AI Noise Reduction

Project Report

Eklavya Mentorship Programme

At

SOCIETY OF ROBOTICS AND AUTOMATION,

VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE, MUMBAI

OCTOBER 2021

ACKNOWLEDGEMENT

We are extremely grateful to our mentors Shreyas Atre, Harsh Shah and Gautam Agarwal for their support and guidance throughout the duration of the project.
We would also like to thank all the members of SRA VJTI for their timely support as well as for organising Eklavya and giving us a chance to work on this project.

Priyal Awankar
priyalawankar@gmail.com

Dhriti Mabian
dhritimabian123@gmail.com

GitHub Link
https://github.com/Dhriti03/ai-noise-reduction

INDEX

	~Sinusoidal Sequence **4. Discrete time signals** - Basic operations on DT signals ~ Addition | Multiplication | Amplitude Scaling | Time Scaling ~ Shifting | Multiplication Transform - Classification of DT Signals :- + Periodic & Non periodic DT signals . + Odd and Even DT signals . + Energy and Power of DT signal **5. Complex exponentials** **6. Convolution** **7. Conversions (of Amp and DB scale)** **8. Fourier Transform** - DFT - Short Time FT - ISTFT **9. Masking and threshold of signal**	
5.	**Future Implementation** **Application :-** - Background noise cancellation (Headphones) - Medical scanning (Heart related noise)	
6.	**Reference Links**	

PROJECT OVERVIEW

To reduce environmental noise with a noise reduction algorithm in python that reduces noise in time-domain signals like speech, bioacoustics, and physiological signals. It relies on a method called "spectral gating" which is a form of Noise Gate. It works by computing a spectrogram of a signal (and optionally a noise signal) and estimating a noise threshold (or gate) for each frequency band of that signal/noise. That threshold is used to compute a mask, which gates noise below the frequency-varying threshold.

Introduction to the project :-

Have you ever woken up without understanding what it was, but knowing for sure that some sound isn't right?

Sound identification is one of our instincts that keeps human beings safe. Sounds play a significant role in our life, Starting from recognizing a predator nearby to being inspired by music, to groups of human voices, to the cry of a bird. Inevitably, developing audio classifiers is a crucial task in our lives.

Ordinarily, it is essential to classify the sounds' source and is already widely used for various purposes. In music, there's a classifier for the genre of music. Recently similar systems began to be used to classify bird calls, historically done by Ornithologists. Their goal is to categorize birds, considering it is challenging to discover bird calls from the fields or noisy surroundings.

Recently, Deep Learning (DL) has become one of the popular technologies to solve multiple tasks in our lives due to its accuracy and the improvement of computational devices like CPU (Central Processing Unit), GPU (Graphics Processing Unit). The below chart shows how influential the deep learning market is and its expected size from the aspects of the software, hardware, and services.

Nature of the signal :-

A signal is defined as the physical phenomenon which carries **some information or Data** . The signals are usually functions of **independent variables of time** . These are some cases where the signals are not a function of time; the electric charge distributed in the body is a signal which is a function of space and not time .

Signals occur naturally and they are also synthesized. The term is used for referring to a wide variety of information.

Consider:

(1) an acoustic wave, which can convey speech or music information,

(2) an electromagnetic wave, which is modulated to transport, for example, image information,

The System is defined as the set of interconnected objects with a definite relationship between objects and attributes. The interconnected components provide desired functions .

- Noise and behaviour of noise .

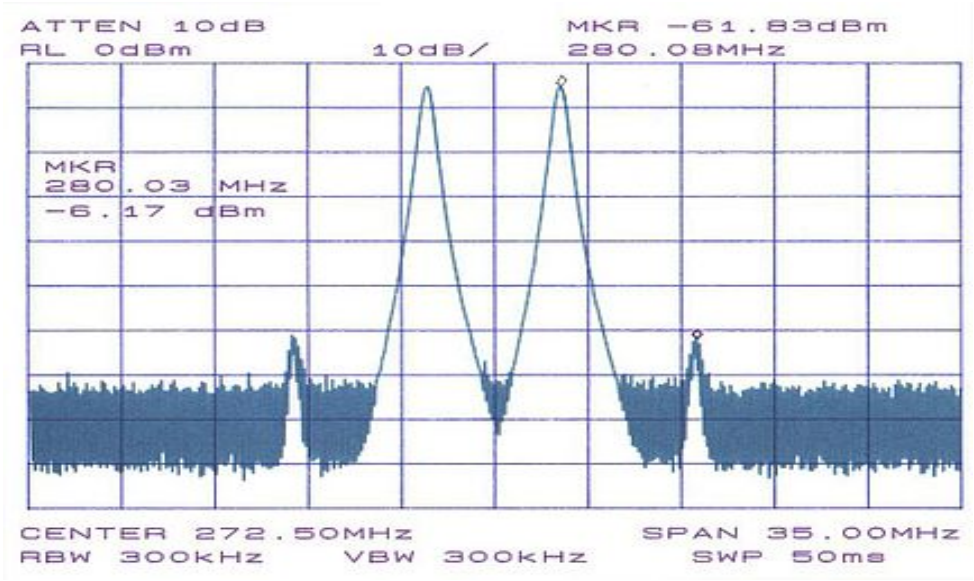
In electronics, noise is an unwanted disturbance in an electrical signal. ... In communication systems, noise is an error or undesired random disturbance of a useful information signal. The noise is a summation of unwanted or disturbing energy from natural and sometimes man-made sources.

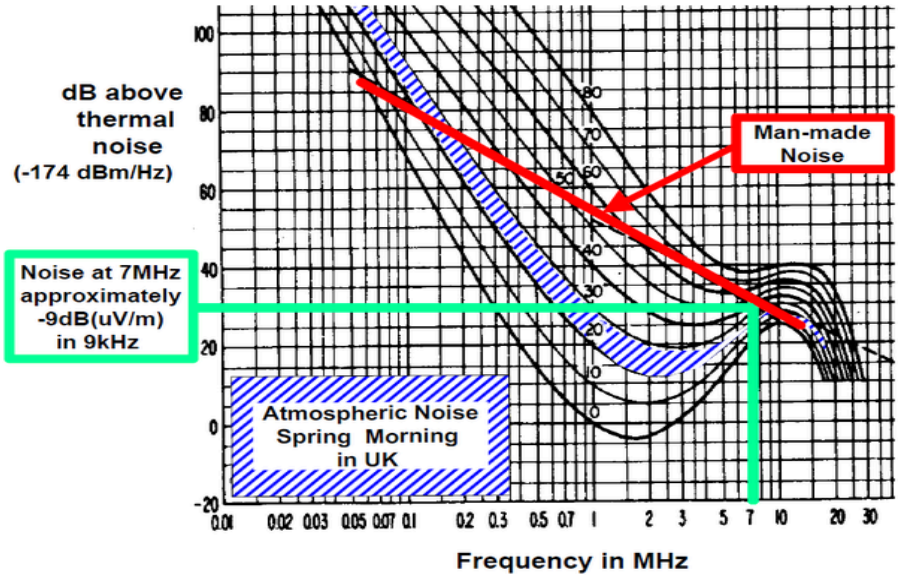
Intermodulation noise

Caused when signals of different frequencies share the same nonlinear medium.

Atmospheric noise:-

Also called static noise, it is caused by lightning discharges in thunderstorms and other electrical disturbances occurring in nature.



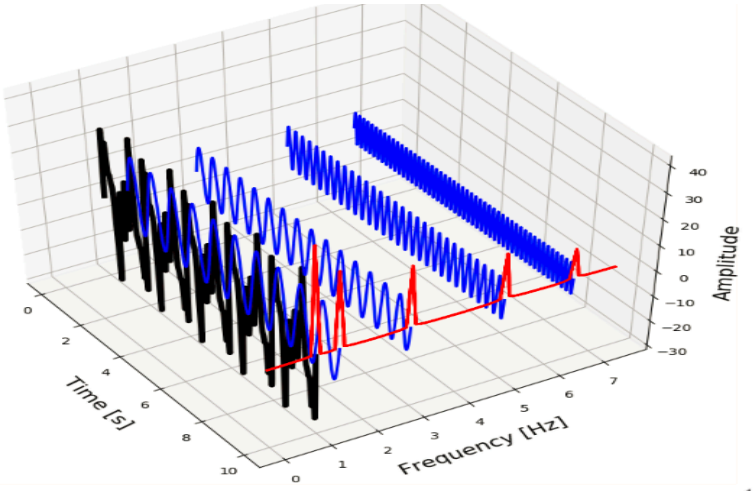


Industrial noise

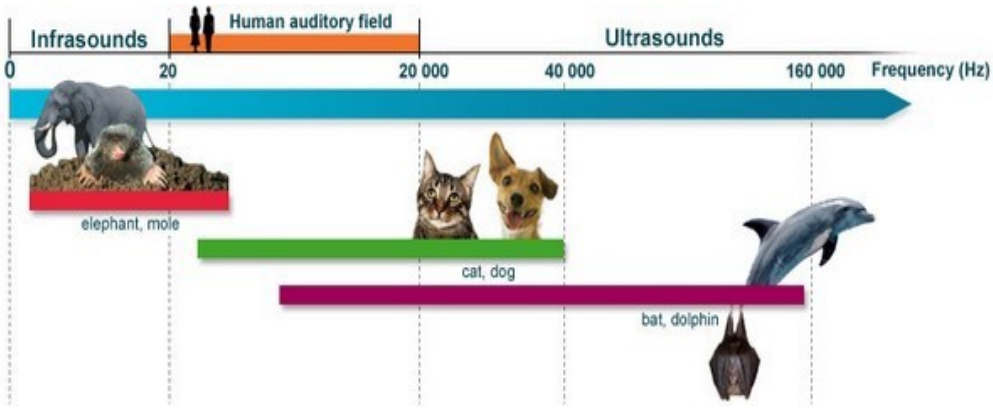
Sources such as automobiles, aircraft, ignition electric motors and switching gear, High voltage wires and fluorescent lamps cause industrial noise. These noises are produced by the discharge present in all these operations.

- **Information extraction from the given Audio signal**

- Audio waves are the vibration of air molecules whenever any sound happens and sound travels from originator to the receiver in the form of a wave.

- As such, this wave has 3 properties to it — Amplitude, Frequency and Time.
- → Amplitude represents the magnitude of the wave signal and it is usually measured in decibels (dB).
- → Time is the time scale, as we all know it.
- → Frequency represents how many complete cycle the wave takes in one second and it is measured in Hz.
- 

- Every living being has a different hearing range of sound waves.
- We (humans) could hear sound waves that are in the range of 20 Hz to 20,000 Hz. Dogs could hear the sound waves upto 45K Hz and dolphins could hear until 150K Hz.

- 

-
-

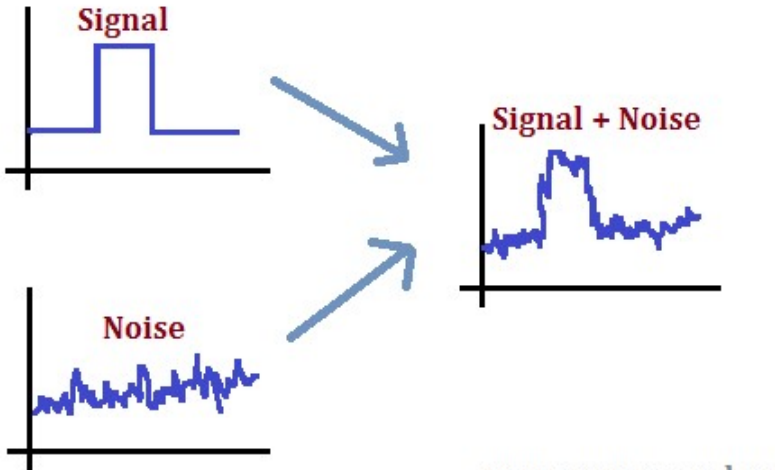
-
- As human hearing range is around 20K Hz, sampling rate of audio files in many libraries are by default set at 22050 per sec. We do have an option of increasing/decreasing it as per the need.
- Note audio waves are continuous in nature — but most of our processing engines are built to process digital/discrete signals.
-

METHODS AND STAGES OF PROGRESS:-

- We give the file location and read the wav file with the data clipping.Display and plot the data and noise of the Audio signal .
- **Define the FFT noise** Then define the band limited noise where a function or time series whose Fourier transform is restricted to a finite range of frequencies or wavelengths with the above defined fft function .
- Then consider the **noise length** and by applying the **band limited noise** applying the noise data and basically adding the noise additionally to data to expand the size of the data .i.e **noise clipping** to limit the band .
- Then to remove the noise from the signal based on the audio and the noise clip define a function for it .
- Define the **Short Time Fourier Transform** (which can perform time–frequency analysis of non-stationary signals in real time.) and calculate it with noise also then the statistics over noise by defining the noise threshold then STFT over signal in Db scale .
- Now for **masking the window** over the time signal and having an absolute value for the STFT we calculate the value of the mask .
- Then make a **smoothing filter** for the mask in time and frequency .calculate the threshold for each frequency/time bin
- Mask if the signal is above the **threshold. convolve the mask** with a smoothing filter mask the signal thus apply the mask and thus **plot the signal** and display the audio of the recovered signal

What Does Noise Signal Mean?

Noise is an unwanted signal, which interferes with the original message signal and corrupts the parameters of the message signal. This alteration in the communication process, leads to the message getting altered.
Noise signal refers to a ratio that is used in science and engineering to compare the levels of a desired signal with the level of background noise. It is expressed as the ratio of signal power to background noise power and is stated in decibels. A ratio in excess of 1:1 means that there is more signal than noise.



Noise signal is sometimes used informally to refer to any ratio of useful information to irrelevant information. Noise can be defined as unwanted electrical or electromagnetic energy that degrades the quality of signals and data. Noises occur in digital and analogue systems and can affect files and communications of all types.

Digital Signal Processing :

Anything that carries information can be called as signal. It can also be defined as a physical quantity that varies with time, temperature, pressure or with any independent variables such as speech signal or video signal.

The process of operation in which the characteristics of a signal
Amplitude,shape,phase,frequency,etc.

These parameters undergo change known as signal processing.

Note − Any unwanted signal interfering with the main signal is termed as noise. So, noise is also a signal but unwanted.

According to their representation and processing, signals can be classified into various categories .

1. Continuous Time Signals

Continuous-time signals are defined along a continuum of time and are thus represented by a continuous independent variable. Continuous-time signals are often referred to as analog signals.

This type of signal shows continuity both in amplitude and time. These will have values at each instant of time. Sine and cosine functions are the best example of Continuous time signals.

A continuous-time (CT) signal is a function, s(t), that is defined for all time t contained in some interval on the real line. For historical reasons, CT signals are often called analog signals.

Linear CT systems are characterized by the familiar mathematics of differential equations, continuous convolution operators, **Laplace transforms, and Fourier transforms**. Similarly, linear DT systems are described by the mathematics of difference equations, discrete convolution operators, *Z*-transforms, and discrete Fourier transforms. It appears that for every major concept in CT systems,

Analytical difficulties often occur at the boundaries between the analog and digital portions of the system because the mathematics used on the two sides of the interface must be different. It is often useful to assume that a sequence $s(n)$ is derived from an analog signal $sa(t)$ by ideal sampling, i.e.

(Eq. 1)

$$\mathbf{s(n)=sa(t)|t=nT}$$

An alternative model for the sampled signal is denoted by $s^*(t)$ and defined by

(Eq. 2)

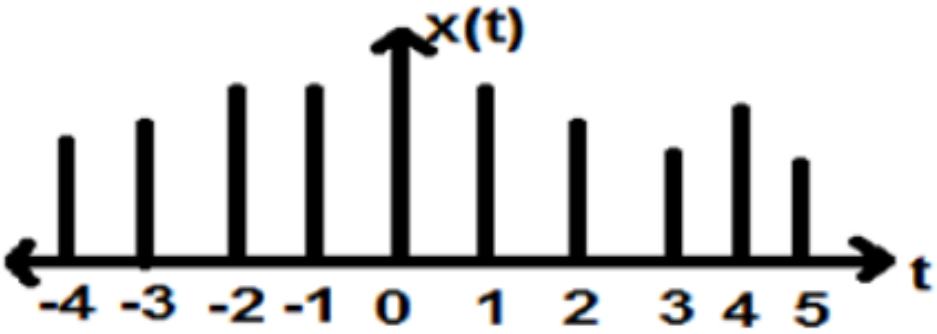
$$\mathbf{s^*(t)=\sum n=-\infty+\infty sa(t)\delta a(t-nT)}$$

where $\delta a(t)$ is an analog impulse function. Both $s(n)$ and $s^*(t)$ are used throughout the literature to represent an ideal sampled signal. Note that even though $s(n)$ and $s^*(t)$ represent the same essential information, $s(n)$ is a DT signal and $s^*(t)$ is a CT signal.

2. Discrete Time signals

The signals, which are defined at discrete times are known as discrete signals. Therefore, every independent variable has a distinct value. Thus, they are represented as sequences of numbers.

Although speech and video signals have the privilege to be represented in both continuous and discrete time format; under certain circumstances, they are identical. Amplitudes also show discrete characteristics. Perfect example of this is a digital signal; whose amplitude and time both are discrete.



Basic operations on DT signals :-

Here we will concentrate on the basic signal operations which manipulate the signal characteristics by acting on the independent variable(s) which are used to represent them. This means that instead of performing operations like addition, subtraction, and multiplication between signals, we will perform them on the independent variable. In our case, this variable is time (*t*).

Discrete-time signals are represented mathematically as sequences of numbers. A sequence of numbers x, in which the nth number in the sequence is denoted x[n],

1 is

formally written as

$x = \{x[n]\}, -\infty < n < \infty$, (2.1)

where n is an integer. In a practical setting, such sequences can often arise from periodic

sampling of an analog (i.e., continuous-time) signal xa(t). In that case, the numeric value

of the nth number in the sequence is equal to the value of the analog signal, xa(t), at

time nT : i.e.,

$x[n] = x_a(nT), -\infty < n < \infty$. (2.2)

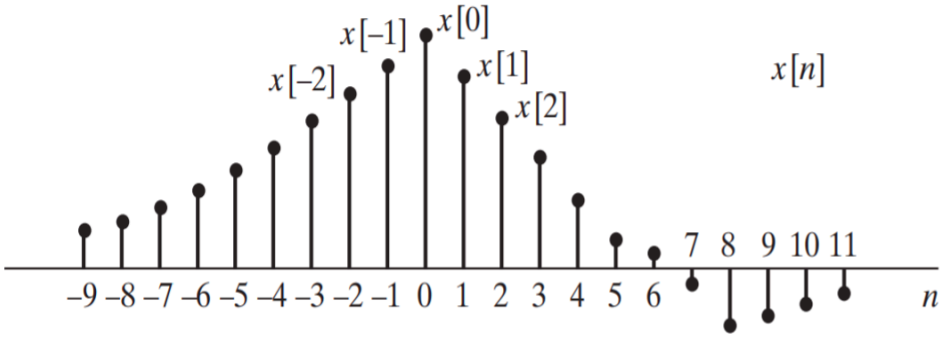
The quantity T is **the sampling period**, and its reciprocal is **the sampling frequency.** Although sequences do not always arise from sampling analog waveforms, it is convenient

1Note that we use [ ] to enclose the independent variable of discrete-variable functions, and we use ( )

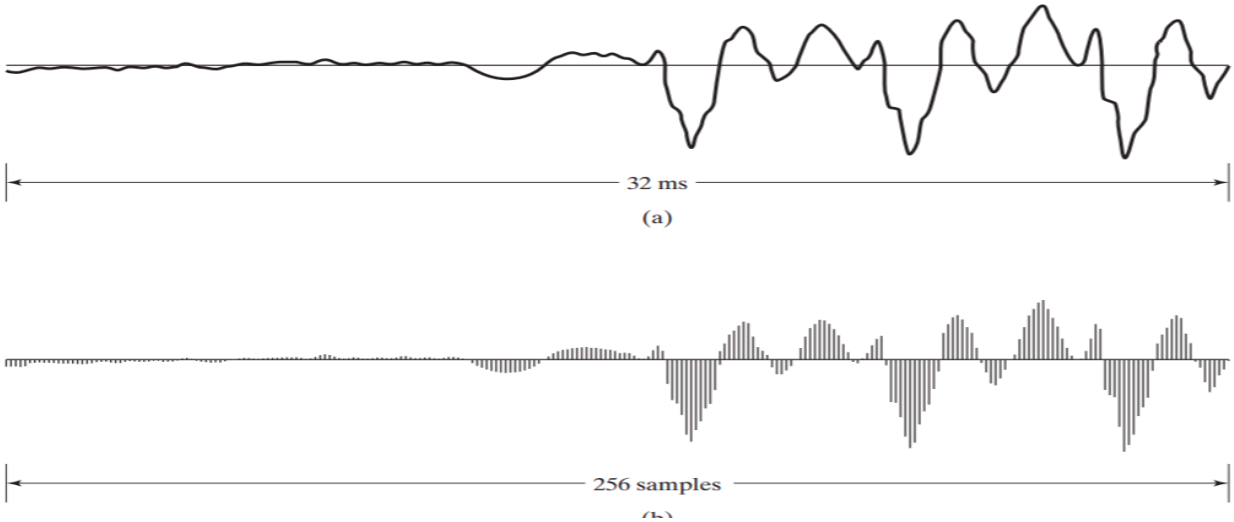
to enclose the independent variable of continuous-variable functions.

to refer to x[n] as the "**nth sample**" of the sequence. Also, although, strictly speaking, x[n] denotes the **nth number in the sequence,** the notation of Eq. (2.1) is often unnecessarily cumbersome, and it is convenient and unambiguous to refer to "the sequence x[n]" when we

mean the entire sequence, just as we referred to "the analog signal xa(t)." We depict **discrete-time signals** (i.e., sequences) graphically, as shown in Figure 2.1. Although the abscissa is drawn as a continuous line, it is important to recognize that x[n] is defined **only for integer values of n.** It is not correct to think of x[n] as being zero when n is not an integer; x[n] is simply undefined for noninteger values of n.



As an example of a sequence obtained by sampling, Figure 2.2(a) shows a segment of a speech signal corresponding to **acoustic pressure variation** as a **function of time,** and Figure 2.2(b) presents a sequence of samples of the speech signal. Although the original speech signal is defined at all values of time t, the sequence contains information about the **signal only at discrete instants.**



The various DT Signals :-

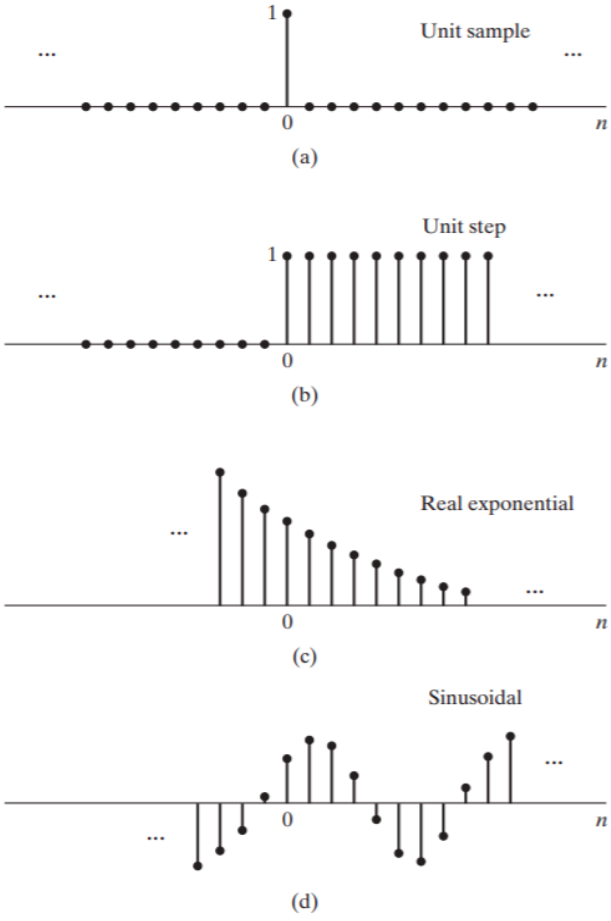


Figure 2.3 Some basic sequences. The sequences shown play important roles in the analysis and representation of discrete-time signals and systems.

Time shifting and Amplitude shifting

Time Shifting

Time shifting means, shifting of signals in the time domain. Mathematically, it can be written as

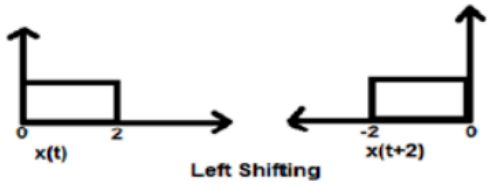
$$x(t) \rightarrow y(t+k)$$

This K value may be positive or it may be negative. According to the sign of k value, we have two types of shifting named as Right shifting and Left shifting.

Case 1 $K > 0$

When K is greater than zero, the shifting of the signal takes place towards "left" in the time domain. Therefore, this type of shifting is known as Left Shifting of the signal.

Example



Case 2 $K < 0$

When K is less than zero the shifting of signal takes place towards right in the time domain. Therefore, this type of shifting is known as Right shifting.

Amplitude Shifting

Amplitude shifting means shifting of signal in the amplitude domain $around X-axis$.

Mathematically, it can be represented as −

$$x(t) \rightarrow x(t)+K$$

This K value may be positive or negative. Accordingly, we have two types of amplitude shifting which are subsequently discussed below.

Case 1 $K > 0$

When K is greater than zero, the shifting of signal takes place towards up in the x-axis. Therefore, this type of shifting is known as upward shifting.

Time Scaling ,Time compression :-

Time Scaling

If a constant is multiplied to the time axis then it is known as Time scaling. This can be mathematically represented as;

$$x(t) \rightarrow y(t) = x(\alpha t) \quad \text{or} \quad x(\tfrac{t}{\alpha}) \quad ; \text{where } \alpha \neq 0$$

So the y-axis being same, the x- axis magnitude decreases or increases according to the sign of the constant $whetherpositiveornegative$. Therefore, scaling can also be divided into two categories as discussed below.

Time Compression

Whenever alpha is greater than zero, the signal's amplitude gets divided by alpha whereas the value of the Y-axis remains the same. This is known as Time Compression.

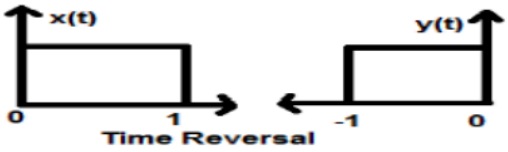
Time Reversal & Amplitude Reversal :-

Time Reversal

Whenever signal's time is multiplied by -1, it is known as time reversal of the signal. In this case, the signal produces its mirror image about Y-axis. Mathematically, this can be written as;

$$x(t) \rightarrow y(t) \rightarrow x(-t)$$

This can be best understood by the following example.



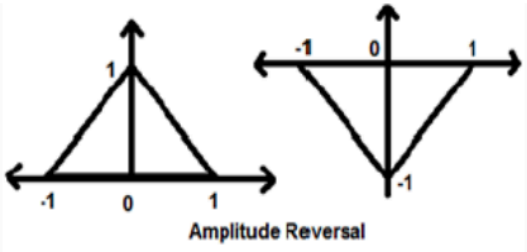
In the above example, we can clearly see that the signal has been reversed about its Y-axis. So, it is one kind of time scaling also, but here the scaling quantity is -1 always.

Amplitude Reversal

Whenever the amplitude of a signal is multiplied by -1, then it is known as amplitude reversal. In this case, the signal produces its mirror image about X-axis. Mathematically, this can be written as;

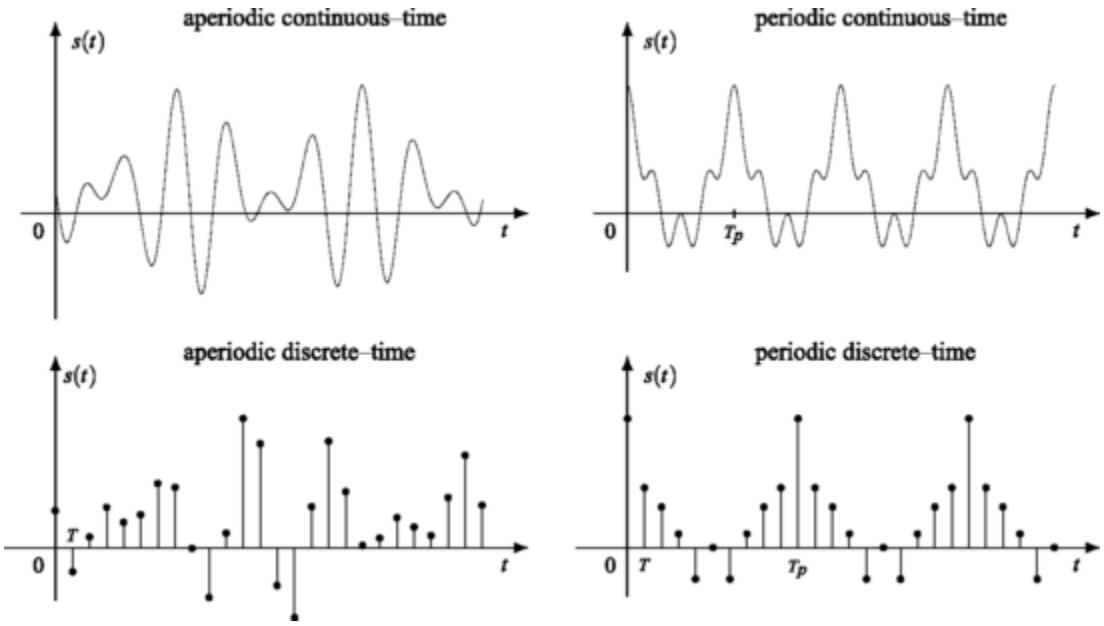
$$x(t) \rightarrow y(t) \rightarrow -x(t)$$

Consider the following example. Amplitude reversal can be seen clearly.



Classification of DT Signals :-

Periodic & Non periodic DT signals .



Odd and Even DT signals .

Even Signal: A signal is referred to as an even if it is identical to its time-reversed counterparts; x(t) = x(-t).

Odd Signal: A signal is odd if x(t) = -x(-t). An odd signal must be 0 at t=0, in other words, odd signal passes the origin.

ve assume for definiteness a unit load (a 1 Ω resistor), then if $f(t)$ is the voltage across the resistor or the current through the resistor, then the ergy dissipated is

$$E_f = \int_{-\infty}^{\infty} f^2(t)dt$$

his quantity is finite, then $f(t)$ is said to be an **energy signal** . Not every signal that we consider analytically is an energy signal: for example, $t) = \cos(\omega_o t)$ is not an energy signal.

bstituting in for the $f(t)$ in terms of the inverse F.T.,

$$E_f = \int_{-\infty}^{\infty} f(t)\left[\frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}d\omega\right]dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\int_{-\infty}^{\infty} f(t)e^{j\omega t}dtd\omega$$

we can write

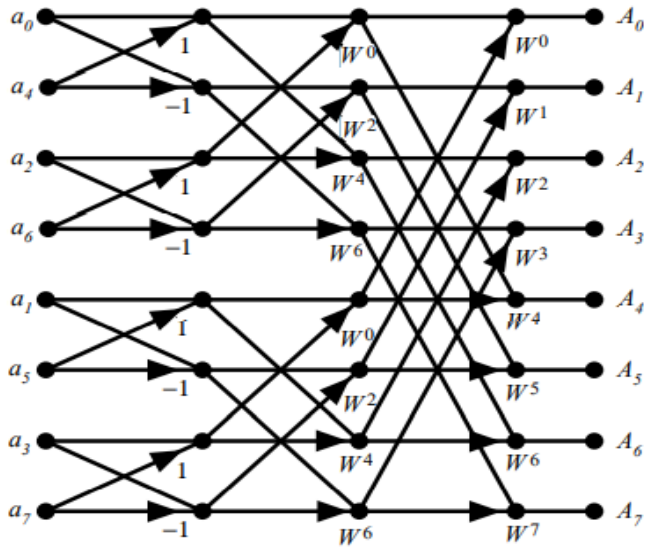
FFT Convolution:-

FFT convolution uses the principle that multiplication in the frequency domain corresponds to convolution in the time domain.
The input signal is transformed into the frequency domain using the DFT, multiplied by the frequency response of the filter, and then transformed back into the time domain using the Inverse DFT.
This basic technique was known since the days of Fourier; however, no one really cared. This is because the time required to calculate the DFT was longer than the time to directly calculate the convolution.
This changed in 1965 with the development of the Fast Fourier Transform (FFT). By using the FFT algorithm to calculate the DFT, convolution via the frequency domain can be faster than directly convolving the time domain signals.
The final result is the same; only the number of calculations has been changed by a more efficient algorithm. For this reason, FFT convolution is also called high-speed convolution.

Fast Fourier Transform :

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa.
The FFT is a fast algorithm for computing the DFT. If we take the 2-point DFT and 4-point DFT and generalize them to 8-point, 16-point, ..., 2r -point, we get the FFT algorithm.
To compute the DFT of an N-point sequence using equation (1) would take O(N2) multiplies and adds. The FFT algorithm computes the DFT using O(N log N) multiplies and adds. There are many variants of the FFT algorithm.
We'll discuss one of them, the "decimation in-time" FFT algorithm for sequences whose length is a power of two (N = 2r for some integer r). Below is a diagram of an 8-point FFT,

where W = W8 = e−iπ/4 = (1 − i)/√2:



As the name implies, the Fast Fourier Transform (FFT) is an algorithm that determines Discrete Fourier Transform of an input significantly faster than computing it directly. In computer science lingo, the FFT reduces the number of computations needed for a problem of size N from O(N^2) to O(NlogN) .

FFT Explained Using Matrix Factorization

The 8-point DFT can be written as a matrix product, where we let
W = W8 = e−iπ/4 = (1 − i)/√2:

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{pmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^1 & W^4 & W^7 & W^2 & W^5 \\ W^0 & W^4 & W^0 & W^4 & W^0 & W^4 & W^0 & W^4 \\ W^0 & W^5 & W^2 & W^7 & W^4 & W^1 & W^6 & W^3 \\ W^0 & W^6 & W^4 & W^2 & W^0 & W^6 & W^4 & W^2 \\ W^0 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{pmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^4 & W^2 & W^6 & W^1 & W^5 & W^3 & W^7 \\ W^0 & W^0 & W^4 & W^4 & W^2 & W^2 & W^6 & W^6 \\ W^0 & W^4 & W^6 & W^2 & W^3 & W^7 & W^1 & W^5 \\ W^0 & W^0 & W^0 & W^0 & W^4 & W^4 & W^4 & W^4 \\ W^0 & W^4 & W^2 & W^6 & W^5 & W^1 & W^7 & W^3 \\ W^0 & W^0 & W^4 & W^4 & W^6 & W^6 & W^2 & W^2 \\ W^0 & W^4 & W^6 & W^2 & W^7 & W^3 & W^5 & W^1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_4 \\ a_2 \\ a_6 \\ a_1 \\ a_5 \\ a_3 \\ a_7 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \cdot & \cdot & \cdot & W^0 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & W^1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & W^2 & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & W^3 \\ 1 & \cdot & \cdot & \cdot & W^4 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & W^5 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & W^6 & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & W^7 \end{pmatrix} \begin{pmatrix} 1 & \cdot & W^0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & W^2 & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & W^4 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & W^6 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & W^0 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & W^2 \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & W^4 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & W^6 \end{pmatrix} \begin{pmatrix} 1 & W^0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & W^4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & W^0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & W^4 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & W^0 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & W^4 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & W^0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & W^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_4 \\ a_2 \\ a_6 \\ a_1 \\ a_5 \\ a_3 \\ a_7 \end{pmatrix}$$

where "·" means 0. These are sparse matrices (lots of zeros), so multiplying by the dense (no zeros) matrix on top is more expensive than multiplying by the three sparse matrices on the bottom. For N = 2^r , the factorization would involve r matrices of size N × N, each with 2 nonzero entries in each row and column

How Much Faster is the FFT?

To compute the DFT of an N-point sequence using the definition,

$$A_k = \sum_{n=0}^{N-1} W_N^{kn} a_n,$$

would require N2 complex multiplies and adds, which works out to 4N2 real multiplies and 4N2 real adds (you can easily check this, using the definition of complex multiplication). The basic computational step of the FFT algorithm is a butterfly. Each butterfly computes two complex numbers of the form p + αq and p − αq, so it requires one complex multiply (α · q) and two complex adds. This works out to 4 real multiplies and 6 real adds per butterfly.

There are N/2 butterflies per stage, and log2 N stages, so that means about 4 · N/2 · log2 N = 2N

log2 N real multiplies and 3N log2 N real adds for an N-point FFT. (There are ways to optimize further, but this is the basic FFT algorithm.)
Cost comparison:

N	$r = \log_2 N$	**BRUTE FORCE** $4N^2$	**FFT** $2N \log_2 N$	speedup
2	1	16	4	4
4	2	64	16	4
8	3	256	48	5
1,024	10	4,194,304	20,480	205
65,536	16	$1.7 \cdot 10^{10}$	$2.1 \cdot 10^{6}$	~10^4

The **FFT algorithm is a LOT faster for big N**. There are also FFT algorithms for N not a power of two. The algorithms are generally fastest when N has many factors, however.

Short-Time Fourier Transform (STFT):-

Short-time Fourier transform (STFT) is a sequence of Fourier transforms of a windowed signal. STFT provides the time-localized frequency information for situations in which frequency components of a signal vary over time, whereas the standard Fourier transform provides the frequency information averaged over the entire signal time interval.

We are primarily concerned here with tuning the STFT parameters for the following applications:

1. Approximating the time-frequency analysis performed by the *ear* for purposes of *spectral display*.
2. Measuring *model parameters* in a short-time spectrum.

Short-Time Fourier Transform

- **Basic Concept:** – Break up the signal in time domain to a number of signals of shorter duration, then transform each signal to frequency domain
- Requires fewer number of harmonics to regenerate the signal chunks
- Helps determine the time interval in which certain frequencies occur.
- Need a local analysis scheme for a time-frequency representation (TFR). Windowed F.T. or Short Time F.T. (STFT)
- Segmenting the signal into narrow time intervals (i.e., narrow enough to be considered stationary).
- Take the Fourier transform of each segment.
- Inverse STFT – The original signal can be recovered from the transform by the Inverse STFT. The most widely accepted way of inverting the STFT is by using the overlap-add (OLA) method, which also allows for modifications to the STFT complex spectrum.
- Calculating the InverseSTFT: – First, it is required that the window function must be scaled such that the area underneath the window function is unity.

MATHEMATICAL DEFINITION OF THE STFT

The usual mathematical definition of the STFT is

$$\begin{aligned} X_m(\omega) &= \sum_{n=-\infty}^{\infty} x(n)w(n-mR)e^{-j\omega n} \\ &= \mathrm{DTFT}_\omega(x \cdot \textsc{Shift}_{mR}(w)), \end{aligned}$$

where

$$\begin{aligned} x(n) &= \text{input signal at time } n \\ w(n) &= \text{length } M \text{ window function } (e.g., \text{Hamming}) \\ X_m(\omega) &= \text{DTFT of windowed data centered about time } mR \\ R &= \text{hop size, in samples, between successive DTFTs.} \end{aligned}$$

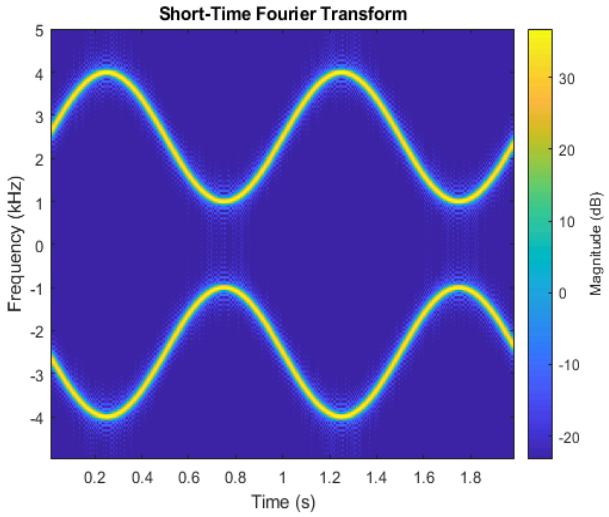
If the window $w(n)$ has the *Constant OverLap-Add (COLA) property* at hop-size R, *i.e.*, if

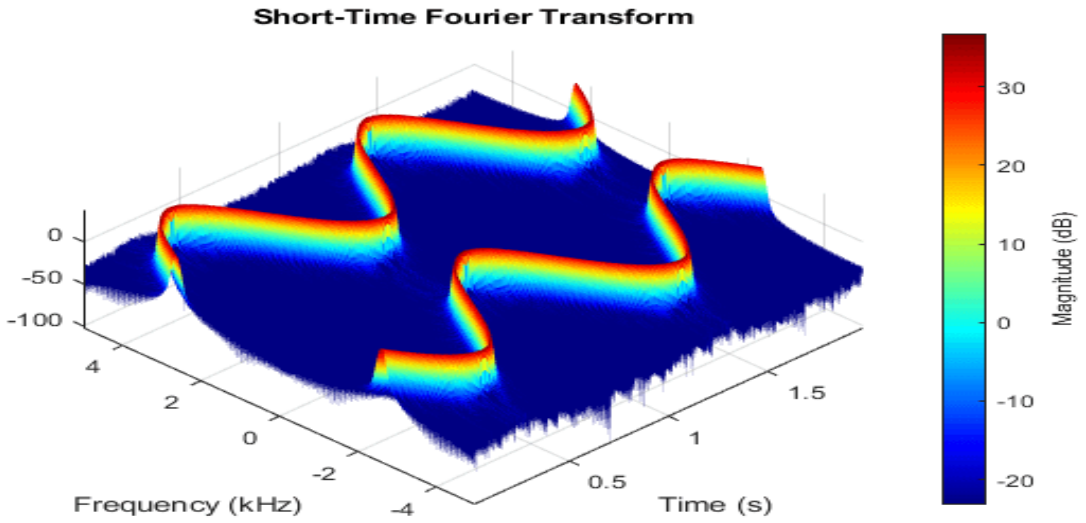
$$\boxed{\sum_{m=-\infty}^{\infty} w(n-mR) = 1, \ \forall n \in \mathbf{Z}} \quad (w \in \textsc{Cola}(R)) \tag{8.2}$$

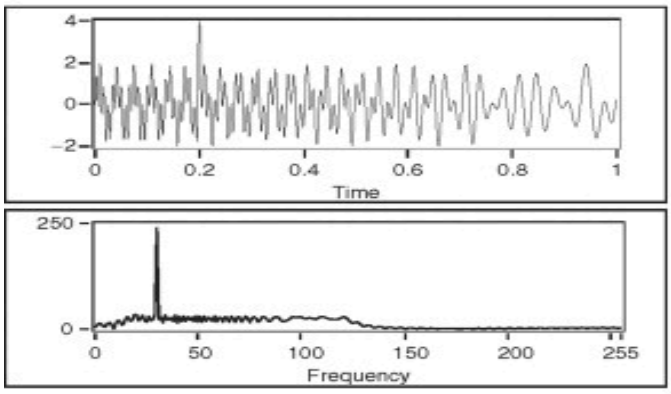
then the sum of the successive DTFTs over time equals the DTFT of the whole signal $X(\omega)$:

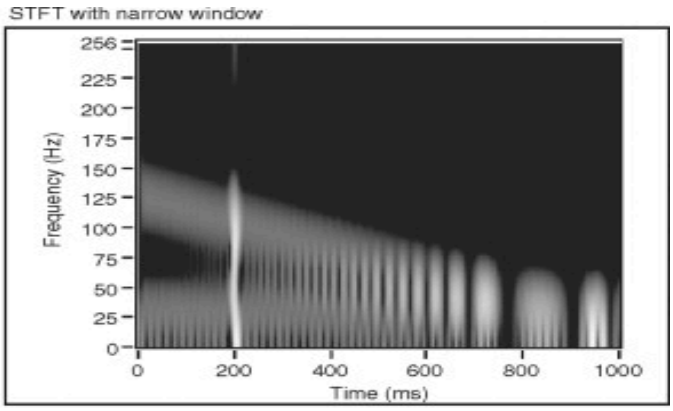
$$\begin{aligned}
\sum_{m=-\infty}^{\infty} X_m(\omega) &\triangleq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x(n) w(n-mR) e^{-j\omega n} \\
&= \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega n} \underbrace{\sum_{m=-\infty}^{\infty} w(n-mR)}_{1 \text{ if } w \in \mathrm{COLA}(R)} \\
&= \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega n} \\
&\triangleq \mathrm{DTFT}_{\omega}(x) = X(\omega).
\end{aligned}$$

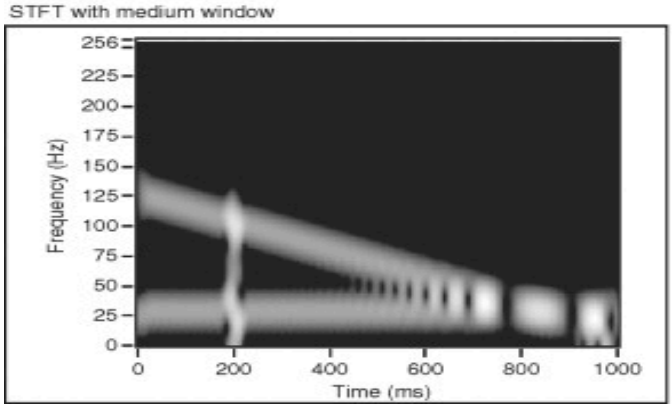
3D STFT Visualization :

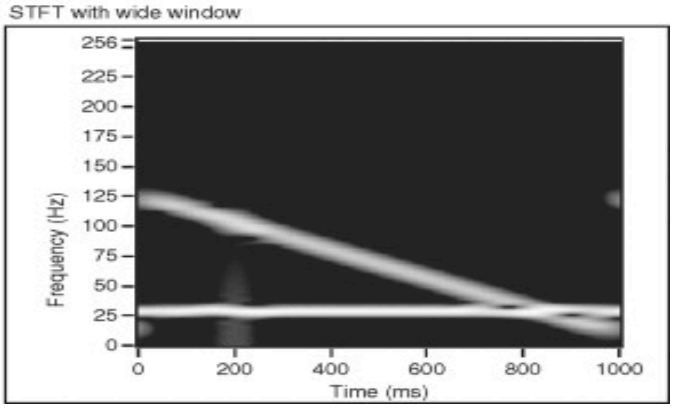


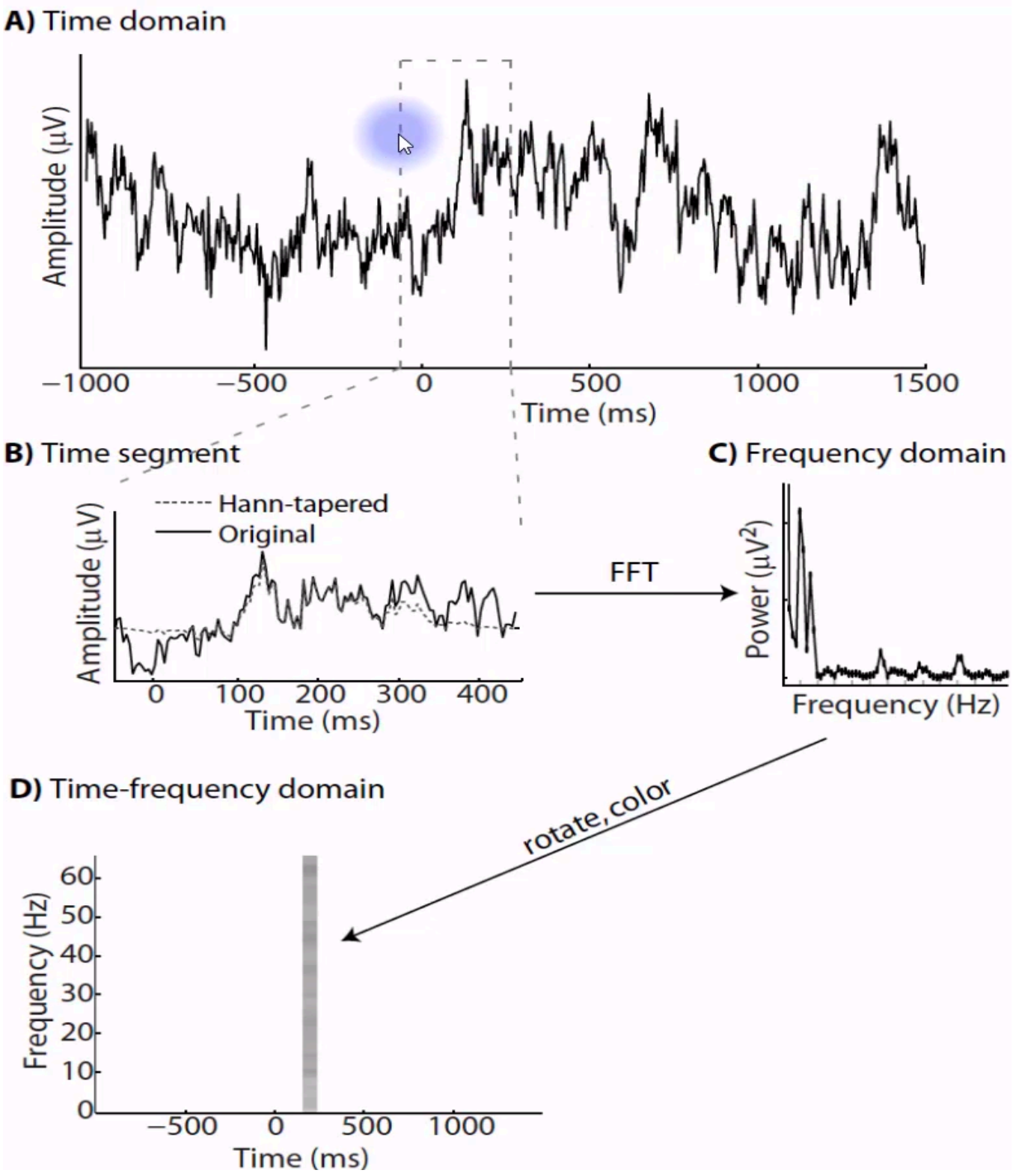

Short-Time Fourier Transform
0
-50
-100
4
2
0
-2
-4
Frequency (kHz)
0.5
1
1.5
Time (s)
30
20
10
0
-10
-20
Magnitude (dB)


4
2
0
-2
0
0.2
0.4
0.6
0.8
1
Time
250
0
0
50
100
150
200
255
Frequency


STFT with narrow window
256
225
200
175
150
125
100
75
50
25
0
Frequency (Hz)
0
200
400
600
800
1000
Time (ms)


STFT with medium window
256
225
200
175
150
125
100
75
50
25
0
Frequency (Hz)
0
200
400
600
800
1000
Time (ms)


STFT with wide window
256
225
200
175
150
125
100
75
50
25
0
Frequency (Hz)
0
200
400
600
800
1000
Time (ms)


A) Time domain
Amplitude (μV)
−1000
−500
0
500
1000
1500
Time (ms)
B) Time segment
Hann-tapered
Original
Amplitude (μV)
0
100
200
300
400
Time (ms)
FFT
C) Frequency domain
Power (μV²)
Frequency (Hz)
rotate, color
D) Time-frequency domain
Frequency (Hz)
60
50
40
30
20
10
0
−500
0
500
1000
Time (ms)

Inverse Fourier Transform :

The Fourier inversion theorem says that for many types of functions it is possible to recover a function from its Fourier transform. Intuitively it may be viewed as the statement that if we know all frequency and phase information about a wave then we may reconstruct the original wave precisely.

if $F(\omega)$ is the Fourier transform of $f(t)$, *i.e.*,

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t}\, dt$$

then

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t}\, d\omega$$

let's check

$$\begin{aligned}
\frac{1}{2\pi} \int_{\omega=-\infty}^{\infty} F(\omega) e^{j\omega t}\, d\omega &= \frac{1}{2\pi} \int_{\omega=-\infty}^{\infty} \left(\int_{\tau=-\infty}^{\infty} f(\tau) e^{-j\omega\tau} \right) e^{j\omega t} d\omega \\
&= \frac{1}{2\pi} \int_{\tau=-\infty}^{\infty} f(\tau) \left(\int_{\omega=-\infty}^{\infty} e^{-j\omega(\tau-t)} d\omega \right) d\tau \\
&= \int_{-\infty}^{\infty} f(\tau) \delta(\tau - t) d\tau \\
&= f(t)
\end{aligned}$$

Adding Mask to the Noise :

A **sound masking system** can be used to reduce the impression of intruding sound (reducing annoyance, distraction) and improve acoustic privacy (including speech privacy). However, the masking sound produced by an electroacoustical system may also be disruptive if the sound masking system is improperly designed, improperly commissioned, or not verified by a professional acoustician.

Non Threaded environments have only one thread of control. Each signal is masked or unmasked for the entire process.

Threads spawned by a parent thread inherit the signal mask of the parent thread. You can build applications to take advantage of signal-mask inheritance. If a signal must be masked for an

entire process, mask it for the main or initial thread. Any thread created thereafter inherits this thread's signal mask.
The perceptual weighting filter is based on the masking effect of the human audio system. In speech, the noise that is carried in a higher energy frequency band (around formant) is less easily perceived than the one carried in a lower energy frequency band. Thus, this conclusion can be used to measure the difference between original speech and synthetic speech, allowing a relatively large difference in a higher energy frequency band.

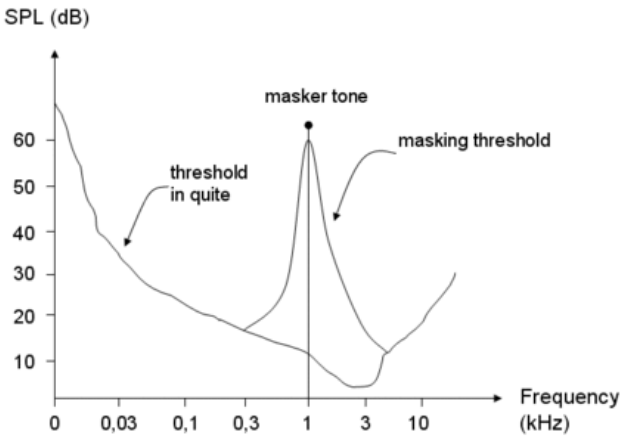
Masking Threshold :

Masking threshold refers to a process where one sound is rendered inaudible because of the presence of another sound. If someone listens to a soft and a loud sound at the same time, the subject may not hear the soft sound. The soft sound is masked by the loud sound.

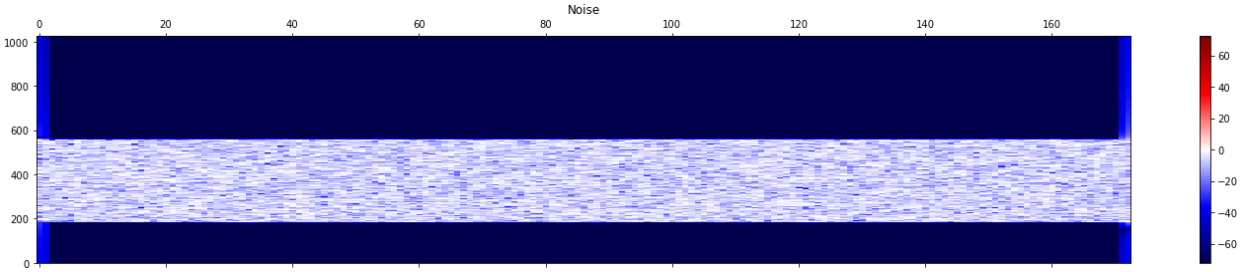
So the masking threshold is the sound pressure level of a sound needed to make the sound audible in the presence of another noise called a "masker". This threshold depends upon the frequency, the type of masker, and the kind of sound being masked. The effect is strongest between two sounds close in frequency.

In the context of audio transmission, there are some advantages to being unable to perceive a sound. In audio encoding for example, better compression can be achieved by omitting the inaudible tones. This requires fewer bits to encode the sound, and reduces the size of the final file.

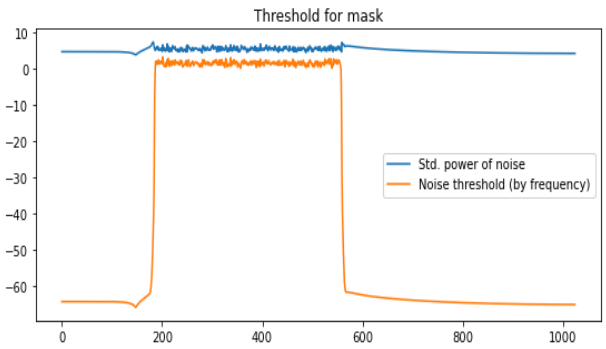
It is uncommon to work with only one tone. Most sounds are composed of multiple tones. There can be many possible maskers at the same frequency. In this situation, it would be necessary to compute the *global masking threshold* using a high resolution Fast Fourier transform via 512 or 1024 points to determine the frequencies that comprise the sound. Because there are bandwidths that humans are not able to hear, it is necessary to know the signal level, masker type, and the frequency band before computing the individual thresholds. To avoid having the masking threshold under the threshold in quiet, one adds the last one to the computation of partial thresholds. This allows computation of the signal-to-mask ratio (SMR).

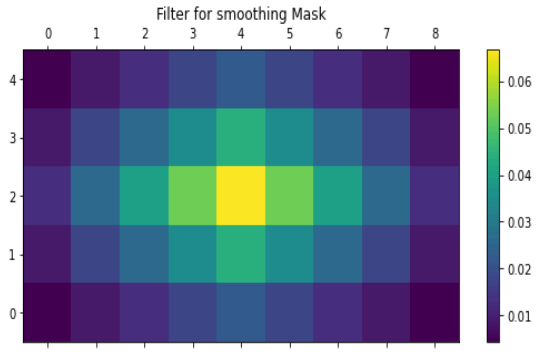


Standard Power of Noise :

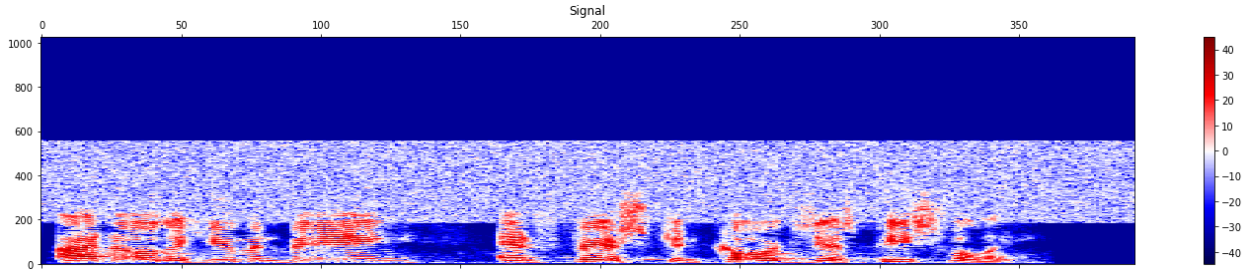


After Threshold Masking :

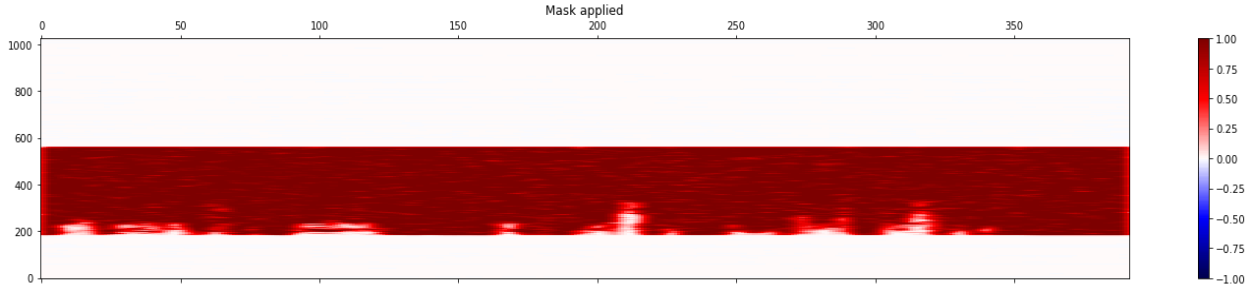




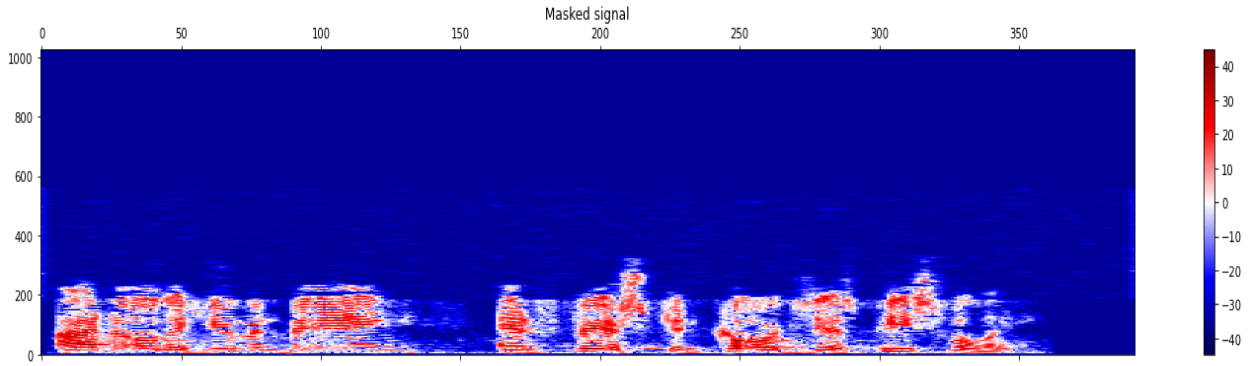
After Addition Of Noise Signal :



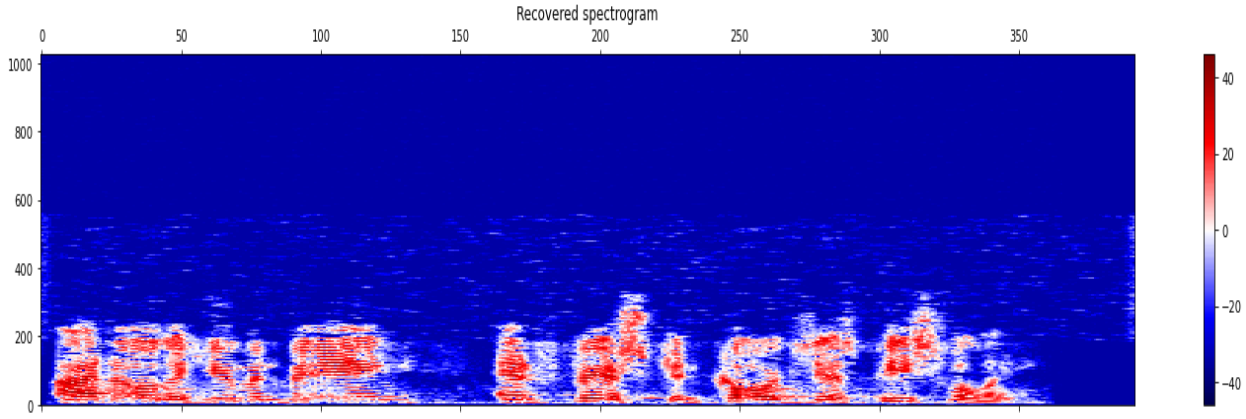
Mask Applied :



Signal After Applying Mask :



Recovered Signal :



Briefing About the code :-

Here these are the libraries which were used for the signal computation .

```
# Importing libraries
import IPython
from scipy.io import wavfile
import scipy.signal
import numpy as np
import matplotlib.pyplot as plt
import librosa
from noisereduce.noisereduce import SpectralGateStationary
import noisereduce as nr
import soundfile as sf
from noisereduce.generate_noise import band_limited_noise
import urllib.request
import io
%matplotlib inline
```

Denoise Data with FFT Algorithm .

```python
def fftnoise(f):
    f = np.array(f, dtype="complex")
    Np = (len(f) - 1) // 2
    phases = np.random.rand(Np) * 2 * np.pi
    phases = np.cos(phases) + 1j * np.sin(phases)
    f[1 : Np + 1] *= phases
    f[-1 : -1 - Np : -1] = np.conj(f[1 : Np + 1])
    return np.fft.ifft(f).real
```

The Band limits Noise to a certain limit wrt the max and min frequency .

```python
def band_limited_noise(min_freq, max_freq, samples=1024, samplerate=1):
    freqs = np.abs(np.fft.fftfreq(samples, 1 / samplerate))
    f = np.zeros(samples)
    f[np.logical_and(freqs >= min_freq, freqs <= max_freq)] = 1
    return fftnoise(f)
```

Adding noise to the data to incense the weight of the training data set .

```
# Adding noise to the original data to increse its size of the training data set read for computation
noise_len = 2 # seconds
noise = band_limited_noise(min_freq=4000, max_freq = 12000, samples=len(data), samplerate=rate)*10
noise_clip = noise[:rate*noise_len]
audio_clip_band_limited = data+noise
```

Applying Short time Fourier Transform

```
def _stft(y, n_fft, hop_length, win_length):
    return librosa.stft(y=y, n_fft=n_fft, hop_length=hop_length, win_length=win_length)
```

Conversion from Amplitude to Decibels and vice versa for the final output Signal .

```
def _amp_to_db(x):
    return librosa.core.amplitude_to_db(x, ref=1.0, amin=1e-20, top_db=80.0)


def _db_to_amp(x,):
    return librosa.core.db_to_amplitude(x, ref=1.0)
```

DeNoise the noise above the threshold limit .

```
def removeNoise(
    audio_clip,
    noise_clip,
    n_grad_freq=2,
    n_grad_time=4,
    n_fft=2048,
    win_length=2048,
    hop_length=512,
    n_std_thresh=1.5,
    prop_decrease=1.0,
    verbose=False,
    visual=False,
):

```

Calculate statistics over noise

```
mean_freq_noise = np.mean(noise_stft_db, axis=1)
    std_freq_noise = np.std(noise_stft_db, axis=1)
    noise_thresh = mean_freq_noise + std_freq_noise * n_std_thresh
```

Calculate value to mask dB

```
mask_gain_dB = np.min(_amp_to_db(np.abs(sig_stft)))
```

Create a smoothing filter for the mask in time and frequency

```
smoothing_filter = np.outer(
        np.concatenate(
            [
                np.linspace(0, 1, n_grad_freq + 1, endpoint=False),
                np.linspace(1, 0, n_grad_freq + 2),
            ]
        )[1:-1],
        np.concatenate(
            [
                np.linspace(0, 1, n_grad_time + 1, endpoint=False),
                np.linspace(1, 0, n_grad_time + 2),
            ]
        )[1:-1],
    )
    smoothing_filter = smoothing_filter / np.sum(smoothing_filter)
```

Plotting the spectrogram with a motive to view the computed signal as the input .

```
def plot_spectrogram(signal, title):
    fig, ax = plt.subplots(figsize=(20, 4))
    cax = ax.matshow(
        signal,
        origin="lower",
        aspect="auto",
        cmap=plt.cm.seismic,
        vmin=-1 * np.max(np.abs(signal)),
        vmax=np.max(np.abs(signal)),
    )
```

Calculate the threshold for each frequency/time bin

```
db_thresh = np.repeat(
        np.reshape(noise_thresh, [1, len(mean_freq_noise)]),
        np.shape(sig_stft_db)[1],
        axis=0,
    ).T
```

Mask if the signal is above the threshold

```
sig_mask = sig_stft_db < db_thresh
```

Convolve the mask with a smoothing filter

```
sig_mask = scipy.signal.fftconvolve(sig_mask, smoothing_filter, mode="same")
    sig_mask = sig_mask * prop_decrease
```

Mask the signal

```
sig_stft_db_masked = (
        sig_stft_db * (1 - sig_mask)
        + np.ones(np.shape(mask_gain_dB)) * mask_gain_dB * sig_mask
    )
```

Apply the mask real

```
sig_imag_masked = np.imag(sig_stft) * (1 - sig_mask)
    sig_stft_amp = (_db_to_amp(sig_stft_db_masked) * np.sign(sig_stft)) + (
        1j * sig_imag_masked
    )
```

Recover the signal

```
recovered_signal = _istft(sig_stft_amp, hop_length, win_length)
    recovered_spec = _amp_to_db(
        np.abs(_stft(recovered_signal, n_fft, hop_length, win_length))
    )
```

Thus you get the reduced noise data signal as an output for your Application . Future Aspects :-

Will try this with the live audio signal with the microphone and Esp32 and thus will get the wav file for the further computation and signal processing .
Also will try with the PyAudio libraries taking Audio signal clip live and then computing FFT on it and the other operations for Noise cancellations .

Applications of Noise Cancellation :-

1. **Noise Cancelling Headphones**
    a. **Automobiles :**One of vital fields of application for noise cancellation would be automobiles. In general any automobile consists of an engine and other moving parts which cause vibrations to build up and are transmitted across the body of the vehicle. One such disturbance is due to noise.
    b. **Medical Scanning**
    Heartbeat sound analysis is a convenient way to diagnose heart disease. Heartbeat sound classification in THE heart sound segmentation and feature extraction. Dataset-B applied in this study contains three categories: Normal, Murmur and Extra-systole heartbeat sound. In the proposed framework, we remove the noise from the heartbeat sound signal by applying the band filter. After that we fixed the size of the sampling rate of each sound signal. Then we applied down-sampling techniques to get more discriminant features and reduce the dimension of the frame rate. However, it does not affect the results and also decreases the computational power and time. Then we applied a purposed model Recurrent Neural Network (RNN) that is based on Long Short-Term Memory (LSTM), Dropout, Dense and Softmax layer. As a result, the proposed method is more competitive compared to other methods.

Conclusion :

In the journey of this project, **AI Noise Reduction,**we went through various learning stages .We faced a lot of issues and difficulties regarding the implementation of **Denoising of the FFT algorithm** and the to get the accuracy for the noise removal and also the manipulation of the data signal according to the demand of the algorithm .
Also we learnt concepts related to Deep learning like logistic regression , activation functions (ReLU,Sigmoid,etc) and gradient descent.
To get a decent idea of the project we went through some research papers.
Thus by taking **FFT** of the noise and the data signal, and by setting frequencies and wavelengths with maximum and frequency limit ,we add the data and noise signal as adding the noise expands the size of the training data set within the limited band.
Also applying STFT,ISTFT and converting from amplitude into dB which inverts amplitude to dB scale. Also applied the mask by setting a threshold to cancel unwanted

frequencies and **convoluted the mask** with a smoothing filter .Thus getting the recovered signal after doing all computations over the given audio signal.

Links for the *research papers* and *youtube links* given below :-

1. **Python-speech-enhancement**
https://github.com/chenwj1989/python-speech-enhancement/blob/master/test_pns.py
upf-smc-speech-enhancement-thesis
https://github.com/eagomez2/upf-smc-speech-enhancement-thesis/blob/be07a0c00bafa1109f07c9791d0dfad505b6f5e9/src/dns_dataset.py

Librosa Doc
https://librosa.org/doc/latest/index.html
Audio processing
(https://opensource.com/article/19/9/audio-processing-machine-learning-python)

Notes made till now :-

https://drive.google.com/drive/folders/1hL5EDf4_YkNlkzSNg5S_fbKVMGUHT5oJ?usp=sharing
Digital math
https://www.youtube.com/watch?v=hewTwm5P0Gg

But what is a Fourier series? From heat flow to drawing with circles | DE4
SOX doc
http://sox.sourceforge.net/sox.html

(Noise Cancellation Method for Robust Speech Recognition)
https://research.ijcaonline.org/volume45/number11/pxc3879438.pdf

Denoising Data with FFT [Python]
https://youtu.be/s2K1JfNR7Sc

Fourier Series [Python]
https://www.youtube.com/watch?v=dZrShAGqT44&list=PLMrJAkhIeNNT_Xh3Oy0Y4LTj0Oxo8GqsC&index=7
Audacity
https://github.com/audacity/audacity/blob/master/src/effects
_Proposal_Noise_Removal
https://wiki.audacityteam.org/wiki/Completed_
Tim sainburg code
https://timsainburg.com/noise-reduction-python.html
methods-for-sound-noise-reduction

https://www.kaggle.com/mauriciofigueiredo/methods-for-sound-noise-reduction#introduction
how-to-build-a-deep-audio-de-noiser-using-tensorflow

https://betterprogramming.pub/how-to-build-a-deep-audio-de-noiser-using-tensorflow-2-0-79c1c1aea299
acoustic-noise-cancellation-by-machine-learning
https://towardsdatascience.com/acoustic-noise-cancellation-by-machine-learning-4144af497661https://jmvalin.ca/demo/rnnoise/
mel-spectrogram
https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0

Playlist of deep learning courses
(https://www.coursera.org/specializations/deep-learning)

Basics of linear algebra (3B1B)
https://youtube.com/playlist?list=PLkDaE6sCZn6Ec-XTbcX1uRg2_u4xOEky0)

Referred the research papers
.http://www.iosrjournals.org/iosr-jece/papers/Vol.%2012%20Issue%205/Version-1/L1205016475.pdf#:~:text=%20adaptive%20noise%20cancellation%20algorithm%20is%20to%20pass,noise%20cancellation%20%28ANC%29%20attenuates%20low%20frequency%20noise%20that

Aaahh ...!!! ..Finally I get to hear a clear audio .....!!!!!!!!!!