiselessAudio = NoiselessAudio(:,1);

% visual representation of audio using detectSpeech function
detectSpeech(NoiselessAudio,fs,'Window',hann(512,'periodic'),'OverlapLength',256)

%% Finding time locations of splicing in spectogram
window = hamming(1024);  % divide the signal into segments and perform windowing
noverlap = 512;   % samples of overlap between adjoining segments
nfft = 1024;   % sampling points to calculate the discrete Fourier transform
[s,f,t,ps]=spectrogram (NoiselessAudio, window, noverlap, nfft, fs, 'yaxis');

% returns the index at which the mean of the function (x) changes most significantly
P=findchangepts(NoiselessAudio,'MaxNumChanges',50);
```

```
spectrogram (NoiselessAudio, window, noverlap, nfft, fs, 'yaxis');
```

NOTE: The P variable will give the number of splice points which came out be 49. To obtain the locations we converted that value into the time axis by dividing it by 100.



```
P                49x1 double
```

3- Flow chart and discussion of the algorithm used to locate the spliced audios in time.

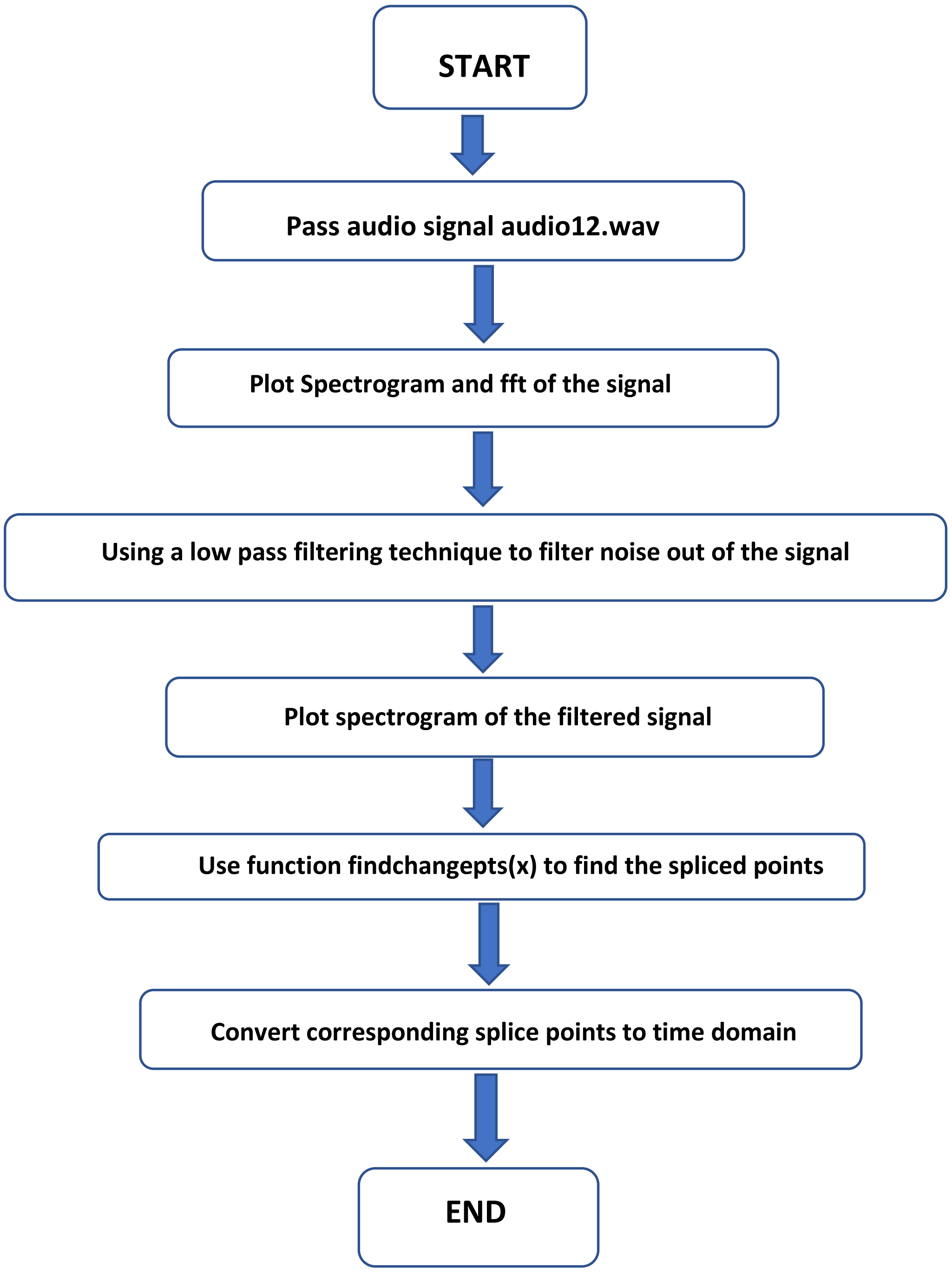
Discussion

The approach we have taken as an initial step is to remove the background noise from the audio signal, thus we get to analyze the signal without much distortion. For this we calculated the array size of the original signal which came out to be 1897967*1. Now we have created two arrays of same length, one for the voice signal and the other one for the noise.

Now we obtained the Fourier transform of the original sample. The noise will have a higher frequency compared to the voice therefore we used a low pass filtering technique to filter out higher frequency components and preserve only the noise.

Now we use this smoother signal to proceed with our analysis. The spectrogram of the new signal was plotted. In the spectrogram the spliced segments were clearly visible with straight lines. Now the challenge was to identify which points have been spliced. For this we used the MATLAB inbuilt function findchangepts(x) function which finds the location at which the mean of the function x changes rapidly. Next, we converted that to the corresponding time domain by dividing it by 100 which gave the time points where splicing has been taken place.

Algorithm


START
Pass audio signal audio12.wav
Plot Spectrogram and fft of the signal
Using a low pass filtering technique to filter noise out of the signal
Plot spectrogram of the filtered signal
Use function findchangepts(x) to find the spliced points
Convert corresponding splice points to time domain
END

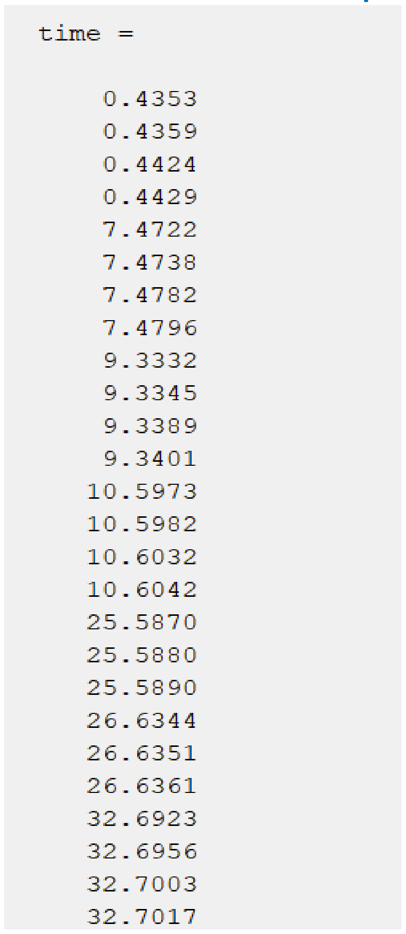
4- **Mention the total number of audios being spliced and their locations in time on the spectrogram.**

ANS: The total number of audios spliced was 49 and the location of its splicing is found as follows. We computed the abrupt change in the mean of the signal using findchangepts(X) function and converted that array into the time axis to get the spliced location.

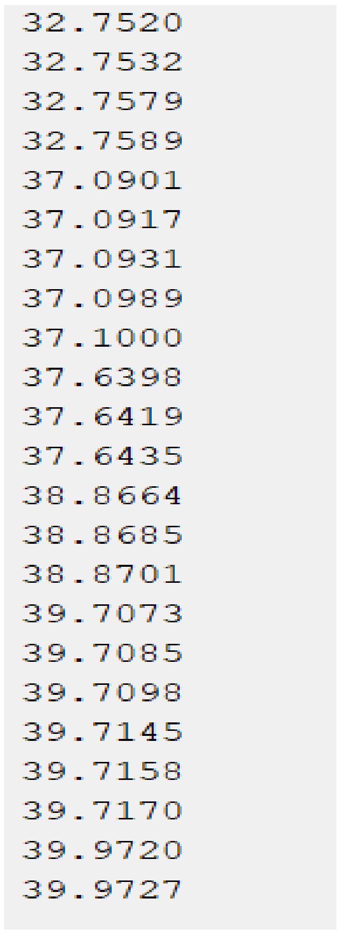
CODE

```
clc
clear all
[NoiselessAudio, fs] = audioread ("NoiselessAudio.wav");

%% Finding time locations of splicing in spectogram
NoiselessAudio = NoiselessAudio(:,1);
window = hamming(1024);  % divide the signal into segments
and perform windowing
noverlap = 512;   % samples of overlap between adjoining
segments
nfft = 1024;   % sampling points to calculate the discrete
Fourier transform
[s,f,t,ps]=spectrogram (NoiselessAudio, window, noverlap,
nfft, fs, 'yaxis');

% returns the index at which the mean of the function (x)
changes most significantly
ipt = findchangepts(pow2db(ps),'MaxNumChanges',50)
```

The below show the corresponding time points.



```
time =

    0.4353
    0.4359
    0.4424
    0.4429
    7.4722
    7.4738
    7.4782
    7.4796
    9.3332
    9.3345
    9.3389
    9.3401
   10.5973
   10.5982
   10.6032
   10.6042
   25.5870
   25.5880
   25.5890
   26.6344
   26.6351
   26.6361
   32.6923
   32.6956
   32.7003
   32.7017
```



```
32.7520
32.7532
32.7579
32.7589
37.0901
37.0917
37.0931
37.0989
37.1000
37.6398
37.6419
37.6435
38.8664
38.8685
38.8701
39.7073
39.7085
39.7098
39.7145
39.7158
39.7170
39.9720
39.9727
```

5- Discussion on the limitations of the spectrogram-based technique.

The spectrogram-based techniques although provide us with a convenient way to detect splicing in an audio, it has few limitations. Firstly, it only provides a line in the spectral graph where has taken place and this does not provide any numerical value. Thus, accuracy is not guaranteed.

Also secondly, the user should know the exact parameters to adjust the window adjustment to get correct balance between the time and frequency component for better results. This is not always feasible with beginners.

Finally in the code

```
P=findchangepts(NoiselessAudio,'MaxNumChanges',50);
```

The user must give the sampling value. Here we gave 50 but the optimized value for each signal varies thus the accuracy is affected when user gives random values. Therefore, we can conclude spectrogram techniques, although is useful has its own limitations.