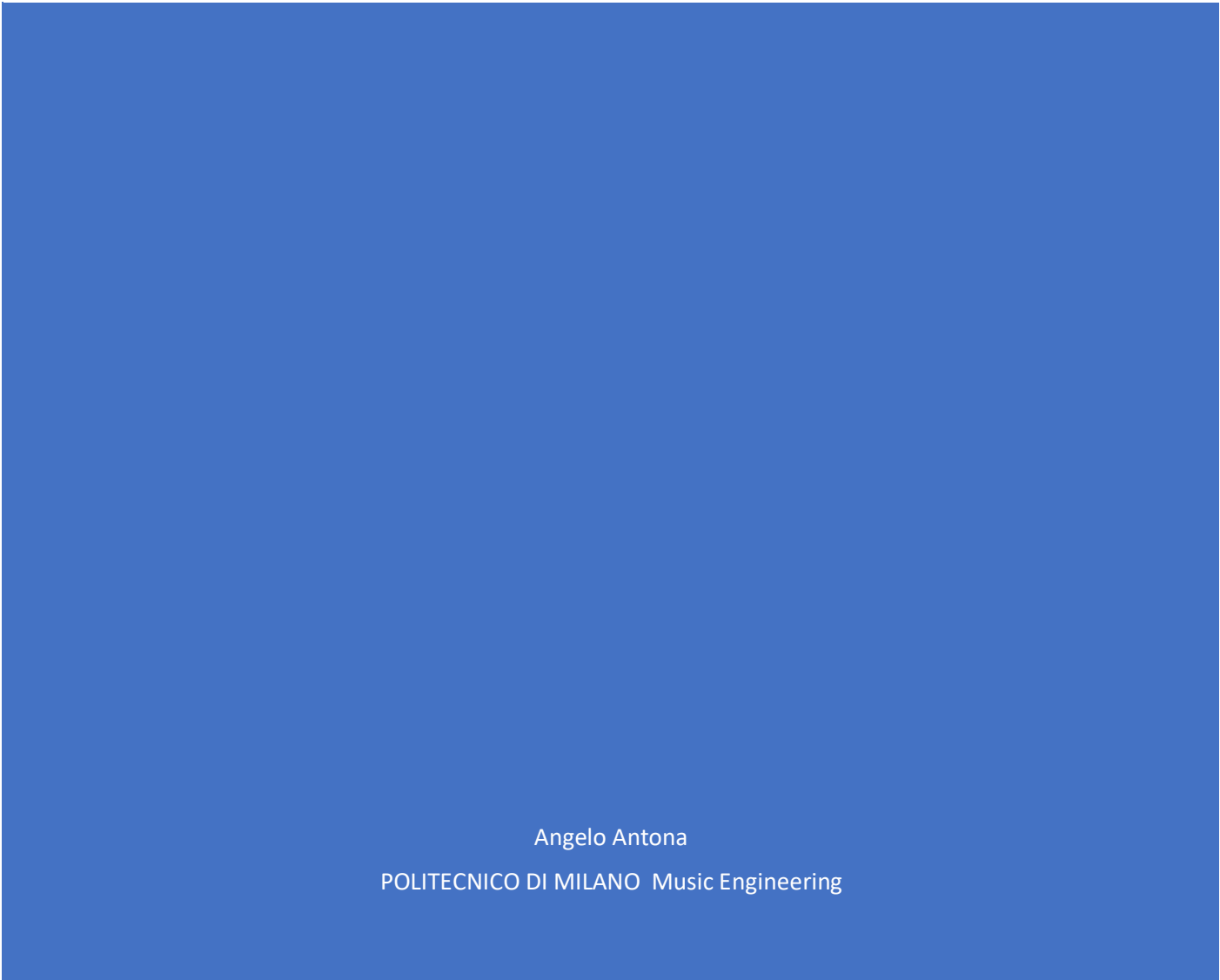




SASP



Angelo Antona
POLITECNICO DI MILANO Music Engineering

1 CONTENTS

2	Sinusoidal Analysis and Modeling.....	4
2.1	Difficulty Matrix.....	4
2.2	Spectral Analysis of Sinusoids.....	4
2.2.1	Choice of Window Length M	5
2.3	Spectral Analysis of Harmonic Signals	5
2.3.1	Resolution of Harmonic Signals (M for Harmonic Signals).....	5
2.4	Spectral Interpolation.....	6
2.4.1	Zero Padding Interpolation	6
2.4.2	Parabolic Interpolation	7
2.5	Additive Synthesis Analysis Techniques	8
2.5.1	Driving a Bank of Oscillators Using STFT Data.....	8
2.5.2	Driving a Bank of Oscillators using Filterbank Data.....	10
2.6	Vocoders.....	11
2.6.1	Channel Vocoders	12
2.6.2	Phase Vocoders	12
2.7	Filtering in the Frequency Domain	14
2.7.1	Linear Time-Invariant Systems	14
2.7.2	Overlap and Add Filtering	14
2.7.3	Wiener Filters	15
3	Audio Restoration	18
3.1	Audio Signal Models	18
3.1.1	AR Model.....	18
3.1.2	ARMA Model	18
3.1.3	Sinusoidal Model.....	19
3.2	Wiener Filter.....	19
3.2.1	Wiener Filter Response	20
3.2.2	Filter's Bandwidth and Action	21
3.2.3	Implementation of the Wiener Filter	21
3.2.4	Wiener Filter Performances	21
3.2.5	Characterization of the Power Spectral Densities (PSD)	22
3.3	Power Spectral Matching	22
3.4	Parametric Wiener Filtering	23
3.5	Background Noise Reduction	23
3.5.1	Short-Time Spectral Attenuation (STSA)	24
3.5.2	Noise Suppression Rules	24
3.5.3	Dereverberation.....	25
3.5.4	Error Concealment	26
3.6	Nonlinearity Effect on Signal Statistics.....	27

3.6.1	PDF Shaping	27
3.6.2	DBX.....	28
3.6.3	Dolby Noise Reduction (NR)	28
3.7	Restoration of local degradations	29
3.7.1	Restoration Process Steps.....	30
3.7.2	Clicks Detection.....	31
3.7.3	Parameter Estimation (Maximum Likelihood (ML) estimation method)	31
3.7.4	Signal Estimation	32
4	Linear Predictive Coding	34
4.1	Data Model.....	34
4.2	Source – Filter Interpretation	34
4.3	Determine the Predictor Coefficients.....	35
4.3.1	Minimum MSE.....	35
4.3.2	Prediction Gain.....	36
4.3.3	Infinite memory linear prediction	36
4.4	Implementation: dealing with non-stationary signals.....	37
4.5	Power Spectrum Envelope Matching	37
4.6	Frequency Selective LPC.....	38
4.7	Applications of LPC.....	39
5	Time and Pitch Scaling	39
5.1	Ideal Time Scaling.....	40
5.2	Ideal Pitch Scaling.....	40
5.3	Ideal Combined Time and Pitch Scaling.....	40
5.4	Duality Principle.....	40
5.5	Methods	41
5.6	Parametric Methods.....	41
5.7	Non-Parametric Methods (Time Domain)	42
5.7.1	Pitch-Independent Methods	42
5.7.2	Modified Tape Recorder	42
5.7.3	SOLA	44
5.7.4	PSOLA.....	44
5.7.5	Remarks on Time Domain Methods.....	46
5.8	Non-Parametric Methods (Frequency Domain)	46
5.8.1	STFT Based Algorithm (Time Scaling)	46
5.8.2	STFT Based Algorithm (Pitch Scaling)	48
6	Applications of Adaptive Filtering (Wiener Filter)	48
6.1	Noise Cancellation.....	48
6.2	Acoustic Echo Cancellation.....	49
6.3	Background Noise Removal.....	49

7	Features and Classification	49
7.1	Sound Classification.....	50
7.2	Audio Features	50
7.2.1	Basic Descriptors	50
7.2.2	Harmonicity Descriptors	50
7.2.3	Timbral Descriptors.....	51
7.2.4	Other Descriptors.....	51
7.3	Classification Methods	51
8	Microphone Arrays: Spatial Methods For DOA Estimation (Narrow Band Models)	51
8.1	Data Model.....	52
8.1.1	Single Source & Single Sensor	52
8.1.2	Single Source Multiple Sensors (Array)	53
8.1.3	Multiple Sources Multiple Sensors.....	54
8.2	Uniform linear array (ULA)	54
8.2.1	Spatial Frequency.....	54
8.2.2	Anti-aliasing Condition in ULA.....	55
8.2.3	DOA Estimation	55
8.2.4	Spatial filtering (Non-parametric methods)	55
8.2.5	Parametric Methods	57
8.2.6	MUSIC – Multiple Signal Classification (Parametric method)	59
8.2.7	ESPRIT (Parametric method)	59
9	Microphone Arrays: Localization of Wide-Band Sources.....	60
9.1	Data Model.....	60
9.1.1	Time Difference of Arrival	61
9.1.2	From TDOA to DOA	61
9.1.3	TDOA Estimation: Cross-Correlation	62
9.1.4	TDOA Estimation: Adaptive Eigenvalue Decomposition (AED)	62
10	Microphone Arrays: Acoustic Source Separation	63
10.1	Source Separation : Binary Masks	64
10.2	Source Separation: Independent Component Analysis.....	65
11	MATLAB	66
11.1	Definition Of a Sinusoidal Signal.....	66
11.2	Modelling the Noise	67
11.3	FIR Filtering.....	67
11.3.1	FIR filtering via Time-Domain Convolution	67
11.3.2	FIR filtering via Frequency-Domain Convolution	68
11.3.3	Filtering in Time/Freq domain (MATLAB)	69
11.3.4	Overlap And Add.....	69

2 SINUSOIDAL ANALYSIS AND MODELING

Sinusoidal modeling is a technique used in signal processing and audio synthesis where complex signals are represented as the sum of simpler sinusoidal components (sine waves). A sinusoidal wave is described by its frequency (how fast it oscillates), amplitude (how loud it is), and phase (where in its cycle it starts).

A large class of musical sounds can be efficiently modeled as sums of sinusoidal components. This means that complex musical tones can be broken down into simpler sine waves at different frequencies, amplitudes, and phases.

The process of modeling sounds is divided into several steps:

1. **Analysis:** Determining the components of the sound. Two methods are possible:
 - **Short Time Fourier Transform (STFT):** This method is preferable when the frequency content of the sound is of more interest than the exact timing of these frequencies.
 - **Filter bank:** This approach is used when the timing of the sound components is more important than the exact frequencies.
2. **Data Reduction:** An optional step that may involve reducing the amount of data to be processed, potentially by eliminating redundant or unnecessary information.
3. **Modification:** Another optional step where the sound can be altered or processed to achieve a desired effect.
4. **Synthesis:** Reconstructing the sound from its components. Two methods are possible:
 - **Inverse STFT (I-STFT):** This is used when the frequency content is more important in the reconstruction phase.
 - **Bank of Oscillators:** This method is used when the timing of the components is more important in the reconstruction.

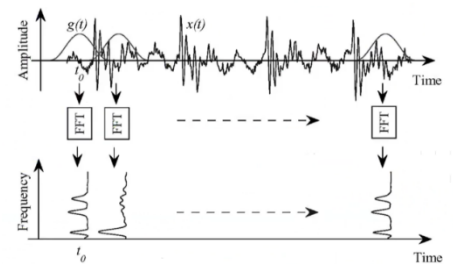


Figure 1

2.1 DIFFICULTY MATRIX

The difficulty matrix (figure 2) indicates the relative difficulty of synthesis using either the Inverse STFT or a bank of oscillators, given the STFT or a filter bank as the analysis method.

2.2 SPECTRAL ANALYSIS OF SINUSOIDS

When we analyse a sinusoid, we approximate it as a sum of sinusoids $s_s(t) = \sum_k [a_k(t) * \cos(\phi(t))]$, defining the number of sinusoids, their frequency, amplitude, and phase. The problem in doing this is the fact that the signals that we process are finite (windowed), and this makes more complex our analysis.

Let's consider a single sinusoid: its Fourier transform is ideally a pulse at the frequency of the sinusoid (and its relative negative frequency pulse). When we crop the sinusoid, we are multiplying the sinusoid for a rectangle, which Fourier transform is a sinc $\left(\frac{\sin(x)}{x}\right)$. We know that if we multiply two signals in time domain, we are also convolving them in the frequency domain: the result is a sinc centred in the position of the pulse.

Analysis $\Rightarrow$ Synthesis $\Downarrow$	STFT	Filterbank
I-STFT	Easy	Hard
Bank of Oscillators	Hard	Easy

Figure 2

We obtain a similar result when we crop the signal with a “softer” window, as showed in *figure 3*:

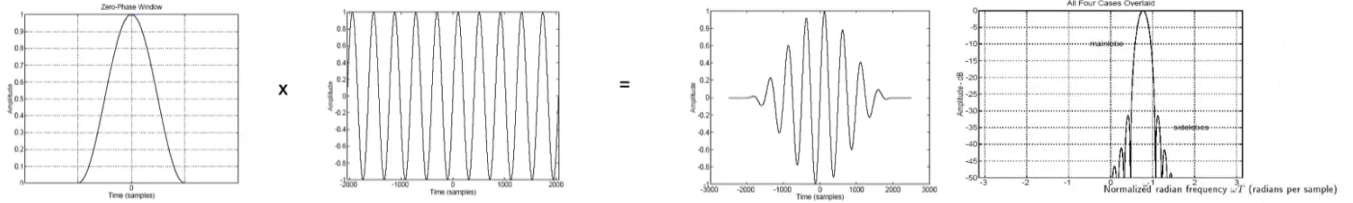


Figure 3

This effect negatively impact on the detection of the number and of the frequencies of the component sinusoids.

For example, if we window two cosines with a rectangle window of M samples, the frequency spectrum that we obtain changes based on the number of samples of the window:

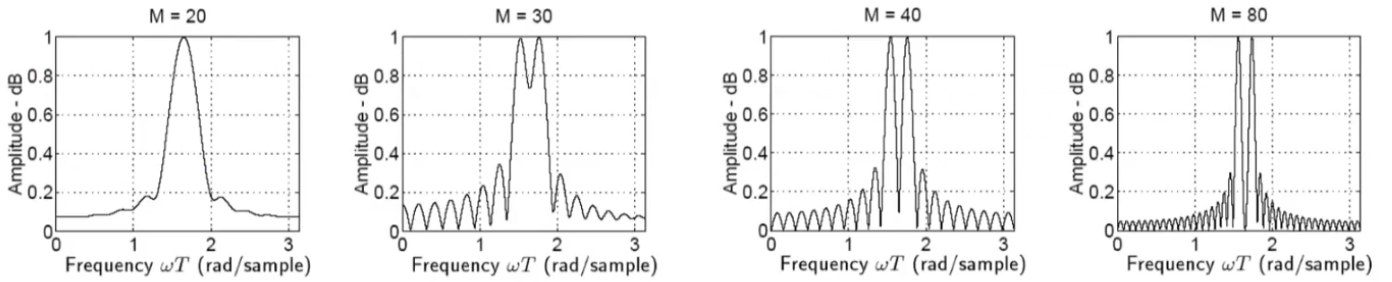


Figure 4

2.2.1 Choice of Window Length M

We can easily see that with a smaller number of samples (plot on the left in *figure 5*) we cannot detect the frequencies of the two cosines (the frequency of the pulses of the cosines), and that increasing the number of samples, we obtain a more precise representation of the two pulses. This makes us understand that it's important to choose a window length M that allows for the main lobes of their spectral representations to be distinct. In the context of noise, this requirement becomes more stringent to ensure the two components can be individually identified.

A correct window length must verify the following inequality:

$$B_W \leq \Delta \text{ where } \begin{cases} B_W = \frac{L * F_s}{M} = \text{main lobe width (L = type of window (Rect} \rightarrow L = 2; \text{Hamm} \rightarrow L = 4, \dots)) \\ \Delta = \text{minimun distance between the frequencies.} \end{cases}$$

Hence is necessary that $M \geq L * \left(\frac{F_s}{f_2 - f_1} \right)$.

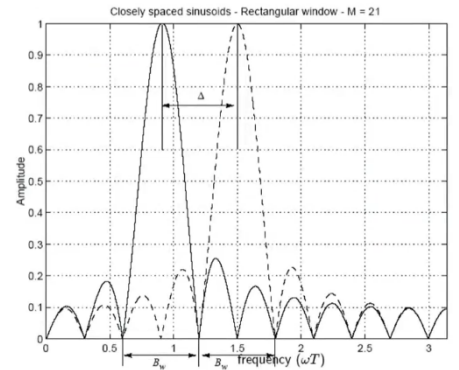


Figure 5

2.3 SPECTRAL ANALYSIS OF HARMONIC SIGNALS

A harmonic signal is a periodic signal composed of a fundamental frequency and its integer multiples. The fundamental frequency, often denoted as f_0 , is the lowest frequency of the signal and is also considered the first harmonic. The frequencies that are integer multiples of the fundamental frequency are known as the higher harmonics. For example, if the fundamental frequency is f_0 , then the second harmonic would be $2 * f_0$, the third harmonic would be $3 * f_0$, and so on.

2.3.1 Resolution of Harmonic Signals (M for Harmonic Signals)

When we spectral analyse an harmonic signal, we want to accurately distinguish and measure the individual harmonic components within the signal.

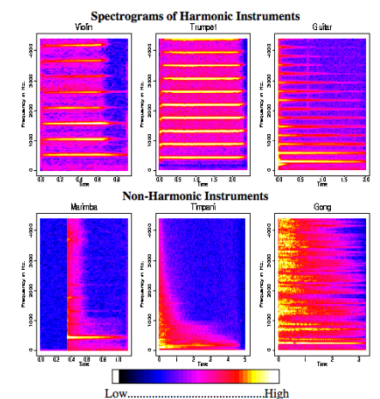


Figure 6

When two harmonic components are very close to each other in frequency, it might be challenging to see them as two separate entities. They might appear as a single peak or be too blurred together to accurately determine their individual characteristics. Resolving them would mean adjusting the window length in the FFT so that these components appear as separate peaks, allowing to measure the amplitude, frequency, and phase of each harmonic accurately.

Considering a harmonic signal with fundamental frequency f_0 , it's necessary to use a minimum window length M of

$$M \geq \frac{L * F_s}{f_0} = L * P \text{ where } \begin{cases} \frac{F_s}{F_0} = \frac{T_0}{T_s} = P \end{cases} \text{ that is the period of the fundamental component in samples.}$$

L is the parameter of the window type.

We have seen that, given a signal, if we know:

- the minimum distance between possible frequency peaks in the frequency domain (*this is the fundamental frequency if the signal is harmonic*),
- the window function that we must use (we need the L parameter),

we can calculate how long (in samples) the window should be in order to be able to correctly detect the frequency peaks.

2.4 SPECTRAL INTERPOLATION

Spectral interpolation is a technique used in signal processing to estimate the frequency of a spectral peak more accurately from a Fast Fourier Transform (FFT). When you perform a FFT on a sampled signal, you obtain a spectrum that shows the frequency content of the signal. However, the actual frequency of a spectral peak may not coincide exactly with one of the discrete frequency bins produced by the DFT, because these bins are spaced at regular intervals determined by the sampling rate and the number of points in the FFT (*figure 7*).

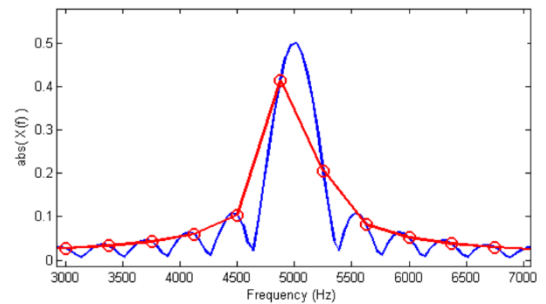


Figure 7

We will see two main types of interpolation: zero padding and parabolic interpolation. These techniques are explained more in detail in the next paragraphs.

2.4.1 Zero Padding Interpolation

[Zero padding interpolation](#) consists in zero-pad the end of the signal in time domain before the FFT is performed. This operation is equivalent to interpolating between the frequency bins in the frequency domain. **It doesn't increase the true resolution of the signal** (in other words, if the two sinusoids peaks are superimposed, after the interpolation they will remain superimposed, meaning that it's not useful to apply this kind of techniques if we didn't chose the correct size M of the window, like showed in the *left plot of figure 9*) but gives more points between the bins (*figure 8*), which helps in more precisely locating the peaks in the spectrum, allowing to get a more accurate estimate of the peak's frequency.

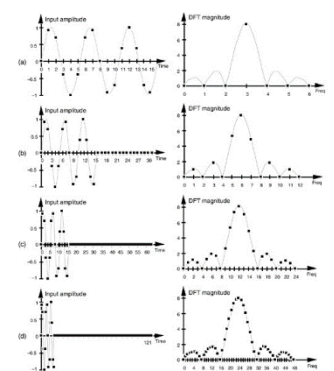


Figure 8

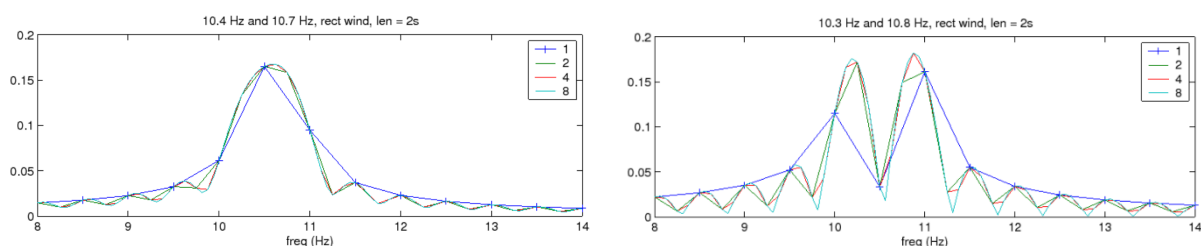


Figure 9

The error in locating the peak frequency is half the distance between consecutive frequency bins, given by

$$\Delta f = \pm \frac{F_s}{2N} \text{ where } \begin{cases} F_s \text{ is the sampling frequency.} \\ N \text{ is the number of points in the FFT after zero padding.} \end{cases}$$

Knowing the maximum error Δf that we get based on the number of points in the FFT after zero padding, we can choose the precision of our analysis taking advantage of the human perception of sound.

When we perform pitch detection, for example, we can consider the Just Noticeable Difference (JND = minimum variation of pitch that the human ear is able to perceive at a specific frequency and SPL) and pad the signal with a number of zeros that makes the detection error smaller of the JND, so that the listener will not perceive it.

JND = Just Noticeable Difference
 $\Rightarrow$ minimum frequency change in a pure tone (sinusoid) that we are able to perceive
 A good approximation for the JND is
 $JND = 0.5\% f$, for $f \geq 600$ Hz
 $JND = 3$ Hz, for $f < 600$ Hz

2.4.2 Parabolic Interpolation

Parabolic interpolation uses a quadratic function to estimate the true peak of a spectral component within the FFT bins. The method assumes that the shape of the main lobe of a spectral peak can be approximated by a parabola. The parabola is defined by three points: the highest amplitude point in the FFT (the peak bin) and the points immediately to the left and right of this peak bin (figure 10). This parabola is of the form:

$$y(x) \cong a(x - p)^2 + b$$

where $\begin{cases} y(x) \text{ is the magnitude of the FFT at sample } x. \\ a \text{ is the curvature of the parabola.} \\ b \text{ is the peak amplitude.} \\ p \text{ is the distance of the actual peak from the peak sample.} \end{cases}$

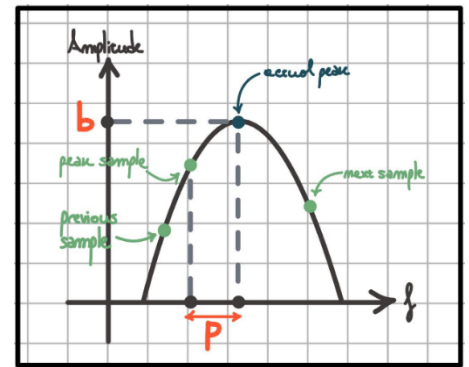


Figure 10

Using the known values of the FFT at the peak bin and its neighbours, we can solve for the coefficients a , b , p and (skipping all the mathematical passages) we get that the value of the offset of the peak sample from the actual peak is:

$$p = \frac{1}{2} * \frac{(\alpha - \gamma)}{\alpha - 2 * \beta + \gamma} \text{ where } \begin{cases} \alpha = y(-1) = \text{previous sample.} \\ \beta = y(0) = \text{peak sample.} \\ \gamma = y(1) = \text{next sample.} \end{cases}$$

The value of p is in the range $[-\frac{1}{2}, \frac{1}{2}]$ because it corresponds to the “peak detection error” and, as we have seen, it is equal to half the distance between two samples.

Knowing p , we can also calculate the estimated peak height as:

$$y(p) = \beta - \frac{1}{4} * (\alpha - \gamma) * p$$

Notes: parabolic interpolation is particularly effective and precise when interpolating the spectral samples of a Gaussian window on a dB scale.

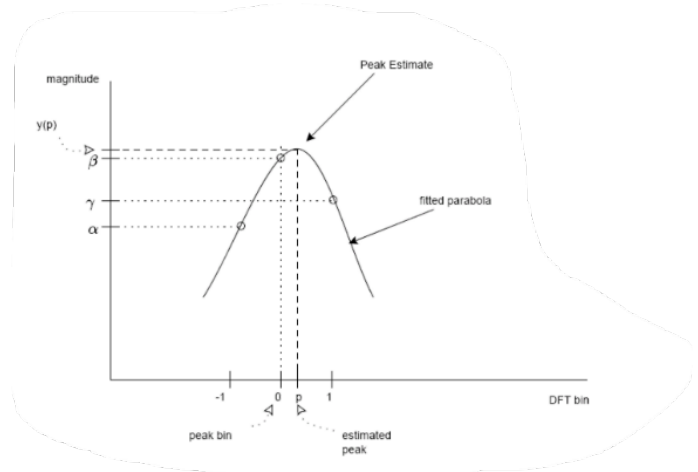


Figure 11

2.5 ADDITIVE SYNTHESIS ANALYSIS TECHNIQUES

2.5.1 Driving a Bank of Oscillators Using STFT Data

In this paragraph we will see how we can synthesize audio signals by driving a bank of oscillators with data obtained from a Short-Time Fourier Transform (STFT) analysis.

The situation in which we are is the following (illustrated in the third quadrant of the plot in [figure 2](#)):

1. **The analysis of the signal is performed using the STFT:** the STFT breaks down the signal into its frequency components over short time frames, providing a time-frequency representation of the signal.
2. **We want to perform the synthesis using a bank of oscillators:** each oscillator corresponds to a sinusoidal component of the signal.

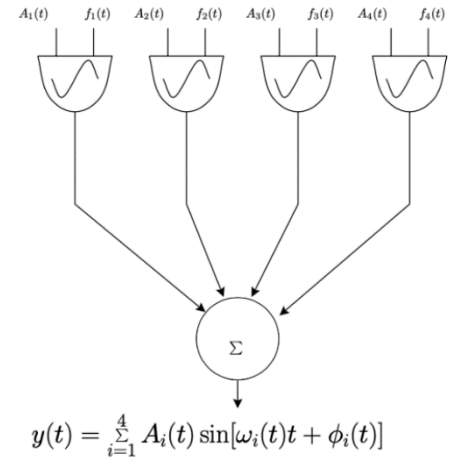


Figure 12

The problem in doing this is that there is a domain mismatch, which is a disconnect between the analysis and synthesis stages. To overcome this domain mismatch, spectral continuation is needed. It involves:

- **Peak detection and matching:** [Identifying the peaks](#) in the frequency domain and matching them across successive frames of the STFT to track the evolution of each frequency component.
- **Peak interpolation:** [Estimating the exact frequency of each peak](#), which may lie between the discrete frequency bins of the STFT.
- **Generating frequency and amplitude trajectories:** creating smooth trajectories of frequency and amplitude for each sinusoidal component over time, which will drive the oscillators during synthesis.

2.5.1.1 Additive Synthesis (Bank of Oscillators)

Additive synthesis is a technique where a complex signal is reconstructed from the summation of sinusoidal components, each with a time-varying amplitude and frequency. The general formula for additive synthesis is given by:

$$y(t) = \sum_{i=1}^N A_i(t) * \sin(\omega_i(t) * t + \phi_i(t)) \text{ where } \begin{cases} A_i(t) \text{ is the amplitude envelope of the } i_{th} \text{ sinusoid.} \\ \omega_i(t) \text{ is the instantaneous angular frequency.} \\ \phi_i(t) \text{ is the instantaneous phase.} \\ N \text{ is the total number of sinusoidal components.} \end{cases}$$

To accurately reconstruct the signal, we must first analyse it to determine the amplitude and frequency trajectories for each sinusoidal component. In order to do that, we use the [Sinusoidal Peak Tracking processing](#) technique.

2.5.1.2 Sinusoidal Peak Tracking

Tracking Sinusoidal Peaks in a sequence of FFTs is a process that's particularly suited to signals with a rich harmonic content, like those from a piano. The approach focuses on discerning the significant features within the spectral content of a signal, namely the peaks which correspond to the fundamental tone and its harmonics, rather than examining every detail across the full spectrum as filter banks might.

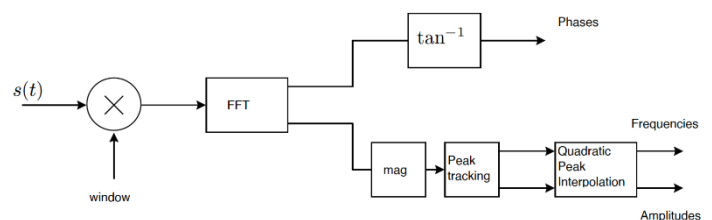


Figure 13

Practically, peak tracking (across frames) involves following the trajectory of individual spectral peaks, each representing a sinusoidal component of the sound, over successive time frames of the signal.

The process involves the following steps:

- **Analysis:** we start with the analysis process (*figure 13*):

1. **STFT:** the process starts by performing the Short-Time Fourier Transform (STFT). It involves:
 - **Slice the signal into frames:** the signal is sliced into shorter segments that are small enough to consider the signal within them as stationary.
 - **Window the slices:** each frame is multiplied by a window function to reduce spectral leakage (*perdita*).
 - **Perform FFT:** FFT is performed to each frame.

Conceptually, this process can be thought of as applying a filter bank, where each FFT bin corresponds to a different filter capturing a specific frequency component.

2. **Identify peaks:** for each frame, a non-coherent estimation of sinusoidal parameters from the signal's magnitude spectrum is performed. This means we're identifying the peak amplitude and center frequency of the signal components without necessarily considering the phase relationship between them. To minimize the error in frequency peaks identification, we interpolate the frequency spectrum of each frame. Two methods of interpolation are commonly used in peak tracking:
 - **Linear Interpolation** is used when phase information is not critical, which is often the case with steady-state signals. This method interpolates between the instantaneous amplitudes and frequencies of the peaks to create a smooth transition from frame to frame.
 - **Cubic Phase Interpolation** is employed when phase information is important, especially in frames containing transients. This method provides a more accurate representation of the phase evolution, which is vital for accurately reconstructing the time-domain signal.

3. **Peak Tracking:** Peak tracking is then applied to follow these spectral components across frames. When doing peak tracking, it's crucial to maintain a coherent trajectory of each peak from one frame to the next, effectively creating a continuous line that represents the evolution of that particular frequency component over time. Peaks are associated from frame to frame by finding the closest peak in the subsequent frame that matches the frequency trajectory and amplitude characteristics (*figure 14*).

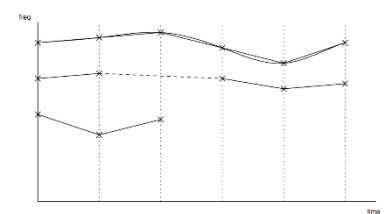
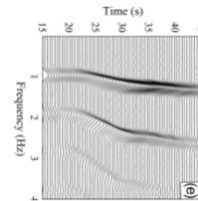


Figure 15

A key challenge in peak tracking is accurately identifying the 'birth' and 'death' of peaks (*figure 15*), which represent the points in time where new spectral components appear, or existing ones vanish.



Figure 14

This can be complicated by noisy spectra, where distinguishing between actual signal components and noise can be difficult. To address this, a transient detector or additional methods such as examining the differentiated amplitude envelope of a high-pass filtered signal or using linear prediction error may be used to improve the robustness of peak tracking.

A smart choice for effective peak tracking could be consider the signal's harmonicity, which can help in associating peaks across frames even when the fundamental frequency is not explicitly present. It's also convenient to avoid hasty (*frettolose*) decisions about the end or beginning of a peak's trajectory, as gaps might appear in the tracking due to various reasons, such as masking by other components or temporary dips in amplitude below the threshold of detection.

- **Synthesis:** it is the process of reconstructing a signal from its constituent sinusoidal components and noise. It involves the following steps:

1. **Creation of the sinusoidal component starting from the analysis**

output parameters: the synthesis begins by using a bank of amplitude and phase-modulated oscillators. Each oscillator corresponds to a sinusoidal component identified during the analysis phase. The oscillators are controlled by three main parameters:

- **Amplitude ($A(t)$):** the amplitude of each oscillator varies over time. It is interpolated linearly between frames to create a smooth amplitude envelope.
- **Frequency ($f(t)$):** the frequency of each oscillator also varies over time, following the trajectory of the tracked sinusoidal peaks across the analysis frames.
- **Phase ($\phi(t)$):** the phase is more complex to handle due to its cyclical nature. Phase unwrapping is often necessary to deal with the 2π discontinuities. The unwrapped phase is then interpolated to provide a continuous phase trajectory for each oscillator.

These parameters are then fed into the oscillators to generate sinusoidal waveforms.

2. **Sum of the sinusoidal components:** the sum of these sinusoidal components results in the deterministic part of the signal. This is illustrated by the equation:

$$y(t) = \sum_{i=1}^k A_i(t) * \sin(\omega_i(t) * t + \phi_i(t)) \text{ where } \begin{cases} y(t) = \text{synthesized signal.} \\ A_i(t) = \text{time varying amplitude of the } i_{\text{th}} \text{ component.} \\ \omega_i(t) = \text{angular frequency.} \\ \phi_i(t) = \text{instantaneous phase.} \end{cases}$$

3. **Introduce not deterministic component:** for the noise component, which represents the stochastic part of the signal, white noise is filtered through a time-varying filter whose impulse response is $h(t)$. This filtered noise is then added to the sum of the sinusoidal components to form the complete signal:

$$s(t) = \sum_{r=1}^R A_r(t) * \cos(\theta_r(t)) + e(t)$$

where the noise component $e(t) = \int_{-\infty}^{\infty} h_t(t - \tau) * u(\tau) d\tau$ with $u(\tau) = \text{white noise}$.

2.5.1.3 Complete analysis and synthesis plots for FFT → Filterbank case

The complete process of additive synthesis analysis is showed in the following schemes:

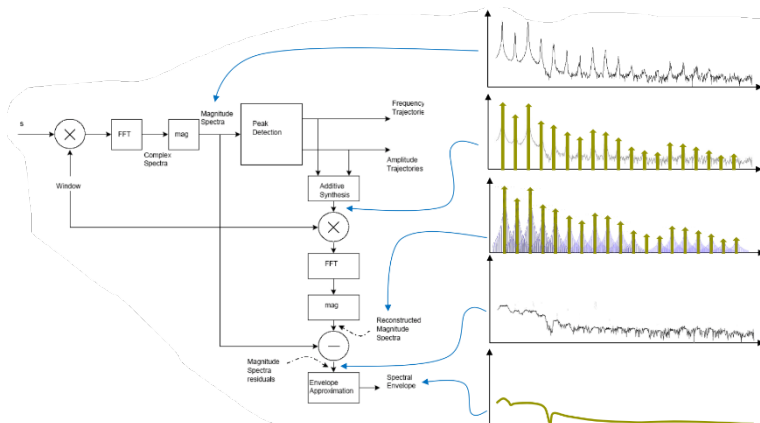


Figure 18 - Analysis scheme.

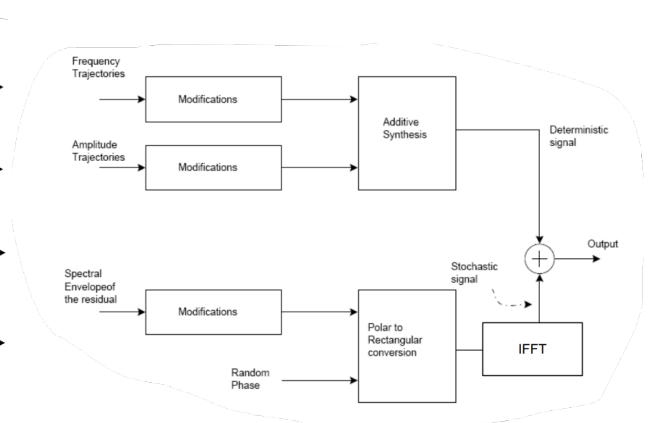


Figure 18 - Synthesis scheme.

2.5.2 Driving a Bank of Oscillators using Filterbank Data

In this case we are in the fourth quadrant of [figure 2](#). This process involves the following steps:

- **Analysis:** the analysis of the signal is performed using a filterbank. A filterbank consists of multiple band-pass filters that divide the input signal into several components, each corresponding to a specific frequency band.

- **Synthesis:** The synthesis of the signal is based on a bank of oscillators. Each oscillator in the bank is designed to recreate a single sinusoidal component of the sound, and collectively, they reconstruct the original signal. The data to drive these oscillators is derived from the filterbank analysis. Synthesis using a bank of oscillators is “easy” because once the filterbank analysis data is available, driving the oscillators and creating the signal is a relatively straightforward process.

2.6 VOCODERS

Vocoders, short for "Voice Coders," are a category of electronic devices or software that can analyze and synthesize the human voice or other complex sounds.

Vocoders function as an analysis/synthesis system, with the following key components:

- **Analysis:** the signal $x(t)$ is decomposed by a filterbank into multiple frequency bands. This process captures the essential characteristics of the signal within each frequency band, represented by the amplitude of the signal within each band.
- **Processing:** the analysed data can be manipulated for various purposes, such as data compression, which is vital for efficient storage and transmission, or noise reduction to improve signal quality.
- **Synthesis:** the manipulated data is then used to drive a bank of oscillators, which reconstruct the signal by synthesizing each frequency band's components.

Vocoders often assume that within each channel or sub-band, there is no more than one sinusoid present at a time. This simplifies the synthesis model, allowing for the expression of the signal in the k_{th} subband as:

$$x_k(t) = A_k(t) * \cos(\omega_k * t + \phi_k(t)) \text{ where } \begin{cases} A_k(t) \text{ is the amplitude modulation term.} \\ \omega_k \text{ is the fixed center frequency of the channel.} \\ \phi_k(t) \text{ is the phase modulation term.} \end{cases}$$

These parameters enable the reconstruction of the signal through the summation of the output of all oscillators in the bank.

The instantaneous phase modulation $\phi_k(t)$ is differentiated to find the instantaneous frequency deviation, which provides the exact frequency content within each band over time: $\Delta\omega_k(t) = \frac{d\phi_k(t)}{dt}$. This differentiation helps in capturing the fine variations in frequency that occur in the natural speech or in complex audio signals.

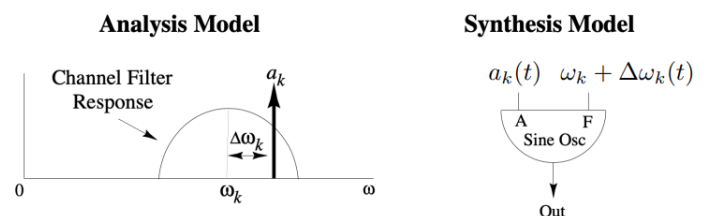


Figure 19

The synthesis model for a phase vocoder uses the instantaneous amplitude $a_k(t)$ and the instantaneous frequency $\omega_k + \Delta\omega_k(t)$ to drive each sine oscillator, allowing for a detailed and dynamic recreation of the original sound.

There are two main types of vocoders:

- **Channel Vocoder:** it determines only the magnitude of the signal within each filter bank. This type of vocoder is often used to create the [classic robotic voice effect](#) by modulating the amplitude of each band's oscillator with the input signal's corresponding amplitude.
- **Phase Vocoder:** in addition to the magnitude, it also determines the phase of the signal in each filter bank. This allows for a more accurate reconstruction of the original signal, preserving both the tonal and rhythmic content.

We will describe them in detail in the next paragraphs.

2.6.1 Channel Vocoders

Channel vocoders are a type of vocoder that synthesize speech by extracting and then reapplying the amplitude envelopes of the signal's frequency bands. The key to a channel vocoder's operation lies in its ability to compute these amplitude parameters accurately.

The process for computing the parameters in a channel vocoder (and so, only the amplitude) involves the following steps:

1. **Rectification:** each sub-band signal $x_k(t)$ is rectified. Rectification is a process that converts all negative values of the waveform to positive, typically by taking the absolute value of the signal. This creates a waveform that oscillates above the zero line without changing sign.
2. **Envelope Detection:** the rectified signal is then passed through a low-pass filter. The purpose of the low-pass filter is to average out the rapid fluctuations in the rectified signal, effectively extracting its envelope. This envelope corresponds to the slow variations in amplitude that carry the important information about the spectral energy distribution over time.

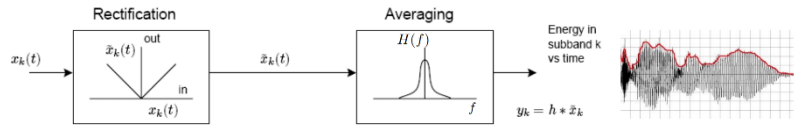


Figure 20

The envelope follower for each sub-band tracks the amplitude changes of that particular band. In synthesis, these changing amplitudes modulate a carrier signal, a process that recreates the natural dynamics of the original sound. The carrier in a channel vocoder is typically a noise source or a set of sine waves that correspond to the center frequencies of the sub-band filters (see figure 21).

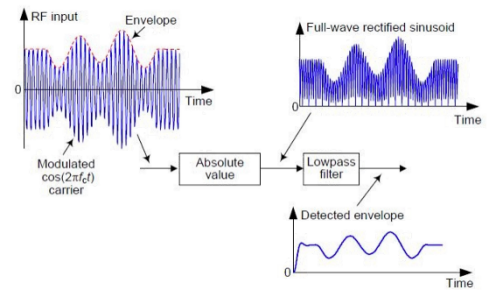


Figure 21

2.6.2 Phase Vocoders

Phase vocoders are a type of vocoder that captures both the amplitude and phase information from a signal for analysis and synthesis. Here's how they work:

- **Analysis:**
 1. **Sub-band Decomposition:** the signal is split into sub-bands using a filterbank. In the case of a phase vocoder, we need to determine both the amplitude and phase of the signal in each sub-band. Assuming no more than one sinusoid per sub-band, the signal in each channel can be represented as:

$$x_k(t) = a_k(t) * \cos(\omega_k * t + \phi_k(t)) \text{ where } \begin{cases} \omega_k \text{ is the center frequency of the } k_{th} \text{ channel.} \\ a_k(t) \text{ is the instantaneous amplitude.} \\ \phi_k(t) \text{ is the instantaneous phase modulation.} \end{cases}$$

2. **Parameter extraction:** we want to extract the instantaneous amplitude $a_k(t)$ and the instantaneous phase $\phi_k(t)$. It is not convenient to directly get these two parameters separately; it is smarter to get the analytic signal correspondent to the sub-band, and calculate the two parameters that we need from this transformed signal. The analytic signal $x_k^a(t)$ is created by suppressing the negative frequency components of $x_k(t)$ and doubling the magnitude of the positive frequency components. This is typically achieved by taking the original signal and adding to it an imaginary component which is the Hilbert transform of the original signal. The analytic signal is therefore a complex signal where:
 - The **real part** is the original signal $x_k(t)$.
 - The **imaginary part** is the **Hilbert transform**. The Hilbert transform is a linear operator that takes a signal $x(t)$ and produces a signal $\tilde{x}(t)$ that is the original signal with a -90 degree phase shift (without modify amplitude). In the

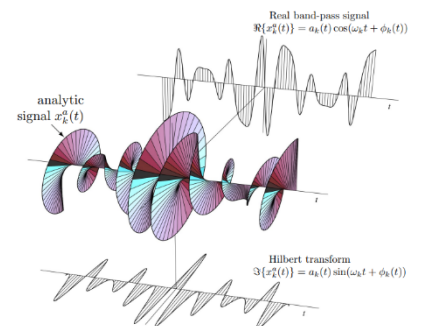


Figure 22

frequency domain, the Hilbert transform shifts the phase of positive frequency components by -90 degrees and the phase of negative frequency components by $+90$ degrees, effectively discarding the negative frequency components. The frequency response of the Hilbert transform is given by $-j * \text{sgn}(f)$, which denotes this phase shift property.

- **Synthesis:**

1. The analytic signal is used to extract the amplitude and phase information:

- **The amplitude $a_k(t)$** of the signal can be directly obtained by taking the magnitude of the analytic signal.
- **The phase $\phi_k(t)$** can be extracted by computing the angle of the analytic signal, which is the arctangent of the imaginary part over the real part, adjusted by the known carrier frequency ω_k :

$$\phi_k(t) = \tan^{-1} \left(\frac{\text{Im}\{x_k^a(t)\}}{\text{Re}\{x_k^a(t)\}} \right) - \omega_k * t$$

The $\tan^{-1}$ function is not so convenient because it has issues such as computational complexity and phase wrapping. A much better idea is to work with the instantaneous frequency deviation $\Delta\omega_k(t)$ instead of the phase. This instantaneous frequency deviation is the derivative of the phase over time, which gives us the rate of change of the phase, and we can calculate it as:

$$\Delta\omega_k = \frac{d\phi(t)}{dt} = \frac{x\dot{y} - \dot{x}y}{x^2 + y^2} \text{ where } \begin{cases} x = \text{Re}\{x_k^a(t)\} \\ y = \text{Im}\{x_k^a(t)\} \end{cases}$$

A complete phase vocoder scheme is present in *figure 23*.

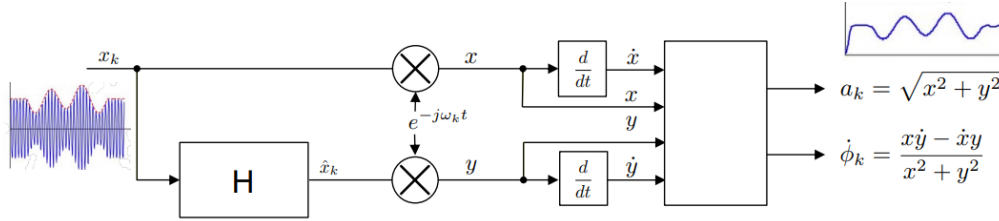


Figure 23

Notice: the phase vocoders are not able to handle more than one sinusoid per sub-band, necessitating many filters. Moreover, it's a poor model for signals with transients or sharp attacks and it's inconvenient for inharmonic signals. It's also inefficient for signals with noise-like qualities (e.g., flute) and computationally expensive.

2.7 FILTERING IN THE FREQUENCY DOMAIN

A filter is a system that performs mathematical operations on an input signal to modify, reduce or enhance certain components of features of that signal. In this section we will see different types of frequency filters.

2.7.1 Linear Time-Invariant Systems

The LTI systems are defined by two properties:

- **Linearity:** if an input x_1 produces an output y_1 and an input x_2 produces an output y_2 , then the composed input $a_1 * x_1 + a_2 * x_2$ produces the output $a_1 * y_1 + a_2 * y_2$.
- **Time Invariance:** if an input $x(t)$ produces an output $y(t)$, then $x(t - t_0)$ produces the output $y(t - t_0)$.

where a_1, a_2, t_0 are real-valued scalars.

Given an input signal $x(t)$, the output of an LTI system $y(t)$ can be obtained through **convolution**.

2.7.2 Overlap and Add Filtering

The Overlap-Add (OLA) method allows for efficient frequency domain filtering by breaking the signal into overlapping blocks, processing each block in the frequency domain, and then reconstructing the signal by overlapping and adding the processed blocks. Here's the step-by-step OLA process:

1. **Signal Framing with a Window Function:** the input signal $x(n)$ is divided into overlapping blocks $x_m(n)$ using a window function $w(n)$. Each block is obtained by multiplying $x(n)$ with $w(n - mR)$, where m is the frame index and R is the hop size, i.e., the number of samples by which the window is shifted for each successive frame.
2. **Constant Overlap-and-Add (COLA) Condition:** To ensure perfect reconstruction of the signal after processing, the window function must satisfy the COLA (constant OLA) condition $\sum_{m=-\infty}^{\infty} w(n - m * R) = 1$. This means that the sum of all windowed segments at any point in time equals one.
3. **Frequency Domain Processing:** for frequency domain processing like filtering, each windowed block $x_m(n)$ is converted to the frequency domain using the Fast Fourier Transform (FFT). However, to avoid time-domain aliasing, zero-padding is performed on each block to increase its length to $N_y = N_g + M - 1$, where N_g is the length of the filter response and M is the length of the window.
4. **The OLA Algorithm:** Each block $x_m(n)$ is then processed in the frequency domain according to the desired operation (e.g., convolution with a filter):
 - a. The product of the FFTs of the windowed block and the filter response is computed, $\hat{Y}_m(\omega k) = \hat{X}_m(\omega k) \cdot G(\omega k)$, representing the convolution in the frequency domain.
 - b. An Inverse FFT (IFFT) is then applied to convert the processed block back to the time domain, yielding $\hat{y}_m(n)$.
5. **Reconstruction of the Signal:** the processed blocks are then overlapped and added back together to reconstruct the signal. Each block $\hat{y}_m(n)$ is shifted by $-mR$ to align it with its original position, resulting in $y_m(n) = \hat{y}_m(n - mR)$. The final output $y(n)$ is obtained by summing all shifted blocks $y_m(n)$.

Perfect reconstruction is achievable if the COLA condition on the window and the anti-aliasing condition on the zero-padding are met. For causal signals with limited support, the values of m are limited to those for which the sliding window and the signal support overlap. This range is determined by the length of the signal N_x and the window length M , divided by the hop size R .

2.7.3 Wiener Filters

Consider the situation of a remote lesson at Politecnico (figure 24): we have a microphone that takes as input two signals:

- The signal $s(n)$ emitted by the professor.
- The signal $u(n)$ emitted from the speakers and filtered by the room $h(n)$.

The sum of these two signals gives us $d(n)$. Ideally, we want to transmit only the voice of the professor, so $s(n)$. To do that, we must process $d(n)$ with a filter $W(n)$ that discards $u(n) * h(n) = y(n)$ (figure 25).

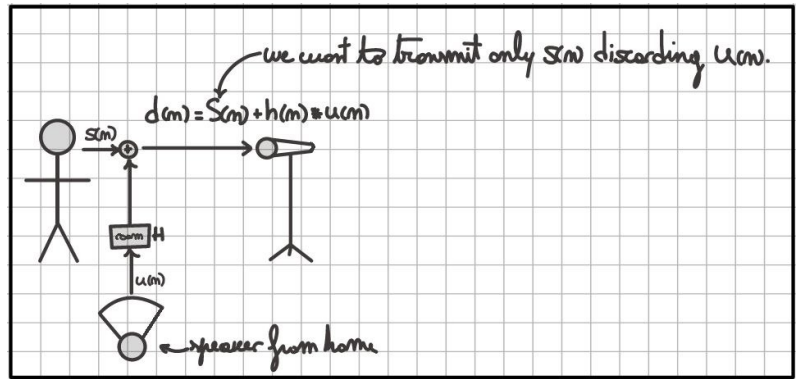


Figure 24

We assume that:

- $u(n), d(n), y(n), e(n)$:
 - are stationary (variance and mean do not change over time).
 - have mean=0.
 - $\in \mathbb{C}$.
- $W(z) \in \mathbb{C}$.

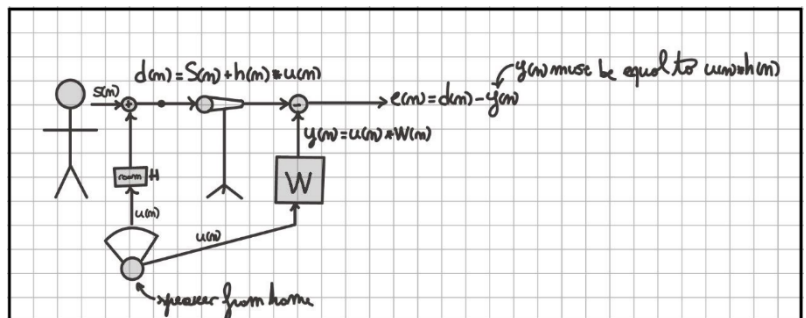


Figure 25

Done these assumptions, we can calculate $y(n)$ through linear convolution $y(n) = \sum_{k=0}^{\infty} W(k)^* * u(n - k)$.

2.7.3.1 Input – Output - Error Relations

We want to find the optimum filter W that minimizes the $MSE = E\{e(n) * e^*(n)\}$ of the result. We assume that the filter has a form of the type $W(k) = a(k) + jb(k)$.

To find the $\text{argmin}_{W(z)}(MSE)$ we can just perform the derivative of the MSE and set that derivative = 0 to find W_0 (optimal coefficients).

The MSE depends on the real and on the imaginary parts, so we must use the nabla operator $\nabla_k = \frac{\partial}{\partial a_k} + j \frac{\partial}{\partial b_k}$:

$$\begin{aligned}
 \nabla_k \cdot MSE &= \frac{\partial MSE}{\partial a_k} + j \frac{\partial MSE}{\partial b_k} = \frac{\partial E\{e(n) \cdot e^*(n)\}}{\partial a_k} + j \frac{\partial E\{e(n) \cdot e^*(n)\}}{\partial b_k} = \\
 &= E \left\{ \frac{\partial e(n)}{\partial a_k} \cdot e^*(n) + \frac{\partial e^*(n)}{\partial a_k} \cdot e(n) + j \frac{\partial e(n)}{\partial b_k} \cdot e^*(n) + j \frac{\partial e^*(n)}{\partial b_k} \cdot e(n) \right\} = 0 \quad \text{minimum condition} \\
 &\quad \text{replace the results} \quad \frac{\partial e(n)}{\partial a_k} = \frac{\partial (d(n) - \sum_{l=0}^{\infty} W^*(l) \cdot u(n-l))}{\partial a_k} = \frac{\partial d(n)}{\partial a_k} - \frac{\partial \sum_{l=0}^{\infty} (a(l) + jb(l)) \cdot u(n-l)}{\partial a_k} = -\frac{\partial \sum_{l=0}^{\infty} a(l) \cdot u(n-l)}{\partial a_k} + \frac{\partial \sum_{l=0}^{\infty} j b(l) \cdot u(n-l)}{\partial a_k} \\
 &\quad \text{no depend on } a_k \rightarrow \text{derivative} = 0 \quad \quad \quad = -u(n-k) \\
 &= E \left\{ -u(n-k) \cdot e^*(n) + j u(n+k) \cdot e(n) - u^*(n-k) \cdot e^*(n) - j u(n-k) \cdot e(n) \right\} = -2E \left\{ u(n-k) \cdot e^*(n) \right\} \rightarrow \nabla_k MSE = 0 \quad \forall k
 \end{aligned}$$

Figure 26

The fact that $\nabla_k \cdot MSE = 0 \quad \forall k$ and so $E[u(n - k) * e_{opt}^*(n)] = 0 \quad \forall k$ (orthogonality principle) means that $e_{opt} \perp u(n)$, meaning that the optimum condition is when the input and the optimal error are uncorrelated.

We can also say that $E[y_{opt}(n - k) * e_{opt}^*(n)] = 0 \quad \forall k$ (corollary of orthogonality principle):

$$E[y_{opt}(m) \cdot e_{opt}^*(m)] = 0 \rightarrow E\left[\sum_{k=0}^{\infty} W_{opt}^*(k) \cdot u(m-k) \cdot e_{opt}^*(m)\right] = \sum_{k=0}^{\infty} W_{opt}^*(k) \cdot E[u(m-k) \cdot e_{opt}^*(m)] = 0$$

$u(m-k)$ and $e_{opt}^(m)$ are orthogonal and so their product is equal to zero*

Figure 28

Graphically, the situation is the one shown in figure 27.

2.7.3.2 Normalized MSE

Considering the following definitions:

- $\hat{d} = y_{opt}$
- $\text{variance}_d = \sigma_d^2 = E[|d(n)|^2]$
- $\text{variance}_{\hat{d}} = \sigma_{\hat{d}}^2 = E[|\hat{d}(n)|^2]$
- $MSE_{min} = E[|e_{opt}(n)|^2]$

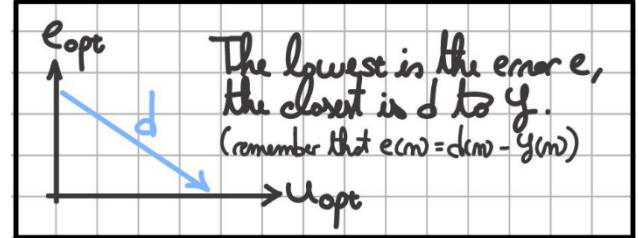


Figure 27

It is easy to verify that $e_{opt} = d - \hat{d} \Rightarrow d = e_{opt} + \hat{d}$. Applying the corollary of the orthogonality principle seen before, we can write that $\sigma_d^2 = MSE_{min} + \sigma_{\hat{d}}^2 \Rightarrow MSE_{min} = \sigma_d^2 - \sigma_{\hat{d}}^2$. If we now divide this quantity for the variance of d , we get the normalized MSE:

$$MSE_{normalized} = \frac{MSE}{\sigma_d^2} = 1 - \frac{\sigma_{\hat{d}}^2}{\sigma_d^2} \text{ with } 0 < MSE_{normalized} < 1$$

2.7.3.3 Find the coefficients of the filter

Until now, we've found a bunch of properties and relations between the input, the output and the error. Now we want to actually calculate the optimum coefficients of the filter. Let's start from the orthogonality principle:

$$E\{u(m-k) \cdot e_{opt}^*(m)\} = E\left\{u(m-k) \left[d^*(m) - \sum_{l=0}^{\infty} W_{opt}^*(l) \cdot u(m-l)\right]\right\} = 0 \rightarrow \sum_{l=0}^{\infty} W_{opt}^*(l) \cdot E\{u(m-k) \cdot u(m-l)\} = E\{u(m-k) \cdot d^*(m)\}$$

we want to cancel this *autocorrelation $r(l-k)$* *cross-correlation $P(l-k)$*

Figure 29

What we have found are the **Wiener-Hopf equation** (linear system equations, which define the optimum filter coefficients): $\sum_{l=0}^{\infty} w_{opt}(l) * r(l-k) = p(-k)$ with $k = 0, 1, 2, \dots$. The Wiener-Hopf equations define an infinite set of equations whose solution is the IIR Wiener filter. Obviously this is an ideal case, because in a real filter we can't have an infinite number of coefficients. Anyway, an approximate solution can be obtained by truncating the summation to compute a FIR filter with M coefficients:

$$\sum_{l=0}^M w_{opt}(l) * r(l-k) = p(-k) \text{ with } k = 0, 1, 2, \dots, M$$

Consider now the following definitions:

- Input vector $u(n) = [u(n), u(n-1), \dots, u(n-M+1)]^T$;
- Cross-correlation vector $p = E[u(n) * d^*(n)] = [p(0), p(-1), \dots, p(1-M)]^T$;
- Vector of optimum coefficients $w_{opt} = [w_{opt}(0), w_{opt}(1), \dots, w_{opt}(M-1)]^T$;
- Autocorrelation matrix $R = \begin{bmatrix} r(0) & r(1) & \dots & r(M-1) \\ r^*(1) & r(0) & \dots & r(M-2) \\ \vdots & \vdots & \ddots & \vdots \\ r^*(M-1) & r^*(M-2) & \dots & r(0) \end{bmatrix}$.

The Wiener-Hopf equations can be written in matrix form, and the solution is found by matrix inversion:

$$R * w_{opt} = p \Rightarrow w_{opt} = R^{-1} * p$$

The Wiener-Hopf equation can also be derived by examining the dependence of the cost function MSE on the coefficient weights of the filter:

$$\begin{aligned} MSE &= E \{ e(m) \cdot e^*(m) \} = E \left\{ \left[d(m) - \sum_{k=0}^{M-1} W^*(k) \cdot u(m-k) \right] \cdot \left[d^*(m) - \sum_{k=0}^{M-1} W(k) \cdot u^*(m-k) \right] \right\} \\ &= E \left\{ |d(m)|^2 \right\} - \sum_{k=0}^{M-1} W^*(k) \cdot E \{ u(m-k) \cdot d^*(m) \} - \sum_{k=0}^{M-1} W(k) \cdot E \{ u^*(m-k) \cdot d(m) \} + \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} W^*(k) \cdot W(l) \cdot E \{ u(m-k) \cdot u^*(m-l) \} \\ &= \sigma_d^2 - \sum_{k=0}^{M-1} W^*(k) \cdot P(-k) - \sum_{k=0}^{M-1} W(k) \cdot P^*(-k) + \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} W^*(k) \cdot W(l) \cdot r(l-k) \end{aligned}$$

Compute the gradient of the MSE:

$$\nabla_k MSE = \frac{\partial MSE}{\partial a_k} + \frac{\partial MSE}{\partial b_k} = -2P(-k) + 2 \sum_{l=0}^{M-1} W(l) \cdot r(l-k) = 0 \rightarrow \sum_{l=0}^{M-1} W(l) \cdot r(l-k) = P(-k) = \text{Wiener-Hopf equations}$$

Figure 30

2.7.3.4 Minimum MSE

We are now interested in computing the value of $MSE_{min} = \sigma_d^2 - \sigma_a^2$. The unknown value is σ_a^2 , so let's calculate it:

$$\hat{d}(m) = \sum_{k=0}^{M-1} W_{opt}^*(k) \cdot u(m-k) = W_{opt}^H \cdot u \rightarrow \sigma_a^2 = E[W_{opt}^H \cdot u \cdot u^H \cdot W_{opt}] = W_{opt}^H \cdot \underbrace{R}_{\text{cross correlation } P} \cdot W_{opt} = W_{opt}^H \cdot P = P^H \cdot W_{opt} = P^H \cdot R^{-1} \cdot P$$

Notes: W_{opt} is the solution of the equation $W_{opt} = R^{-1} \cdot P$. R is the cross correlation P .

Now we can put all together getting $MSE_{min} = \sigma_d^2 - \sigma_a^2 = \sigma_d^2 - (p^H * R^{-1} * p)$.

This result can be also write in matrix form:

$$MSE(w) = MSE_{min} + (w - w_{opt})^H * R(w - w_{opt})$$

The above equation shows explicitly the existence of a unique minimum of the cost function, occurring when $w = w_{opt}$.

The expression of the cost function can be further simplified by operating a change of basis on which it is defined. Consider:

- The eigenvalue decomposition of R: $R = Q * \Lambda * Q^H$, where $\Lambda = \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_M \end{bmatrix}$;
- The transformed version of the tap-weights $v = Q^H(w - w_{opt}) = [v_1, v_2, \dots, v_M]^T$.

Using the above definitions, the canonical form is defined as:

$$MSE = MSE_{min} + v^H * \Lambda * v = \dots = MSE_{min} + \sum_{k=1}^M \lambda_k |v_k|^2$$

3 AUDIO RESTORATION

Digital audio restoration is an essential field that combines a meticulous understanding of historical audio formats with advanced noise reduction techniques for contemporary recordings. The overarching goal is to enhance sound quality to meet today's high standards and to preserve valuable historical recordings, including old records and tapes, which are prone to various forms of degradation.

Degradation of audio can generally be categorized into two types:

- **Localized degradations:** it refers to specific instances of disruptions in the audio signal, such as clicks, crackles, or scratches, which affect only certain samples within the recording.
- **Global degradations:** it regards pervasive issues that compromise the entire recording. These can range from constant background noise to 'wow and flutter', a warbling effect due to speed fluctuations in the playback mechanism, and various types of nonlinear distortion that affect the integrity of the recorded waveform.

3.1 AUDIO SIGNAL MODELS

The Autoregressive (AR) model and the Autoregressive Moving Average (ARMA) model, which are critical tools in the analysis and processing of audio signals, particularly in the field of digital signal processing and audio restoration.

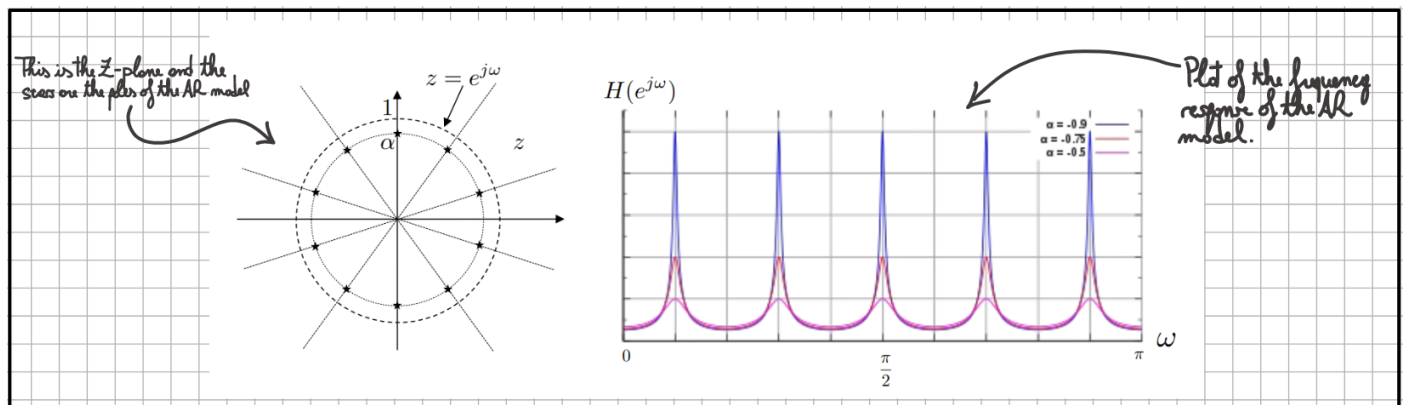
3.1.1 AR Model

The AR model, also known as the all-pole model, is expressed mathematically as

$$s(n) = \sum_{i=1}^P a_i * s(n-i) + e(n)$$

that is a summation of previous samples of the signal, each weighted by a coefficient a_i , plus an innovation process $e(n)$ which introduces new information into the signal. P is the model's order.

This model is particularly suitable for many stationary linear processes, which are processes whose statistical properties do not change over time. The AR model is adept at representing noise-like signals, which would have poles close to the origin in the z-plane, and near-harmonic signals, where poles are close to the unit circle, reflecting the periodic nature of the signal.



3.1.2 ARMA Model

On the other hand, the ARMA model combines the AR model with a moving average component, making it more flexible and capable of modelling a broader range of signal types, including non-stationary processes. The moving average component considers the average of different subsets of the total number of observations. However, this model often requires a higher order (more than 30 for simple signals) to achieve the desired level of accuracy and must be made adaptive to effectively cope with signals that change their statistical properties over time.

3.1.3 Sinusoidal Model

This model is widely used for representing speech signals. In its basic form, speech can be modelled as a sum of sinusoidal components, each representing a different frequency component of the sound.

Within the context of the sinusoidal model, a specific technique known as Short-Time Spectral Attenuation (STSA) is used for noise reduction. STSA operates on short segments of the signal (called frames) to analyse and reduce the noise while preserving the speech component.

The mathematical representation of this model is:

$$s(n) = \sum_{i=1}^{P_n} a_i(n) * \cos\left(\int_0^{nT} \omega_i(t) dt + \phi_i\right)$$

$s(n)$ is the signal at a discrete time n .
 P_n is the number of sinusoidal components at time n .
 $a_i(n)$ represents the amplitude of the i_{th} component, which can change with time.
 $\omega_{ii}(t)$ is the frequency of the i_{th} component as a function of continuous time t .
 ϕ_i is the initial phase of the i_{th} sinusoidal component.

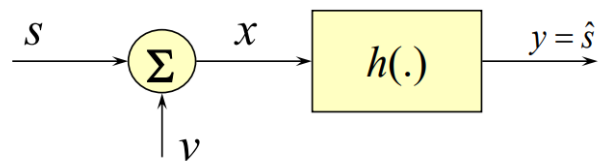
This model is capable of handling frequency and amplitude modulation, which are common in speech signals. It can also represent the 'birth' and 'death' of individual frequency components in speech, meaning it can model the starting and ending of tones or sounds within the speech.

Estimating the parameters of this model (such as the amplitudes, frequencies, and phases of the sinusoids) is difficult due to the non-stationary and complex nature of speech, where the parameters are changing rapidly over time.

3.2 WIENER FILTER

The Wiener filter is used for signal processing tasks such as noise reduction or signal estimation. It operates by applying a filter $h(\cdot)$ to a noisy signal x to produce an estimate $\hat{s}$ of the desired signal s .

As we can see in the picture on the right, the input of the filter (x) is composed of the desired signal s plus noise v , indicated by the summation symbol Σ . The signal s and the noise v are assumed to be stationary (their statistical properties do not change over time) and uncorrelated (no predictable relationship between their values at any times). They also have a mean of zero, which is typical for random processes in signal processing.



The goal of the Wiener filter is to find a filter h that minimizes the Mean Squared Error ($MSE = E[(s - y)^2]$).

The MSE is minimized when the error e , which is the difference between the desired signal s and the output y , is orthogonal to the input x . Orthogonality in this context means that the error is uncorrelated with the input signal.

Mathematically, orthogonality is achieved when $E[(s(n) - y(n)) * x(m)] = E[s(n) - y(n)] * E[x(m)]$ for all times n and m , simplifying to $E[(s(n) - y(n))x(m)] = 0$ because the signals have zero mean. This condition ensures that there is no information about the noise v in the error e , and thus the Wiener filter is doing the best job possible in estimating the desired signal s .

Note: In mathematics, two vectors are orthogonal if their dot product is zero. This concept is extended to functions or signals in signal processing. Two signals are orthogonal if their product, integrated over time, averages to zero. When we say the error $e(n)$ is orthogonal to the input $x(m)$, it implies that, on average, the error and the input are not correlated. To test this, we multiply the error at any time n with the input at any time m and find the expected value (or average) of this product:

$$E[(s(n) - y(n)) \cdot x(m)]$$

If the result is zero for all n and m , it indicates there is no consistent pattern between the error and the input. In practical terms, this means that the noise v that is part of x is not influencing the error e ; in other words, the error is purely due to the randomness of s not captured by x and not due to the noise v .

3.2.1 Wiener Filter Response

The Wiener filter's response is a sophisticated signal processing technique for extracting a desired signal s from a noisy signal x . It's widely used in applications where you want to reduce noise while maintaining the integrity of the original signal.

Let's understand how it is derived:

- **Orthogonality Principle and Initial Equations:** as we have seen, the **orthogonality principle** states that the error between the estimated signal and the actual signal should be orthogonal (not related or independent) to the input signal. Mathematically, this principle is written as:

$$E[(s(n) - y(n))x(m)] = 0$$

This equation means that if you multiply the error at any time n by the input signal at any other time m , on average (the expected value $E[\cdot]$), it should equal zero.

- **Time Domain Analysis:** from the orthogonality principle, we can derive that the cross-correlation between the signal s and the input x should equal the cross-correlation between the output y and the input x , given by:

$$E[s(n)x(m)] = E[y(n)x(m)]$$

Here, $y(n)$ is the output of the filter, which is a combination (convolution) of all past input values weighted by the filter's response h_h . This is expressed in the summation form as:

$$y(n) = \sum_k h(k)x(n - k)$$

- **Cross-Correlation and Convolution:** this summation leads to the definition of the cross-correlation function $R_{sx}(n - m)$ in terms of the filter's impulse response $h(k)$ and the autocorrelation function of the input R_x :

$$R_{sx}(n - m) = \sum_k h(k) * R_x(n - k - m)$$

- **Frequency Domain Analysis:** when we move to the frequency domain, we take the Fourier transform of the above time-domain relationships. The key relationship derived from the orthogonality principle in the frequency domain is:

$$H(\omega) = \frac{P_{sx}(\omega)}{P_x(\omega)} \text{ where } \begin{cases} H(\omega) \text{ is the frequency response.} \\ P_{sx}(\omega) \text{ is the cross spectral density between the signal and the input.} \\ P_x(\omega) \text{ is the power spectral density of the input.} \end{cases}$$

- **Wiener Filter Response:** using the information that the signal s and the noise v are uncorrelated, we can derive that:

$$P_x(\omega) = P_s(\omega) + P_v(\omega)$$

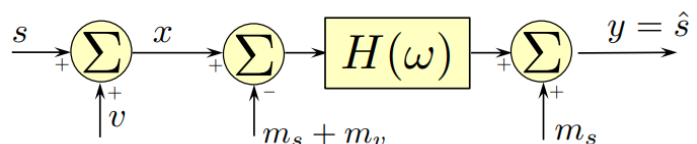
With this, we can refine the expression for $H(\omega)$ to:

$$H(\omega) = \frac{P_s(\omega)}{P_s(\omega) + P_v(\omega)}$$

Until now we've assumed that the signals have zero mean. If it's not the case and the original signal s has a non-zero mean m_s and the noise v has a non-zero mean m_v , then simply applying the Wiener filter directly on the noisy observation x (which is the sum of s and v) would not give the best estimate of the signal s .

To address this, the following steps are taken:

- **Subtract the Means:** before applying the filter, subtract the sum of the means of s and v from the observed signal x . The signal to be filtered is therefore $x - (m_s + m_v)$.

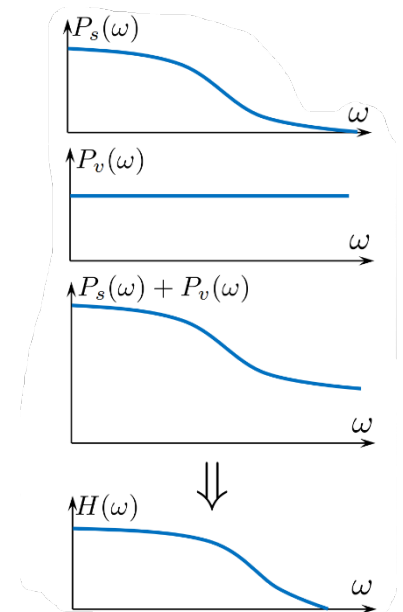


- **Apply Wiener Filter:** with the mean-adjusted signal, apply the Wiener filter $H(\omega)$ as usual. This filter operates on the frequency domain representation of the signal.
- **Add Back the Mean of s :** after filtering, add back the mean of the original signal m_s to the output y . This step is crucial because the Wiener filter only works on the fluctuating part of the signal (the part with zero mean), so we need to reintroduce the mean that was present in the original signal.

3.2.2 Filter's Bandwidth and Action

The Wiener filter is typically a lowpass filter, meaning it tends to allow low-frequency signals to pass through while attenuating high-frequency signals, and its frequency response is influenced by the signal-to-noise ratio (SNR) at different frequencies:

- **High SNR:** when the signal's power $P_s(\omega)$ at a particular frequency is much higher than the noise's power $P_v(\omega)$ at that frequency (i.e., the SNR is high), the frequency response $H(\omega)$ of the Wiener filter approaches 1. This means that the filter does little to no filtering because the signal is already strong compared to the noise at these frequencies. It's as if the filter "trusts" the signal at these frequencies, allowing it through without attenuation.
- **Low SNR:** conversely, when the signal's power at a certain frequency is much lower than the noise's power at that frequency (i.e., the SNR is low), the frequency response $H(\omega)$ of the Wiener filter approaches 0. In this case, the filter heavily attenuates the signal components at these frequencies because it "distrusts" the signal; it's more likely to be noise rather than the true signal.



3.2.3 Implementation of the Wiener Filter

The Wiener filter is usually implemented in the frequency domain using the Discrete Fourier Transform (DFT) and its inverse (IDFT). This is because convolution in the time domain is equivalent to multiplication in the frequency domain, which can be more computationally efficient.

One issue to be aware of is that the DFT assumes the signal is periodic, leading to circular convolution. To correctly implement a linear convolution (which does not assume signal periodicity), the input sequences must be zero-padded adequately to account for wrap-around effects.

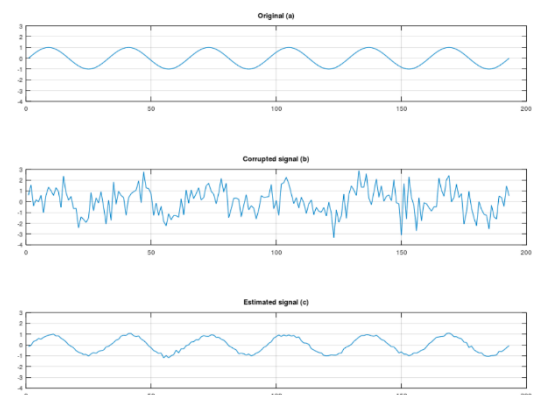
3.2.4 Wiener Filter Performances

The Wiener filter is effective in reducing noise in signals, particularly stationary noise.

It is optimal in the mean square error (MSE) sense, meaning it minimizes the average squared differences between the estimated signal and the true signal. However, the MSE is not the most relevant indicator for human sound perception quality: what is mathematically optimal may not always align with human perception.

Moreover, the filter assumes that signals are stationary, which means their statistical properties do not change over time. This assumption might not hold for non-stationary signals (where the signal properties do change), and the slide questions whether this is acceptable.

Also, the sound quality is impacted from the Wiener Filter: because the Wiener filter acts as a strong low-pass filter, it can possibly remove desirable high-frequency components along with the noise. This is again due to the assumption of stationarity.



3.2.5 Characterization of the Power Spectral Densities (PSD)

To apply noise reduction algorithms like the Wiener filter effectively, you need to know the PSD of both the noise (P_v) and the signal (P_s). The Wiener filter uses these PSDs to attenuate frequencies where the noise dominates and preserve frequencies where the signal dominates.

It's not always so easy to estimate these quantities:

- **P_s estimation:** Estimating P_s is challenging because it requires knowledge of the signal without the presence of noise, which is not always available. In some cases, it's possible to estimate P_s if there are segments of the signal that are known to be clean or have the same power spectral density as the signal. Alternatively, P_s can be extracted from portions of the recording where noise is absent if such regions exist.
- **P_v estimation:** estimating P_v is generally easier because recordings often have segments where only the noise is present (like silent parts in an audio recording).

We can use a periodogram for estimating PSD, but before that we must understand the following background concepts.

The Power Spectral Density (PSD) $P_s(\omega)$ is typically defined as the Fourier Transform of the autocorrelation function $R_s(n)$, where $R_s(n) = E[s(k)s(k+n)]$. However, this definition is not practical for computational purposes.

Wiener-Khinchin Theorem provides a more "practical" definition of PSD. It states that the PSD is equal to the expectation of the square of the Fourier Transform of the signal: $P_s(\omega) = E[|F\{s(n)\}|^2] = E[|S(\omega)|^2]$. This theorem allows us to calculate the PSD using the squared magnitude of the signal's Fourier Transform, which is much more computationally feasible.

Now we can talk about the periodogram.

A periodogram involves dividing the signal into frames $s_i(n)$ and applying the Fast Fourier Transform (FFT) to each frame. The PSD is then estimated as the average of the squared absolute values of the FFT of these frames:

$$P_s(\omega) = \frac{1}{N} * \sum_{i=1}^N |S_i(\omega)|^2 \text{ where } \begin{cases} S_i(\omega) \text{ is the Fourier Transform of the } i_{th} \text{ frame of the signal.} \\ N \text{ is the total number of frames.} \\ S_i(\omega) \text{ is the FFT of the } i_{th} \text{ frame.} \end{cases}$$

This estimation is valid if the signal is stationary (its statistical properties do not change over time) across all the N windows (or frames) used in the computation. If the signal is not stationary, one common approach is to window the signal (divide it into shorter frames or segments where the signal can be approximated as stationary). Within each window, you compute the Fourier Transform and then the squared magnitude, $|S_w(\omega)|^2$, which represents an estimate of the PSD for that specific frame. This process helps to capture the changing characteristics of the signal over time. This method does not provide a single PSD for the entire signal but rather a series of PSDs for each window, reflecting the signal's changing nature.

3.3 POWER SPECTRAL MATCHING

The Wiener filter does not lead to power spectral matching: a Wiener filter's main aim is to minimize the overall mean square error between the desired signal and the actual processed signal, but this doesn't ensure that the power spectral densities (the distribution of power into frequency components composing that signal) of the original signal and the processed signal match.

An alternate noise reduction method is the **Power Spectral Matching**, which focuses on matching the power spectrum of the processed signal (y) to that of the original signal (s). The power spectrum measures how the power of a signal is distributed with frequency. PSM can be more relevant from a perceptual standpoint as humans perceive sound in terms of its spectral properties.

The Power Spectral Matching filter equation is the following:

$$\hat{H} = \sqrt{\frac{P_s}{P_s + P_v}} \approx \frac{|X_w(\omega)|^2 - |V_w(\omega)|^2}{|X_x(\omega)|^2} \text{ where } \begin{cases} P_s \text{ is the power spectral density of the signal.} \\ P_v \text{ is the power spectral density of the noise.} \\ X_x(\omega), X_w(\omega), V_w(\omega) \text{ represent the PSD of different signal components.} \end{cases}$$

This equation balances the signal and noise power to adjust the filter, aiming to produce an output with a power spectrum similar to that of the original signal (s).

With the given $\hat{H}$, the power spectrum of the output y is:

$$P_y = P_x |\hat{H}|^2 = P_x * \frac{P_s}{P_s + P_v} = P_s$$

This ensures that the output power spectrum P_y matches the desired signal power spectrum P_s .

In simpler terms, Power Spectral Matching is suggested as a method to make the output signal sound more like the original by ensuring they share the same power distribution over frequencies, which might be more aligned with how we hear and perceive sounds.

3.4 PARAMETRIC WIENER FILTERING

We have seen that Wiener filter can be represented by the following formula:

$$H(\omega) = \frac{P_x(\omega) - P_v(\omega)}{P_x(\omega)} = 1 - \frac{P_v(\omega)}{P_x(\omega)}$$

This filter aims to produce an output that minimizes the mean square error between the desired signal and the actual output.

The Parametric Wiener Filter $H_p(\omega)$, which is a variation of the Wiener filter, allows for "morphing" between the traditional Wiener filter and Power Spectral Matching (PSM), which was discussed in the previous paragraph. It gives the user control over the filter's behavior by introducing a parameter β that can be adjusted.

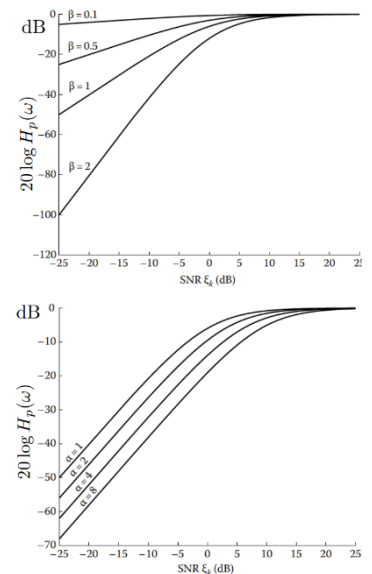
The formula provided for the Parametric Wiener Filter is:

$$H_p(\omega) = \left(\frac{P_x(\omega) - \alpha P_v(\omega)}{P_x(\omega)} \right)^\beta \text{ where } \begin{cases} \alpha \text{ parameter can be thought of as a scaling factor for the noise PSD.} \\ \beta \text{ modifies the overall shape of the filter's response.} \end{cases}$$

An alternate form of the Parametric Wiener Filter is:

$$H_p(\omega) = \left(1 - \alpha * \frac{P_v(\omega)}{P_x(\omega)} \right)^\beta$$

On the right side, the two graphs show the magnitude response of the Parametric Wiener Filter in decibels (dB) for different values of β and α . The x-axis represents the Signal-to-Noise Ratio (SNR) in dB, while the y-axis shows the magnitude response of the filter also in dB.



3.5 BACKGROUND NOISE REDUCTION

As we have seen noise is assumed to be additive, meaning the noise v adds to the signal s to create the observed signal x . The noise and signal are assumed to be uncorrelated, which implies they do not have any linear dependency on each other.

Background noise, which may manifest as a hiss, is described as a single noise process with several potential sources:

- **Electrical circuit noise:** Unwanted disturbances generated by electronic circuits.
- **Irregularities of storage media:** Imperfections or degradation in the physical media where audio is stored.
- **Ambient noise from recording environment:** Sounds from the environment where the recording took place that are not part of the intended signal.

This type of noise typically has components at all frequencies and is often considered a white random process. White noise has equal intensity at different frequencies, giving it a constant power spectral density.

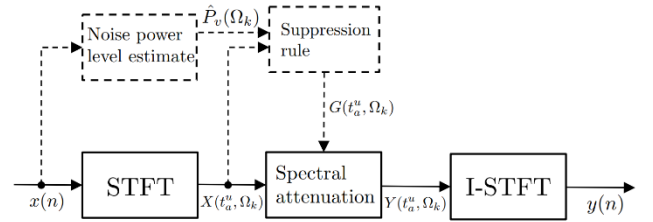
There are two main techniques for Noise Reduction:

- **Stationary Formal Solution:** The Wiener filter, a statistical approach to signal processing used to produce an estimate of a desired or target signal by suppressing noise. It is optimal in minimizing the overall mean square error between the estimated and true signal.
- **Adaptive Version:** Short-Time Spectral Attenuation (STSA), which is an adaptive method for noise reduction. STSA operates on short segments of the signal (frames), allowing it to adapt to changes in both the signal and noise characteristics over time. This is particularly useful for non-stationary noise, which changes its properties over time.

3.5.1 Short-Time Spectral Attenuation (STSA)

Wiener filtering it's often inadequate for real-world applications where noise characteristics can change over time. Therefore, an adaptive counterpart is needed. Short-Time Spectral Attenuation (STSA) is an adaptive method that deals with non-stationary or time-varying noise. STSA works as follows:

- **Analysis:** The noisy signal $x(n)$ is analyzed using the Short-Time Fourier Transform (STFT) to convert the time domain signal into the time-frequency domain. This helps to examine the signal's frequency content at successive time intervals.
- **Attenuation:** For each time-frequency bin (represented by t_u for time and Ω_k for frequency), STSA applies an attenuation rule to suppress noise.
- **Recombination:** The processed time-frequency bins are recombined using the inverse STFT (I-STFT) to reconstruct the time domain signal $y(n)$, which is expected to have reduced noise.



In this process, the spectral attenuation function $G(t_u, \Omega_k)$ determines how much attenuation to apply to each time-frequency bin based on:

1. The power level of the noisy signal at the same bin $|X(t_u, \Omega_k)|^2$.
2. The estimated power of the noise $\hat{P}_v(\Omega_k)$ at frequency k . This noise power is assumed to be stationary, meaning it does not change over time. The expected value $E[|V(t_u, \Omega_k)|^2]$ represents the noise power level estimate.

3.5.2 Noise Suppression Rules

These rules are applied to the signal after it has been transformed by the Short-Time Fourier Transform (STFT), and they define how much the signal should be attenuated in each frequency band to reduce the noise. There are three suppression rules:

1. **Wiener Suppression Rule:** This rule is the basic form used in Wiener filtering. The attenuation depends on the ratio of the noise power spectral density $\hat{P}_v(\Omega_k)$ to the power of the noisy signal at a specific frequency bin $|X(t_u, \Omega_k)|^2$. The rule is designed to reduce the noise while trying to preserve the original signal.

$$G(t_u^a, \Omega_k) = 1 - \frac{\hat{P}_v(\Omega_k)}{|X(t_u^a, \Omega_k)|^2}$$

2. **Power Spectral Matching Suppression Rule:** This rule is used for Power Spectral Matching (PSM), which, unlike the Wiener filter, aims to match the power spectrum of the output to that of the original signal more closely.

$$G(t_u^a, \Omega_k) = \sqrt{1 - \frac{\hat{P}_v(\Omega_k)}{|X(t_u^a, \Omega_k)|^2}}$$

3. **Parametric Wiener Suppression Rule:** This is a more flexible rule that introduces a parameter β to allow a continuous transition between the traditional Wiener filtering and PSM. The parameter α is used to scale the

noise power spectral density. By adjusting β , one can control the level of noise suppression and how aggressively the filter behaves.

$$G(t_u^u, \Omega_k) = \left(1 - \alpha * \frac{\hat{P}_v(\Omega_k)}{|X(t_u^u, \Omega_k)|^2} \right)^\beta$$

For all the above rules, two extreme conditions are considered:

- As the Signal-to-Noise Ratio (SNR) approaches infinity, the suppression function G approaches 1, meaning no attenuation is applied, as the signal is already much stronger than the noise.
- As the SNR approaches 0 (meaning noise dominates), the suppression function G approaches 0, indicating maximum attenuation.

3.5.3 Dereverberation

Dereverberation, or reverberation removal, is the process of mitigating the effects of reverberation from an audio signal. Reverberation is the persistence of sound after the sound source has been stopped due to multiple reflections off surfaces in an environment like walls, ceilings, and objects. This can make sound recordings unclear and can degrade the performance of systems that rely on audio input, like speech recognition systems.

The audio signal $x(n)$ is assumed to be a convolved signal of the clean speech signal $s(n)$ and the room impulse response (RIR) $h(n)$. The RIR characterizes how sound propagates in a room, including direct paths and multiple reflections (echoes). The general approach to dereverberation involves filtering the recorded signal $x(n)$ with an inverse filter of $h(n)$, aiming to recover the clean speech signal.

The dereverberation process involves several steps to remove the effects of reverberation from the recorded signal:

3.5.3.1 Cepstrum Based Reverberation

We first determine $h(n)$ by measuring it directly or estimating it from the reverberant speech signal $x(n)$. Cepstrum-based dereverberation is one method to estimate $h(n)$.

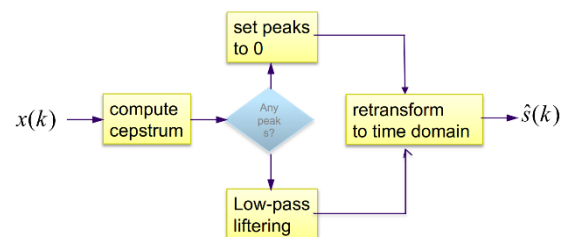
The Cepstrum method involves transforming the signal $x(n)$ into the cepstral domain. The complex cepstrum $c_x(n)$ is computed using the Discrete Fourier Transform (DFT) of the signal $x(n)$, followed by the logarithm of the magnitude of the DFT (to separate the magnitude and phase), and then the Inverse Discrete Fourier Transform (IDFT).

The complex Cepstrum separates the multiplication of the frequency components in the spectral domain into an addition in the cepstral domain. This separation makes it possible to distinguish the contributions of the RIR $h(n)$ and the speech signal $s(n)$.

In the Cepstrum domain:

- Speech is generally represented by low-frequency components and is concentrated around the origin in the cepstral domain.
- Reverberation channels correspond to high-frequency components and manifest as pulses far from the origin.

One can remove reverberation by identifying and eliminating the high-frequency components that correspond to the reverberation channel. The process involves setting the peaks representing reverberation in the cepstral domain to zero, applying a low-pass filter (liftering), and transforming the result back to the time domain to get the dereverberated signal $\hat{s}_k$.



3.5.3.2 Inverse Filter Estimation

Determine the "inverse" of $h(n)$. This can be challenging since the inverse of a finite impulse response (FIR) filter is an infinite impulse response (IIR) filter, which may not be stable.

Given $h[k]$ (assuming an FIR filter), an inverse filter $g[k]$ is estimated that satisfies the convolution property $h[k] \circ g[k] = \delta[k]$, where $\delta[k]$ is the Kronecker delta function.

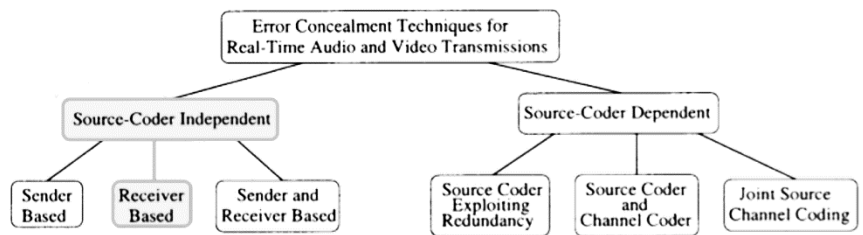
The inverse of an FIR filter is an IIR filter, which may not be stable. To overcome this, the FIR inverse filter is computed as an approximate solution using a least squares method: a least squares solution of an overdetermined set of linear equations is computed to estimate the inverse filter. The method involves constructing a Toeplitz matrix from the RIR and solving for g using the pseudoinverse.

$$\underbrace{\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathbf{d}} = \underbrace{\begin{bmatrix} h[0] & 0 & \dots & 0 \\ h[1] & h[0] & \ddots & \vdots \\ \vdots & h[1] & \ddots & 0 \\ h[N_h] & \vdots & \ddots & h[0] \\ 0 & h[N_h] & \vdots & h[1] \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & h[N_h] \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} g[0] \\ g[1] \\ \vdots \\ g[N_g] \end{bmatrix}}_{\mathbf{g}} \Rightarrow \mathbf{g}_{LS} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{d}$$

Notice that if the RIR is non-minimum-phase, meaning it has unstable zeros, a direct inversion is not possible because the residual error from the least squares solution will not converge to zero as the length of g increases.

3.5.4 Error Concealment

When audio data is streamed over networks such as the internet, packet loss can occur due to unreliable transmission. This loss can be because of packet delay, packets arriving out of sequence, or packets not arriving at all (best-effort delivery service), which does not guarantee Quality of Service (QoS).



Traditional audio compression algorithms assume an error-free channel and aim to maximize coding gain, meaning they are not robust against transmission errors.

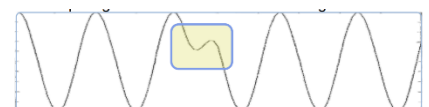
The Solutions for Packet Loss could be robust error-concealment and coding strategies. These techniques can be broadly classified into two categories depending on how they interact with source coders:

- **Source-Coder Independent:** These methods don't rely on the specifics of the audio or video coding techniques. They can be sender-based or receiver-based.
- **Source-Coder Dependent:** These techniques are tightly coupled with the codec being used. They include exploiting source coder redundancy and joint source/channel coding.

3.5.4.1 Receiver-Based Schemes

When no redundancy is added at the encoder side, the receiver has to handle packet loss, often referred to as Packet Loss Concealment (PLC). Advanced PLC approaches are based on signal analysis where the speech signal is extrapolated to fill in the gaps.

A solution to error at receiver side is the **Nonparametric Concealment**: it includes techniques such as Overlap-and-Add (OLA), where the missing frame is reconstructed by time-stretching adjacent frames. Time-scale compression or expansion can be used by repeating or discarding segments and then copying them back into the output signal. In this type of solution, we could incur in the problem of artifacts due to the OLA.

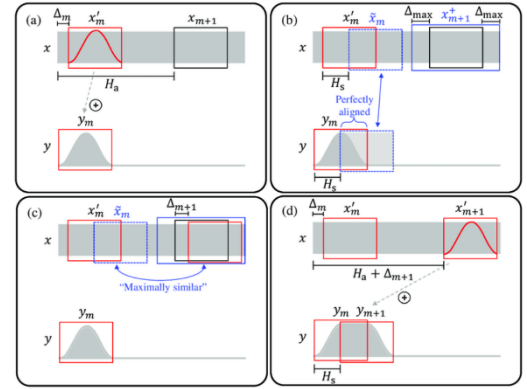


To avoid splicing artifacts, several approaches are possible:

- **Synchronized OLA (SOLA):** Splices are repositioned to correlate better with the existing signal portion.

- **Pitch-Synchronous OLA (PSOLA):** Maintains the speech naturalness by avoiding changes in formant locations or bandwidth.
- **Waveform Similarity OLA (WSOLA):** Looks for segments within the signal that best match the missing part and uses them for reconstruction. This is done through the following steps:

1. **Input Signal:** It starts with an input audio signal x and a windowed analysis frame x_m .
2. **Frame Retrieval:** A frame is retrieved from the extended frame region x_{m+1}^+ that is as similar as possible to the adjusted analysis frame x_m .
3. **Frame Adjustment and Output:** The selected frame is then aligned perfectly with the analysis frame x_m to ensure smooth continuity, and then the adjusted frame is added to the output signal y .

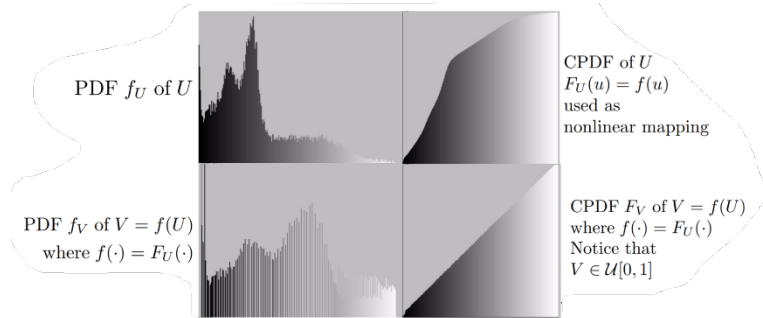


These methods aim to maintain the continuity and naturalness of the signal while filling in the missing parts.

3.6 NONLINEARITY EFFECT ON SIGNAL STATISTICS

Let's assume the audio signal $U(n)$ as a locally stationary random process. This means that while its statistical properties may change over time, over short periods, they can be assumed constant.

Consider the Cumulative Probability Density Function $F_U(u)$ that represents the probability that the signal amplitude is less than or equal to a value u , estimated from the histogram of amplitudes. Applying a generic nonlinearity $g(\cdot)$ to the signal U modifies its CPDF. If the nonlinearity $g(\cdot)$ is chosen such that $g(U) = F_U(U)$, the output signal V will have a uniform distribution across its range.



This can be mathematically expressed as:

$$P[V \leq v] = P[F_U(U) \leq v] = P[U \leq F_U^{-1}(v)] = F_U(F_U^{-1}(v)) = v$$

that means that the CPDF of V , $F_V(v)$, equals v , confirming the uniform distribution.

3.6.1 PDF Shaping

PDF shaping is a technique used to manipulate the probability distribution function (PDF) of a signal. It's often applied to signals to achieve a specific statistical characteristic or to apply a form of non-linear normalization, like histogram equalization. In the slides, PDF shaping is used to transform the distribution of an audio signal to a desired form, particularly for dynamic range compression and noise reduction purposes.

Here's a step-by-step explanation of how PDF shaping is performed:

1. **Equalization:** the process begins with transforming the signal into a uniformly distributed one. This is done by mapping the original signal U through its own cumulative distribution function F_U , resulting in an intermediate signal W , where $W = F_U(U)$. Essentially, this step "equalizes" the histogram of the signal's amplitudes, making the probability of all amplitudes equally likely.
2. **De-equalization (Shaping):** next, to shape the PDF of the equalized signal W , you apply the inverse cumulative distribution function of the desired PDF, F_V^{-1} . The final output signal Y is obtained by mapping W using this inverse function, such that $Y = F_V^{-1}(W) = F_V^{-1}[F_U(U)]$.
3. **Final Nonlinear Function:** the nonlinear function $f(\cdot)$ that's applied to U to obtain Y is the composition of F_V^{-1} (the shaping function) and F_U (the equalizing function).

This composite function transforms the original signal into one with the desired PDF.

3.6.2 DBX

DBX refers to a family of noise reduction systems that were developed by DBX, Inc. The name "dbx" is derived from "decibel expansion," which gives a hint to its operational principle. The technology was originally designed to reduce noise and increase the dynamic range of analog audio signals.

DBX is a type of companding (compressing/expanding) noise reduction system. Companding is a process that compresses the dynamic range of an audio signal in one stage of the processing and then expands it back to the original dynamic range in a later stage.

Here's a simplified explanation of how DBX noise reduction works:

1. **Compression:** In the recording phase or during transmission, the dbx system compresses the dynamic range of the audio signal. It does this by reducing the volume of loud sounds and increasing the volume of quiet sounds. This compression is typically non-linear and varies with the input signal level.
2. **Noise Reduction:** Because the quiet parts of the audio (which often include noise and hiss) are boosted during recording, any noise inherent in the recording medium or transmission path is also boosted. By compressing the dynamic range, the relative level of this noise is reduced.
3. **Expansion:** Upon playback, the dbx system expands the dynamic range of the audio signal. The expander effectively reverses the compression applied during recording, bringing loud sounds back up and quiet sounds back down to their original levels.

The net effect of this process is that the noise introduced during recording or transmission, which had been reduced relative to the desired audio signal, remains quiet during playback. The expansion process does not amplify it back to its original level, thus achieving a lower perceived noise floor.

The DBX noise reduction system was widely used in professional audio recording and broadcast during the analog era and in consumer cassette decks. The process relies on accurate matching of the compression and expansion processes; if these are not perfectly matched (due to calibration errors, for instance), unwanted artifacts can be introduced to the audio. Today this system of noise reduction is less common since digital systems have a naturally wider dynamic range and lower noise floor than analog systems. However, the principles behind DBX are still applied in various forms in digital signal processing to manage dynamic range and reduce perceived noise.

3.6.3 Dolby Noise Reduction (NR)

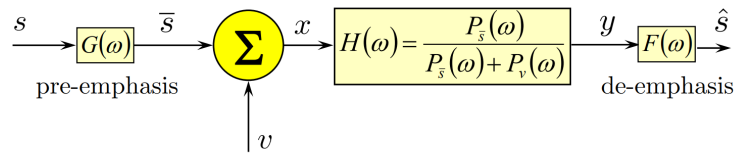
Dolby NR is a system that utilizes dynamic pre-emphasis during the recording phase, followed by dynamic deemphasis during playback. Pre-emphasis boosts certain frequencies of the signal, particularly the higher frequencies that are more susceptible to noise. During playback, deemphasis inversely attenuates these frequencies, which also reduces the noise introduced in the recording or transmission process.

The complete processing chain for treating additive noise with Dolby NR is analogous to the operation of a Wiener filter. The Wiener filter, $H(\omega)$, is viewed as the "cure" for the noise, formulated as:

$$H(\omega) = \frac{P_s(\omega)}{P_s(\omega) + P_v(\omega)} \text{ where } \begin{cases} P_s(\omega) \text{ is the power spectral density of signal.} \\ P_v(\omega) \text{ is the power spectral density of the noise.} \end{cases}$$

The Dolby system can be thought of as a combination of pre-emphasis and deemphasis filters designed to make the Wiener filter's effect transparent (irrelevant). If the pre-emphasis, represented as $G(\omega)$, is chosen to be the square root of the noise-to-signal power ratio, and deemphasis $F(\omega)$ is the inverse of $G(\omega)$, then the action of the Wiener filter becomes unnecessary. This is because the combination of pre-emphasis and deemphasis reshapes the signal and noise in such a way that the Wiener filter would essentially apply a constant gain (in this case, multiplication by 1/2).

- Consider the following scheme



- if we choose $G = \sqrt{\frac{P_v}{P_s}}$ and $F = \frac{\sqrt{2}}{G} = \sqrt{\frac{2P_s}{P_v}}$
- we have

$$P_{\bar{s}}(\omega) = |G(\omega)|^2 P_s(\omega) = \frac{P_v(\omega)}{P_s(\omega)} P_s(\omega) = P_v(\omega) \Rightarrow H(\omega) = \frac{P_{\bar{s}}(\omega)}{P_{\bar{s}}(\omega) + P_v(\omega)} = \frac{1}{2}$$

$$P_y(\omega) = |H(\omega)|^2 P_x(\omega) = \frac{1}{4} (2P_v(\omega)) = \frac{1}{2} P_v(\omega)$$

$$P_{\hat{s}}(\omega) = |F(\omega)|^2 P_y(\omega) = \frac{2P_s(\omega)}{P_v(\omega)} \frac{1}{2} P_v(\omega) = P_s(\omega)$$

$H(\omega)$ is now irrelevant

The power spectral of the result is now matching that of the target signal

3.7 RESTORATION OF LOCAL DEGRADATIONS

Historical recordings frequently suffer from finite-duration defects occurring at random locations, impacting the signal in a localized manner. These degradations are experienced in various forms, such as:

- Tiny tick noises, which can occur in any recording medium, including modern digital sources.
- Scratch and crackle noise associated with most analog disc recordings.

A poor-quality 78 rpm record might typically exhibit around 2000 clicks per second with durations ranging from less than 20 microseconds up to 4 milliseconds in extreme cases. Despite these degradations, in most cases, at least 90% of the samples remain undegraded, making it reasonable to hope for successful audio restoration.

The typical causes of these degradations include:

- Specks of dirt or dust adhering to the grooves of a disc.
- Granularity in the material used for pressing the disc.
- Physical damage such as surface scratches.

There are two models for handling defects in audio recordings:

- Additive model:** the additive model is used when defects such as small scratches or dust are superimposed on the original signal without completely masking it.
- Replacement model:** the replacement model is used for severe defects that completely disrupt the original signal, like large scratches or breaks.

Defects are mathematically modelled as

$$x(n) = s(n) + i(n) * v(n) \text{ where } \begin{cases} s(n) \text{ is the original, undisturbed audio signal.} \\ v(n) \text{ represents the corrupting noise process.} \\ i(n) \text{ is } = 1 \text{ when a degradation is present and } = 0 \text{ otherwise.} \end{cases}$$

In this context, $i(n)$ acts as a switch that adds noise $v(n)$ to the signal $s(n)$ only at points where there is a click or pop due to degradation.

A simple attempt to clicks detection could be Highpass Filtering (HPF):

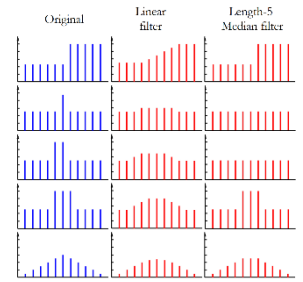
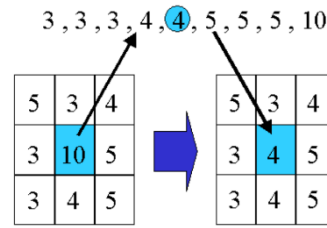
- The process starts with highpass filtering the signal, which magnifies the clicks by removing lower frequency components that are typically not part of the click but part of the underlying audio signal.
- After filtering, thresholding is applied, which aims to separate the clicks from the signal based on amplitude. Any signal above the threshold is considered a click.

This method is straightforward and doesn't require knowledge of the system parameters, but it has significant limitations. It assumes that there's a frequency separation between the signal and the clicks, which is rare. Problems arise when the signal itself has high-frequency content (high-pass) or when the clicks have a limited frequency band (band-limited), which could be variable.

Simple attempts for reconstructing the audio signal after clicks have been detected could be:

1. **Median Filter:** This filter is used to replace the corrupted samples with the median value of the surrounding samples. It works well for modest degradation but is not suitable when there are large gaps.

The median filter operates by moving through each entry (or pixel, in the case of images) in the signal (or image) and replacing it with the median of neighbouring entries. This is typically done using a sliding window of odd length, which makes it easier to define the median (as there will be a middle value with an equal number of values on either side). Fine details in the signal can be lost due to the filtering process.



2. **Splicing:** If the gap is too large for a median filter to be effective, the method suggests splicing uncorrupted data from nearby parts of the signal into the gap. When splicing, you must deal with potential discontinuities in the signal, which can be addressed using time/pitch scaling methods. This requires a periodic signal and an accurate estimation of the pitch period to ensure that the spliced segments blend smoothly with the original signal.

3.7.1 Restoration Process Steps

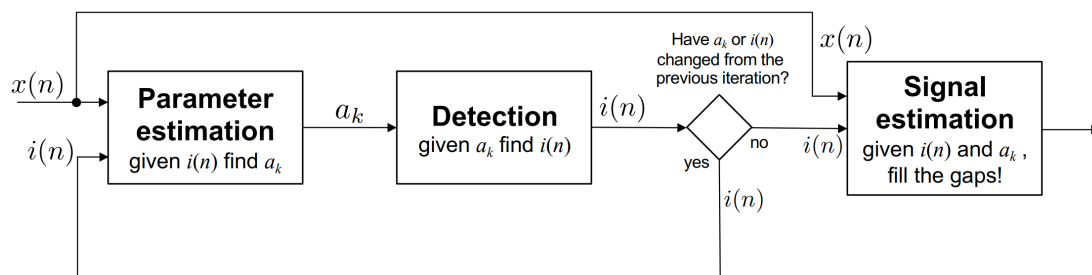
Simple or 'blind' methods for detection and estimation are not very effective. Such methods don't use any model of the signal or the noise and thus have limited ability to accurately detect and estimate the signal.

On the other hand, model-based methods, which use a predefined model like sinusoidal, Auto-Regressive (AR), or periodic models, are necessary. These models help in understanding the structure of the signal and the nature of the noise, which can then be used to detect and repair the signal more effectively.

Ideally, detecting the corrupted segments of the signal and estimating the clean signal would be done together for the best results. However, this is a difficult problem because detection and estimation influence each other. To address this, we can use an iterative approach, where detection and estimation are refined through multiple iterations to improve accuracy.

The process steps are the following:

- **Parameter Estimation:** given the detected corrupted samples $i(n)$, find the parameters a_k that best describe the signal according to the chosen model.
- **Click Detection:** With the estimated parameters a_k , detect which samples are corrupted, updating $i(n)$.
- **Signal Estimation:** Then, with the updated information about corrupted samples $i(n)$ and the parameters a_k , estimate the clean signal by filling the gaps caused by the clicks.
- **Convergence Check:** After a detection and estimation cycle, the process checks if the parameters a_k or the detection sequence $i(n)$ have changed from the last iteration. If there's a change, it repeats the cycle; if not, it assumes convergence has been reached.



Let's see how we can implement these steps.

3.7.2 Clicks Detection

Consider the audio signal $s(n)$ as being generated by a short-term stationary AR process. This means that each sample $s(n)$ can be predicted from the previous P samples using a set of parameters a_k , plus some error term $e(n)$. Let's assume also that we know the AR parameters a_k and the variance of the excitation σ_e^2 are known (*we will see how to get these parameters in the next steps*). These parameters are used to predict current samples from past samples.

The filter $H(z)$, also known as the error prediction filter, is defined as

$$H(z) = 1 - \sum_{k=1}^P a_k * z^{-k}$$

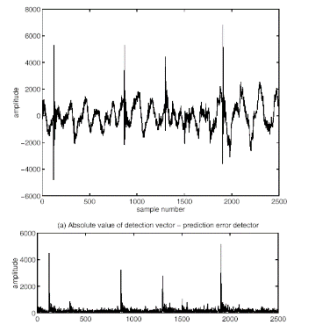
This filter is used to predict the current sample from the past samples and calculate the difference between the predicted and actual sample, which gives the prediction error $e(n)$.

When the corrupted signal $x(n)$, which is $s(n) + i(n) * v(n)$ (where $i(n)$ is an indicator function that is 1 when there is degradation and 0 otherwise), is fed into the prediction error filter, the output $e_d(n)$ reflects the combined effect of the prediction error and the clicks.

The key to click detection is to examine $e_d(n)$, which highlights discrepancies from the expected signal that are caused by clicks.

The variance of the prediction error can be derived from the uncorrupted signal and used to establish a threshold for detecting clicks (*the variance of this prediction error is a measure of how much the signal deviates from the prediction*). The variance σ_e^2 is calculated based on the power spectrum of the uncorrupted signal and the frequency response of the prediction error filter:

$$\sigma_e^2 = \frac{2\pi * \sigma_s^2}{\int_{-\pi}^{\pi} \left| \frac{1}{H(e^{j\omega})} \right|^2 d\omega} \quad \text{where} \quad \begin{cases} 2\sigma_e^2 \text{ is the variance of the prediction error} \\ \sigma_s^2 \text{ is the power of the uncorrupted signal} \\ |H(e^{j\omega})|^2 = \text{squared magnitude of the prediction error filter freq response} \end{cases}$$



This process can be implemented with a simple algorithm. For each sample index n from 1 to N (the length of the signal):

1. Compute the prediction error $e(n)$.
2. If the absolute value of $e(n)$ exceeds a threshold $k * \sigma_e$, then a click is detected at that sample $i(n) = 1$, otherwise no click is detected $i(n) = 0$.

It is very important to choose the correct threshold, based on the variance of the prediction error and the order of the AR model. There is a trade-off between false positives (detecting a click when there isn't one) and false negatives (not detecting a click when there is one).

3.7.3 Parameter Estimation (Maximum Likelihood (ML) estimation method)

3.7.3.1 AR modelling recap

The AR model assumes that the current value of the signal can be linearly predicted from its previous values with some added Gaussian noise, known as the prediction error or innovation. This error is assumed to be a white Gaussian noise process, which means it's uncorrelated and has a constant spectral density (it is identically distributed across time). Mathematically:

$$x(n) = \sum_{k=1}^P a_k * x(n - k) + e(n)$$

So, the observed data x is modeled as a linear combination of the past signal values weighted by the AR parameters a_k , plus the prediction error e .

This can be expressed in a matrix form:

$$\hat{x} = G_x * a + e \text{ where } \left\{ \begin{array}{l} \hat{x} = \begin{bmatrix} x(P+1) \\ \dots \\ x(N) \end{bmatrix} = \text{past } P \text{ samples} \\ G_x = \begin{bmatrix} x(P) & x(P-1) & \dots & x(1) \\ x(P+1) & X(P) & \dots & x(2) \\ \vdots & \vdots & \ddots & \vdots \\ x(N-1) & X(N-2) & \dots & x(N-P) \end{bmatrix} = \text{lagged values of the signal.} \\ a = \begin{bmatrix} a_1 \\ \dots \\ a_P \end{bmatrix} \\ e = \begin{bmatrix} e(P+1) \\ \dots \\ e(N) \end{bmatrix} = \text{vector of prediction errors.} \end{array} \right.$$

3.7.3.2 Parameter Estimation (Maximum Likelihood (ML) estimation method)

We want to find the parameters of an Auto-Regressive (AR) model through Maximum Likelihood (ML) estimation method. The goal of the ML estimator is to find the set of parameters a that maximize the likelihood (*verosimiglianza*) of observing the given data x , assuming that the model structure and the statistical properties of the prediction error are correct.

3.7.3.3 Likelihood Function

The situation is the following: the observed data x is known, and from it, you can often derive the Probability Density Function (PDF) for x when the AR parameters a are known. The likelihood function $L(x; a)$ is the probability of observing the data x given the parameters a .

$L(x; a)$ can be expressed in terms of the probability density of the prediction errors:

$$p_e(\hat{x} - G_x * a) \text{ where } \begin{cases} G_x \text{ is the matrix constructed from lagged values of } i. \\ a \text{ is the vector of AR parameters.} \end{cases}$$

To simplify calculations, it is common to maximize the natural logarithm of the likelihood function instead of maximizing $L(x; a)$ directly (products become sums, which are easier to differentiate).

3.7.3.4 Deriving the ML Estimate

To find the ML estimate, you differentiate the log-likelihood with respect to the parameters a and set the derivative equal to zero.

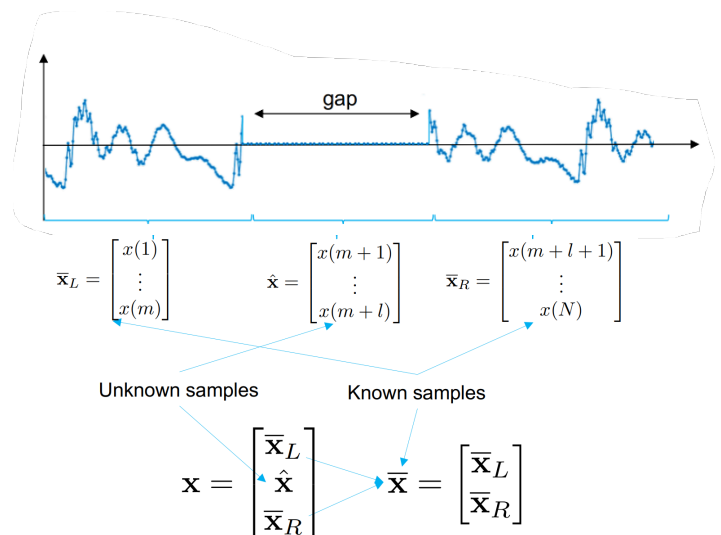
The solution to that set of linear equations (that maximize the likelihood) is given by:

$$a^{ML} = (G_x^T * G_x)^{-1} * G_x^T * \hat{x}$$

3.7.4 Signal Estimation

In the signal estimation step, we want to “fill the gap” in our damaged signal. To reach that goal, we can use the LSAR algorithm. LSAR is commonly used in digital click removal, which is a process of reconstructing audio signals where parts of the data are missing or corrupted.

The method begins with a block of audio samples suspected to contain gaps. These samples are derived from an AR process characterized by its AR coefficients. The block of samples is divided into three parts: known samples before the gap, the missing samples (the gap), and known samples after the gap.



We can mathematically represent the data partitioning in the following way:

$$\mathbf{x} = \begin{bmatrix} \bar{\mathbf{x}}_L \\ \hat{\mathbf{x}} \\ \bar{\mathbf{x}}_R \end{bmatrix}_{[N \times 1]} = \underbrace{\begin{bmatrix} \mathbf{0}_{m \times l} \\ \mathbf{I}_{l \times l} \\ \mathbf{0}_{(N-m-l) \times l} \end{bmatrix}}_{\mathbf{U}_{[N \times l]}} \underbrace{\hat{\mathbf{x}}}_{[l \times 1]} + \underbrace{\begin{bmatrix} \mathbf{I}_{m \times m} & \mathbf{0}_{m \times (N-m-l)} \\ \mathbf{0}_{l \times m} & \mathbf{0}_{l \times (N-m-l)} \\ \mathbf{0}_{(N-m-l) \times m} & \mathbf{I}_{(N-m-l) \times (N-m-l)} \end{bmatrix}}_{\mathbf{K}_{[N \times (N-l)]}} \underbrace{\begin{bmatrix} \bar{\mathbf{x}}_L \\ \bar{\mathbf{x}}_R \end{bmatrix}}_{\bar{\mathbf{x}}_{[(N-l) \times 1]}}$$

where $\mathbf{0}_{n \times m}$ is a $[n \times m]$ zero matrix and $\mathbf{I}_{n \times m}$ is a $[n \times m]$ unit matrix.

$$\mathbf{x} = \mathbf{U} * \hat{\mathbf{x}} + \mathbf{K} * \bar{\mathbf{x}}$$

Matrices $\mathbf{U}$ and $\mathbf{K}$ are used to separate the known components from the unknown components of the signal. This rearrangement facilitates the application of linear algebra techniques to solve for the missing samples.

3.7.4.1 AR alternate form

Notice that, given the AR process $\hat{x} = G_x * a + e \rightarrow e = \hat{x} - G_x * a$, we can express the prediction error e as:

$$\mathbf{e} = \mathbf{A}\mathbf{x} \quad \Leftarrow \text{Linear in } \mathbf{x}$$

$$\begin{bmatrix} e(P+1) \\ \vdots \\ e(N) \end{bmatrix} = \begin{bmatrix} -a_P & -a_{P-1} & \cdots & -a_1 & 1 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & -a_P & \cdots & -a_2 & -a_1 & 1 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & -a_P & \cdots & -a_1 & 1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(N) \end{bmatrix}$$

The matrix $\mathbf{A}$ is structured such that each row corresponds to the coefficients of the AR model that multiply the past signal values to estimate the prediction error for a particular time point.

3.7.4.2 LSAR

LSAR (Least Squares Auto-Regressive) is a method for estimating unknown data samples based on known ones. It aims to estimate unknown samples x by finding the best approximation that minimizes the difference between the actual data and the estimated data, known as the prediction error.

Cost Function: the cost function chosen for LSAR is the energy of the prediction error across the entire data block, formulated as $E = e^T * e$, where e is the error vector. This error vector is calculated as the difference between the actual data and the data predicted by the AR model. As we have seen in the previous paragraph, we can express the prediction error also as $e = \mathbf{A} * \mathbf{x}$, where $\mathbf{A}$ is the matrix related to the AR process and $\mathbf{x}$ are the data samples.

LSAR involves the steps listed below:

1. Estimation process:

- The data block $\mathbf{x}$ is assumed to be generated by an AR process with certain coefficients a_k .
- The error vector e can be written as a linear function of the entire block of data samples $\mathbf{x}$.
- The LSAR method then breaks down the matrix $\mathbf{A}$ into two parts: $\hat{\mathbf{A}}$ for the unknown samples and $\bar{\mathbf{A}}$ for the known samples, leading to the expression:

$$e = \hat{\mathbf{A}} * \hat{\mathbf{x}} + \bar{\mathbf{A}} * \bar{\mathbf{x}} \text{ where } \begin{cases} \hat{\mathbf{x}} = \text{vector of unknown samples.} \\ \bar{\mathbf{x}} = \text{vector of known samples.} \end{cases}$$

2. Finding the LSAR Estimate:

- The LS interpolator aims to minimize the energy E of the prediction error, which is equivalent to finding the value $\hat{x}_{LS}$ that minimizes $E = e^T * e$.
- This minimization leads to solving for $\hat{x}_{LS}$ where the gradient (*derivative*) of E with respect to $\hat{\mathbf{x}}$ is zero.
- The solution involves differentiating the energy function and setting the derivative equal to zero, which after simplification gives the least squares solution:

$$\hat{x}_{LS} = -(\hat{\mathbf{A}}^T * \hat{\mathbf{A}})^{-1} * \hat{\mathbf{A}}^T * \bar{\mathbf{A}} * \bar{\mathbf{x}}$$

4 LINEAR PREDICTIVE CODING

LPC operates on the premise that any sample in a discrete-time signal can be approximated as a linear combination of its past samples. This means that the current value of the signal can be predicted from a linear function of the previous samples.

We use LPC because it efficiently models the human voice and other signals using a minimum number of parameters, thereby being parsimonious in its use of resources. LPC is effective for encoding speech at a low bit rate, making it useful in telecommunications where bandwidth may be limited.

4.1 DATA MODEL

In LPC, we assume that a signal sample $s(n)$ at a given time n can be approximated by a linear combination of p past samples. We can mathematically express this assumption through the following formula:

$$s(n) = \sum_{k=1}^p a_k * s(n - k)$$

The a_k values are the LPC coefficients which weight the past samples. It is assumed that these coefficients are constant over a certain analysis window, meaning they do not change over a short, specified range of samples.

Moreover, if the signal is modelled as an autoregressive (AR) process, the prediction of $s(n)$ can be expressed as

$$s(n) = \sum_{k=1}^p a_k * s(n - k) + G * u(n) \text{ where } \begin{cases} a_k * s(n - k) = \text{sum of the weighted past samples.} \\ G = \text{gain parameter.} \\ u(n) = \text{white noise signal.} \end{cases}$$

The white noise represents random, uncorrelated perturbations in the signal, which add a component that cannot be predicted linearly from past samples.

In voice producing process, the voice signal $s(n)$ can be modelled similarly to the AR process, where $u(n)$ corresponds to the source excitation at the glottis and $s(n)$ is the resulting speech signal.

4.2 SOURCE – FILTER INTERPRETATION

The LPC equation that we just have seen can be transformed from the time domain into the z-domain using the z-transform:

$$S(z) = \sum_{k=1}^p a_k * z^{-k} * S(z) + G * U(z)$$

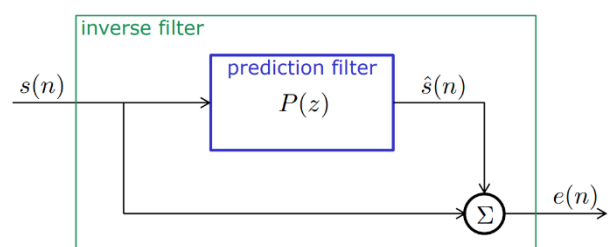
From the previous equation we can get the transfer function $H(z)$ of the filter:

$$H(z) = \frac{S(z)}{G * U(z)} = \frac{1}{A(z)} \text{ where } A(z) = 1 - \sum_{k=1}^p a_k * z^{-k}$$

This function describes how the filter modifies the spectrum of the input signal.

The building blocks of the transfer function have different names:

- **Inverse Filter:** $A(z)$ is called “inverse” or “whitening filter” because it acts to whiten or flatten the spectrum, making the signal more uniform across all frequencies.
- **Forward Filter:** $H(z)$ is termed the forward or shaping filter. It shapes the spectrum of the error signal $E(z)$ to produce the synthesized speech signal $S(z)$.



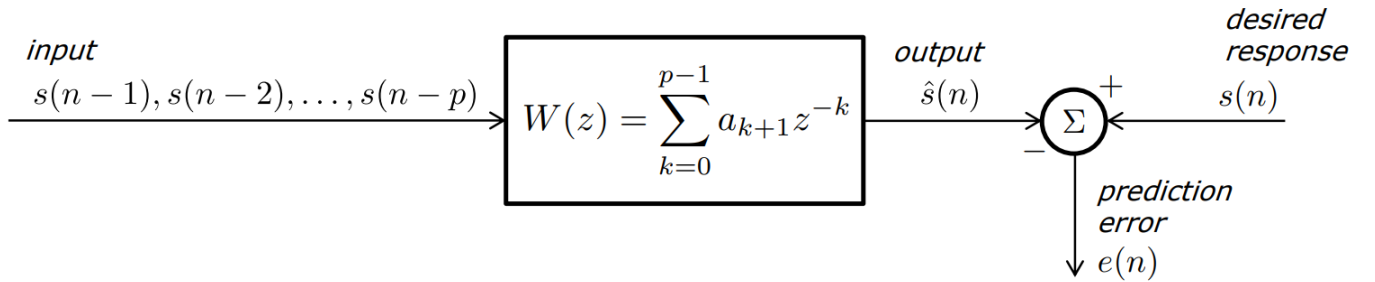
- **Prediction Filter:** $P(z)$ is the prediction filter. It generates the estimated signal $\hat{S}(z)$ from the past samples of the actual signal $S(z)$.

4.3 DETERMINE THE PREDICTOR COEFFICIENTS

The objective of LPC is to find the set of predictor coefficients $\{a_k\}_{k=1}^p$ that will minimize the mean-squared prediction error: $\min_{a_k} \{E[s(n) - \hat{s}(n)]\}$.

It is possible to set the problem as a Wiener Filtering problem. In the context of LPC, the signal $s(n)$ serves as both the input to the filter and the desired response that the filter aims to predict or estimate.

The Wiener filter to be identified corresponds to the coefficients of the prediction filter $\{a_k\}_{k=1}^p$ (the Wiener filter $W(z)$ is described in the z-domain as $W(z) = \sum_{k=0}^{p-1} a_{k+1} z^{-k}$).



As we have seen in the Wiener Filter paragraph, we can find the optimal Wiener Filter's coefficient through the Wiener-Hopf Equations. So, let's start defining the autocorrelation function:

$$r(i) = E[s(n) * s(n - i)]$$

which measures the similarity between the signal $s(n)$ and its own past values $s(n - i)$ at various time lags i . This function is crucial for LPC because it reflects the signal's intrinsic properties over time. So, the Wiener-Hopf Equations for LPC filters are:

$$\sum_{k=1}^p a_k * r(i - k) = r(i) \quad \text{for } i = 1, 2, \dots, p$$

The optimal LPC coefficients can be determined by solving the system of linear equations represented by the Wiener-Hopf equations. The coefficients are found using the matrix equation $\mathbf{a} = \mathbf{R}^{-1} * \mathbf{r}$.

$$\mathbf{a} = \mathbf{R}^{-1} \mathbf{r},$$

$$\text{where } \mathbf{R} = \begin{bmatrix} r(0) & r(1) & \dots & r(p-1) \\ r(1) & r(0) & \dots & r(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ r(p-1) & r(p-2) & \dots & r(0) \end{bmatrix}, \mathbf{r} = \begin{bmatrix} r(1) \\ r(2) \\ \vdots \\ r(p) \end{bmatrix}, \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix}$$

- $\mathbf{R}$ is the auto-correlation matrix, a symmetric and Toeplitz matrix (each descending diagonal from left to right is constant) constructed from the auto-correlation function values of the input signal.
- $\mathbf{r}$ is a column vector consisting of the auto-correlation function values $r(1)$ to $r(p)$.
- $\mathbf{a}$ is a column vector of the predictor coefficients a_1 to a_p .

4.3.1 Minimum MSE

Using Wiener theory, we can also calculate the minimum MSE and Prediction gain:

$$MSE_{min} = \sigma_s^2 - \mathbf{r}^T * \mathbf{R}^{-1} * \mathbf{r} = r(0) - \sum_{k=1}^p a_k * r(k) \quad \text{where } \begin{cases} \mathbf{r} \text{ is the vector of autocorrelation coefficients.} \\ \mathbf{R} \text{ is the autocorrelation matrix.} \\ \mathbf{R}^{-1} \text{ is the inverse of the autocorrelation matrix.} \\ \mathbf{r}^T \text{ is the transpose of the autocorrelation vector.} \\ a_k \text{ are the LPC coefficients.} \end{cases}$$

Note: $r(0)$ is the zero-lag auto-correlation of the signal and it is equal to the variance σ_s^2 of the input signal. This is the total power in the signal or the signal energy at lag zero.

4.3.2 Prediction Gain

We define the Prediction Gain G_p as the ratio $\frac{\sigma_s^2}{MSE_{min}}$. The Prediction Gain indicates how much better the LPC model's prediction is compared to a trivial prediction:

- a high G_p means that the LPC model is effectively reducing the error compared to the variance of the signal. When G_p tends towards infinity, the input signal $s(n)$ is highly predictable.
- A low G_p means that the LPC model is not effective. If G_p approaches 1, the signal is unpredictable (e.g., white noise), and the prediction doesn't provide much reduction in MSE over just using the signal mean.

Essentially, G_p is a measure of the effectiveness of the prediction.

4.3.3 Infinite memory linear prediction

Let's assume that the signal $s(n)$ is predicted using all past samples, leading to an infinite summation in the prediction formula $\hat{s}(n) = \sum_{k=1}^{\infty} a_k * s(n-k)$.

For the Orthogonality Principle, the optimum prediction error $e_{opt}(n)$ should be orthogonal to the past samples $s(n-i)$, meaning their expected cross-product is zero $E[e_{opt}(n) * s(n-i)] = 0$.

Computing the autocorrelation of the optimum prediction error $e_{opt}(n)$ we obtain:

$$\begin{aligned} r_{e_{opt}}(i) &= E[e_{opt}(n) * e_{opt}(n-i)] = E[e_{opt}(n) * (s(n-i) - \hat{s}_{opt}(n-i))] \\ &= E\left[e_{opt}(n) * \left(s(n-i) - \sum_{k=1}^{\infty} a_k * s(n-k-i)\right)\right] \\ &= E[e_{opt}(n) * s(n-i)] - \sum_{k=1}^{\infty} a_k * E[e_{opt}(n) * s(n-k-i)] = 0 \quad \forall i > 0 \end{aligned}$$

This function is zero for all non-zero lags if the LPC coefficients are optimal.

Since the auto-correlation function is even, the results obtained are also valid for negative lags $i < 0$.

The autocorrelation of the error $r_{e_{opt}}(i)$ is non-zero only when $i = 0$, indicating that the error has energy (and thus, variance) only at zero lag. This value is the minimum MSE.

$$r_{e_o}(i) = D_p \delta(i)$$

The auto-correlation of the optimum noise is a Dirac delta: the optimum prediction error is a white noise process

Infinite memory LPC can be seen as a process that extracts all the correlation information from the signal into the inverse filter $A(z)$, leaving the error $e_{opt}(n)$ as a white noise process with no sample-spanning information.

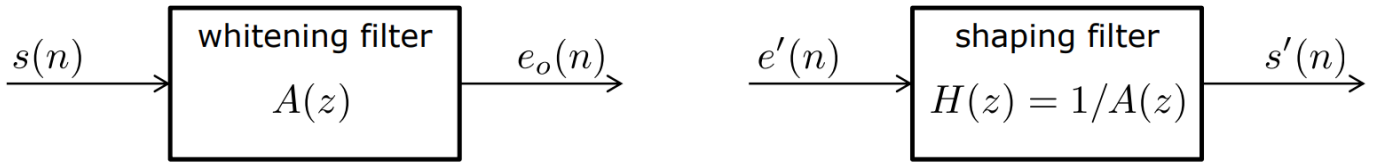
After determining the prediction error $e_{opt}(n)$, the original signal $s(n)$ can be reconstructed by passing $e_{opt}(n)$ through the forward filter $H(z)$, which is the inverse of the prediction filter $A(z)$. Any white noise process with the same variance as the prediction error can be used for signal reconstruction due to the LPC model's whiteness of the error signal. This will yield a reconstructed signal $s'(n)$ that has the same power spectrum as the original signal $s(n)$.

$$\begin{aligned} |S(e^{j\omega})|^2 &= |H(e^{j\omega})|^2 |E_o(e^{j\omega})|^2 = |H(e^{j\omega})|^2 D_p \\ |S'(e^{j\omega})|^2 &= |H(e^{j\omega})|^2 |E'(e^{j\omega})|^2 = |H(e^{j\omega})|^2 D_p \end{aligned} \quad \Rightarrow \quad |S(e^{j\omega})|^2 = |S'(e^{j\omega})|^2$$

Notice that a true whitening filter, which would completely flatten the spectrum of a signal, making it white noise, is achievable only with an infinite-memory linear prediction. Anyway, in LPC $A(z)$ is referred to as the whitening filter by convention, because it attempts to remove the predictability from the signal, leaving a white noise error $e_{opt}(n)$.

In practical scenarios, we deal with a finite prediction order. This means that while $A(z)$ will flatten the spectrum of the prediction error, it won't be a perfect white noise.

$H(z)$, which is the inverse of $A(z)$, is called the shaping filter. It is used to reconstruct the signal from the error.



4.4 IMPLEMENTATION: DEALING WITH NON-STATIONARY SIGNALS

The LPC model so far assumes the signal to be stationary, but real-world signals, like speech, are often non-stationary. Non-stationary signals have statistical properties that change over time. To deal with non-stationarity, LPC analysis is performed on short segments of the signal where the signal can be assumed to be stationary. These segments are of length M , and for each segment, the LPC parameters are calculated to minimize the short-time mean-squared error:

$$MSE_{short-time} = \epsilon_n = \sum_{m=0}^{M+p-1} \left[s_n(m) - \sum_{k=1}^p a_{k(n)} * s_n(m-k) \right]^2$$

To solve the LPC problem for each segment, the Wiener-Hopf equations are used with the short-time auto-correlation function of the signal, whose formula is the following:

$$r_n(|i-k|) = \sum_{m=0}^{M-1-(i-k)} s_n(m) * s_n(m+i-k) \text{ where } \begin{cases} 1 \leq i \leq p. \\ 0 \leq i \leq p. \\ i \geq k. \end{cases}$$

In this case, the Wiener-Hopf equations therefore are:

$$\sum_{k=1}^p a_k(n) * r_n(|i-k|) = r_n(i) \text{ for } i = 1, 2, \dots, p$$

or in matrix form as:

$$\mathbf{a}_n = \mathbf{R}_n^{-1} \mathbf{r}_n,$$

where $\mathbf{R} = \begin{bmatrix} r_n(0) & r_n(1) & \cdots & r_n(p-1) \\ r_n(1) & r_n(0) & \cdots & r_n(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ r_n(p-1) & r_n(p-2) & \cdots & r_n(0) \end{bmatrix}$, $\mathbf{r}_n = \begin{bmatrix} r_n(1) \\ r_n(2) \\ \vdots \\ r_n(p) \end{bmatrix}$, $\mathbf{a}_n = \begin{bmatrix} a_1(n) \\ a_2(n) \\ \vdots \\ a_p(n) \end{bmatrix}$

The LPC parameters for each segment a_n are computed by solving the matrix equation $a_n = R_n^{-1} * r_n$, where R_n is the auto-correlation matrix for the segment, and r_n is the auto-correlation vector.

The estimated autocorrelation leads to a minimum-phase shaping filter $A_n(z)$, which means all the zeros of the filter are inside the unit circle in the z -plane. This is important for the stability of the filter. The stability of the shaping filter $H_n(z) = \frac{1}{A_n(z)}$ is guaranteed because of the minimum-phase nature of $A_n(z)$.

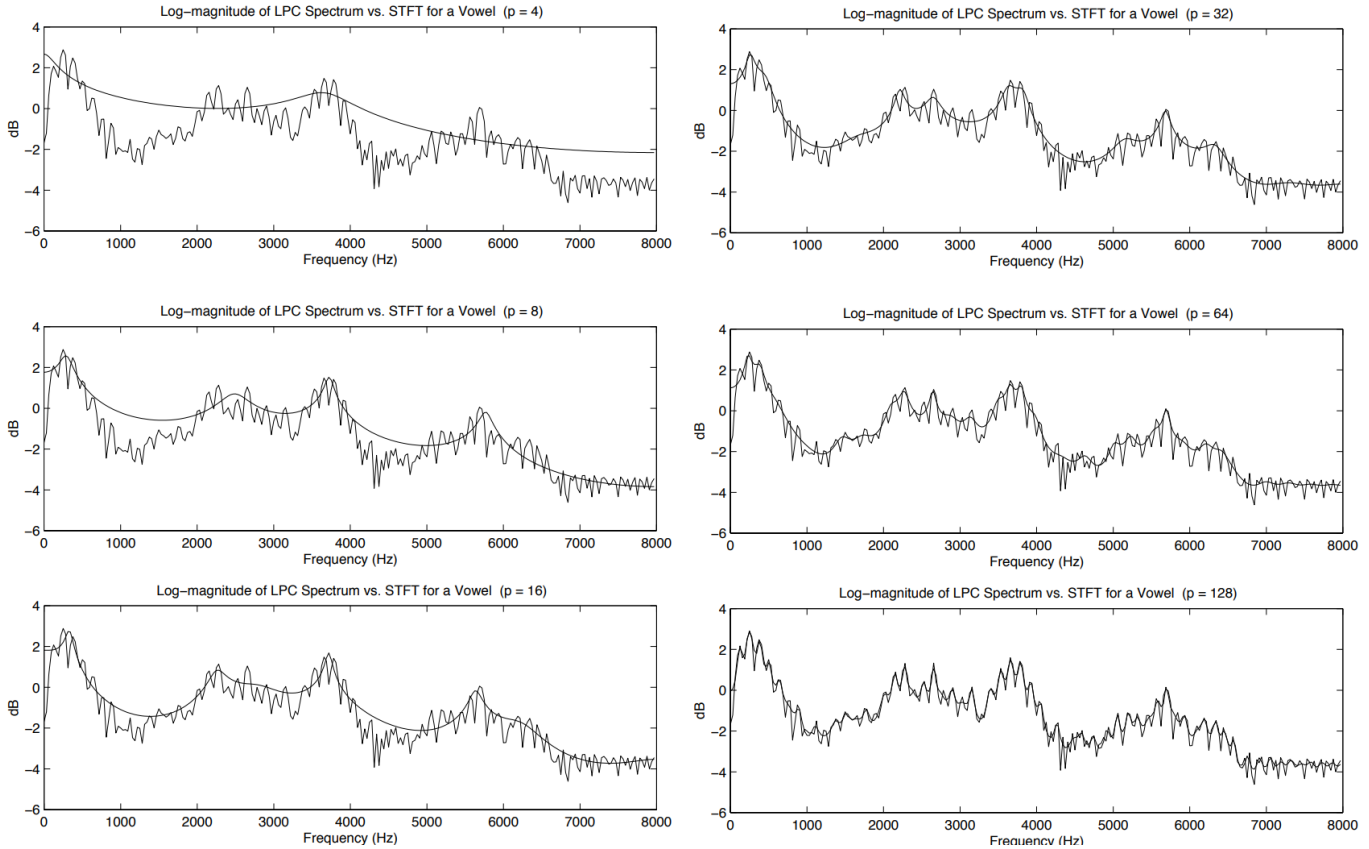
4.5 POWER SPECTRUM ENVELOPE MATCHING

In the case of infinite memory LP, the power spectrum of the signal can be exactly reconstructed from the shaping filter and the error variance. This means that as the prediction order p increases (tends to ∞), the approximation of

the power spectrum of the signal becomes more accurate with an arbitrarily small error using the all-pole shaping filter $H_n(\hat{z})$.

$$\lim_{p \rightarrow \infty} |\hat{S}_n(e^{j\omega})|^2 = |S_n(e^{j\omega})|^2$$

As the prediction order increases, the resulting mean-squared error monotonically decreases.



The short-time error is limited to a specific interval, and thus, the mean-squared error ϵ_n can be expressed over the interval or assumed to be over an infinite interval. Using Parseval's theorem, which relates the time-domain energy of a signal to its frequency-domain energy, the mean-squared error can be expressed in the frequency domain. After some algebraic passages, we get that:

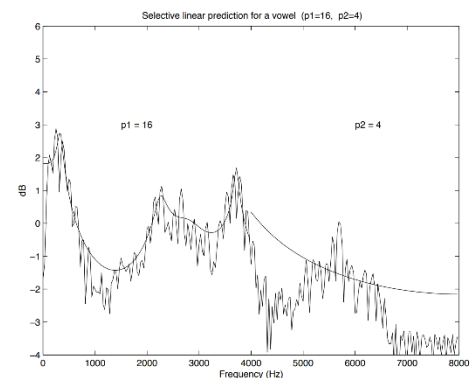
$$\min_{a_n} \epsilon_n \leq \min_{a_n} \frac{|S_n(e^{j\omega})|^2}{|H_n(e^{j\omega})|^2}$$

From what we said, we can deduce that the prediction order p is a very important parameter and it is necessary to choose it accurately. A rule of thumb for speech could be to choose $\frac{f_s}{1000} < p < \frac{f_s}{1000} + 4$.

4.6 FREQUENCY SELECTIVE LPC

Frequency selective LPC is a method to perform LPC analysis focusing on specific parts of the frequency spectrum, especially for non-stationary signals like speech that have different characteristics in different frequency bands.

In this LPC method, the prediction order p can be adjusted for different partition of the spectrum to control the accuracy and smoothness of the LPC spectrum. This is especially useful for analyzing different frequency bands of non-stationary signals.



For example, in speech the lower frequency range (0-4 kHz) is important for voiced speech sounds like vowels, while the higher frequency range (4-8 kHz) is significant for unvoiced sounds like fricatives. In that case we could use a higher order for lower frequencies where more detail is required and a lower order for higher frequencies.

To model a specific frequency range is necessary to perform the following steps:

1. We first map the frequency interval to a normalized frequency range:

$$\omega \in [2\pi f_A, 2\pi f_B] \Rightarrow \omega' \in [0, 2\pi]$$

2. New auto-correlation coefficients for the selected frequency range are obtained using the Inverse Discrete-Time Fourier Transform (IDTFT):

$$r'_n(k) = \frac{1}{2\pi} * \int_{-\pi}^{\pi} |S_n(e^{j\omega})|^2 * e^{j\omega'k} d\omega'$$

3. These new auto-correlation coefficients are then used to solve a new set of Wiener-Hopf equations to find the predictor coefficients for the selected spectral region.

4.7 APPLICATIONS OF LPC

LPC can be used to model the process of speech production, where $e_n(m)$ is considered the excitation at the glottis and $H_n(z)$ is the transfer function of the vocal tract. By encoding the LPC parameters, significant data compression can be achieved, making it efficient for digital storage and transmission. LPC also allows distinguishing between voiced and unvoiced segments in speech, which require different kinds of excitation signals.

- Voiced sounds are typically modelled using a train of pulses (periodic excitation). To extract pitch information, it's necessary to examine the prediction error of voiced speech segments. The periodicity in the error corresponds to the pitch of the voiced sound. By identifying the peaks in the prediction error and measuring the average distance between them, one can estimate the pitch. A more robust pitch estimation is achievable with long-term prediction, where the pitch period guides the prediction of current samples based on past samples.
- Unvoiced sounds are modelled with white noise (random excitation).

Talking Instruments: LPC can also be utilized for cross synthesis in computer music, such as creating "talking" musical instruments. This involves feeding a musical signal through the shaping filter derived from speech signal LPC analysis, transferring vocal characteristics to the instrument's sound. The result combines the musical instrument's harmonic structure with the formant (resonant frequency) structure of the vocal signal, leading to a hybrid sound that blends musical and vocal qualities.

5 TIME AND PITCH SCALING

The primary goal that we would like to achieve is to independently control both the time evolution and pitch of audio signals:

- **Time Scaling:** This involves controlling and modifying how a signal evolves over time without affecting its spectral components. The purpose of time scaling is to adjust the speed of the signal, either to slow it down or speed it up, while keeping its spectral content, such as frequency and harmonics, unchanged.
- **Pitch Scaling:** This process involves controlling and modifying the pitch of a signal without altering its temporal evolution. Pitch scaling allows the pitch of the signal to be altered, possibly in a dynamic manner, without changing the duration or timing of the signal.

It's not so easy to reach these goals: time and frequency domains of a signal are intrinsically linked by definition. This relationship is established through the Fourier transform, which translates a signal from the time domain to the frequency domain and vice versa: when we attempt to scale the time or pitch, the other is usually affected due to this interdependency.

To perform independent scaling of time and pitch, we must "manipulate" the Fourier scaling theorem so that time scaling and pitch scaling can be done without the usual corresponding changes in the other domain. Warping the time axis leads to an inverse warping of the frequency axis according to the theorem.

5.1 IDEAL TIME SCALING

Ideal time scaling allows for the manipulation of an audio signal's timing without affecting its pitch. The methodology uses a sinusoidal signal model, which assumes that any complex signal can be decomposed into a sum of sinusoids.

Consider having a signal (*sum of sinusoid*) expressed like that:

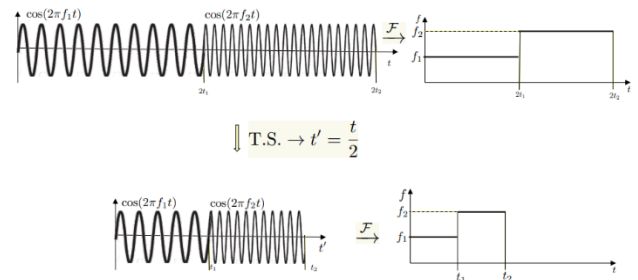
$$x(t) = \sum_{i=1}^{I(t)} A_i(t) e^{j\phi_i(t)} \text{ with phase } \phi_i(t) = \int_{-\infty}^t \omega_i(\tau) d\tau \text{ where } \begin{cases} A_i(t) = \text{amplitude of the } i_{\text{th}} \text{ sinusoid at time } t \\ \omega_i(\tau) = \text{frequency of the } i_{\text{th}} \text{ sinusoid at time } \tau \end{cases}$$

When we time-scale the signal, we apply a **time warping function**, which is a function that maps the original time t and the time in the modified signal $t' = T(t)$. This means that the instantaneous amplitude of the sinusoids at the new time t' corresponds to the amplitude of the sinusoids of the original signal at the original time $T^{-1}(t')$.

Considering this, we obtain that the new signal is:

$$x(t') = \sum_{i=1}^{I(T^{-1}(t'))} A_i(T^{-1}(t')) e^{j\phi_i(t')} \text{ with phase } \phi_i(t') = \int_{-\infty}^{t'} \omega_i(T^{-1}(\tau)) d\tau$$

It is important to note that this method controls the magnitude and frequency of the sinusoids without affecting the phase.



5.2 IDEAL PITCH SCALING

Pitch Scaling consists in ideally scale the pitch of an audio signal while keeping its timing unchanged.

As before, consider a signal $x(t)$ that is a sum of sinusoids with varying amplitudes and phases:

$$x(t) = \sum_{i=1}^{I(t)} A_i(t) e^{j\phi_i(t)} \text{ with phase } \phi_i(t) = \int_{-\infty}^t \omega_i(\tau) d\tau \text{ where } \begin{cases} A_i(t) = \text{amplitude of the } i_{\text{th}} \text{ sinusoid at time } t \\ \omega_i(\tau) = \text{frequency of the } i_{\text{th}} \text{ sinusoid at time } \tau \end{cases}$$

The pitch-scaled signal $x'(t')$ retains the same sinusoidal model structure. However, the instantaneous frequency of each sinusoid is modified by a factor $\alpha(t)$, the pitch modification factor, which can vary over time.

$$x(t') = \sum_{i=1}^{I(t')} A_i(t') e^{j\phi'_i(t')} \text{ with phase } \phi'_i(t') = \int_{-\infty}^{t'} \alpha(\tau) * \omega_i(\tau) d\tau$$

This means that the pitch of the audio signal can be raised or lowered without affecting the timing of the signal's components, such as the rhythm or speed of music or speech.

5.3 IDEAL COMBINED TIME AND PITCH SCALING

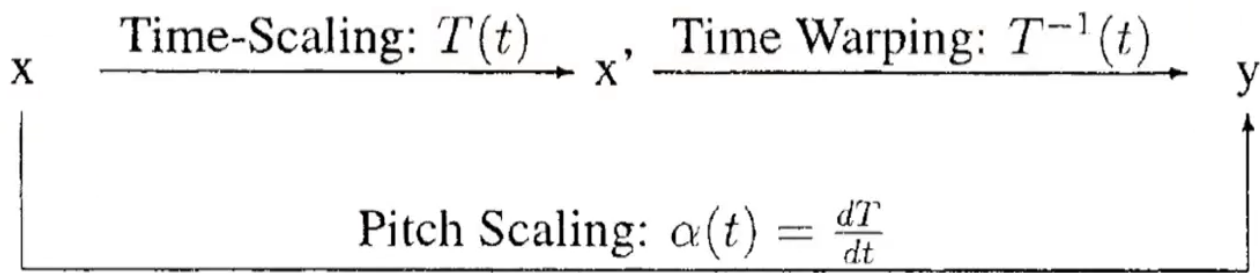
It is possible to combine the two previous concepts in a model for scaling both the time and pitch of an audio signal simultaneously.

In that model, the signal $x'(t')$ is modified using both the time warping function $T(t)$ and the pitch modification function $\alpha(t)$ allowing for the simultaneous adjustment of timing and pitch.

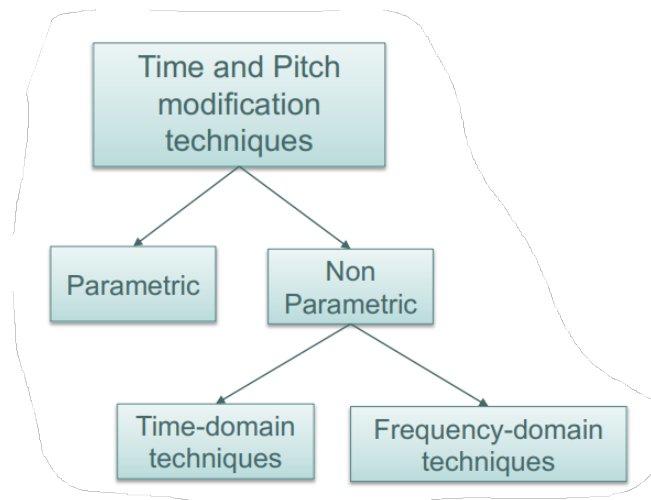
5.4 DUALITY PRINCIPLE

The duality principle in audio signal processing connects time scaling and pitch scaling through the lens of time warping. The duality principle asserts that time scaling can be achieved through a combination of pitch scaling and time warping, and vice versa. Time warping here is essentially a form of resampling. This foundational principle

allows us to interchangeably use pitch scaling and time warping to achieve time scaling, demonstrating their interdependent relationship in signal processing.



5.5 METHODS



5.6 PARAMETRIC METHODS

In parametric frequency-based methods, time and pitch scaling are defined within a sinusoidal model, which breaks down complex audio signals into simpler sinusoidal components. This model allows for direct manipulation of these components, offering a precise strategy for implementing changes in time and pitch.

The strategy for applying time and pitch scaling involves several steps:

- **Analysis:** Using the Short-Time Fourier Transform (STFT) or a filter bank to analyze the audio signal. These techniques convert the signal from the time domain to the frequency domain, making it possible to examine its spectral content.
- **Peak Detection and Tracking:** Identifying and following the peaks in the frequency spectrum over time. These peaks represent the fundamental frequencies and their harmonics, which are essential for perceiving the pitch and timbre of the sound.
- **Modification of Partial:** Adjusting the detected spectral peaks (partials). This might involve altering their frequencies for pitch scaling or changing their timing for time scaling.
- **Synthesis:** Reconstructing the audio signal from the modified sinusoidal components using a bank of oscillators. Each oscillator corresponds to a particular partial, and together they recreate the modified signal.

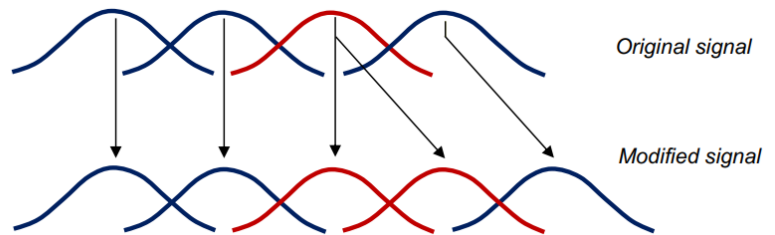
The approach can be broadened from a pure sinusoidal model to a more complex sines plus noise model. This expansion allows for the representation and manipulation of non-harmonic or noisy components of the signal, which are not captured by simple sinusoids.

5.7 NON-PARAMETRIC METHODS (TIME DOMAIN)

A non-parametric method does not rely on a parameterized model of the signal. It treats the signal as an aggregate of its STFT frames, which can be independently manipulated. This allows for flexible and direct manipulation of the signal in the time-frequency domain, useful for tasks like time-stretching, pitch-shifting, and other audio effects.

5.7.1 Pitch-Independent Methods

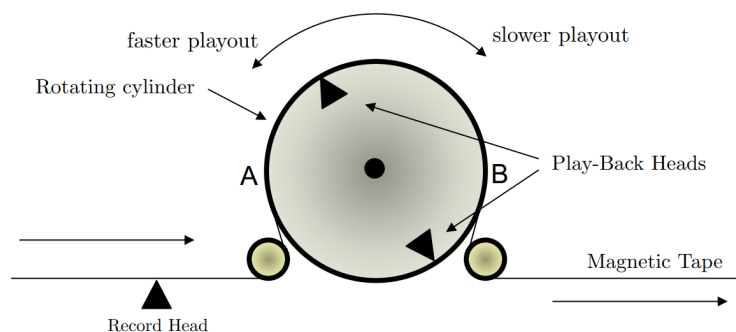
Pitch-independent methods aim to change the duration of an audio signal without affecting its pitch. This is accomplished through the following process:



- **Segmentation:** The audio signal is decomposed into short segments, typically ranging from 10 to 40 milliseconds. These segments are short enough that they can be manipulated without noticeable effects on the pitch.
- **Time-scaling:** This process can either compress or expand the time scale of the audio signal. Time compression is achieved by discarding some segments, while time expansion is done by repeating certain segments. By changing which segments are repeated or discarded, the overall duration of the audio is altered.
- **Overlap and Weighting:** To ensure the signal remains smooth and to avoid abrupt changes or discontinuities between segments, the segments are overlapped. They are then multiplied by weighting windows, which helps in smoothing out the transition between consecutive segments.

5.7.2 Modified Tape Recorder

The modified tape recorder is an analog device that was used to alter the pitch and duration (time-scale) of audio signals before the advent of digital signal processing. It used a rotary playback head with multiple read heads.



Let's analyse how it works.

5.7.2.1 Modified Tape Recorder: Pitch Scaling

- **Fixed Record Head:** The audio signal is initially recorded onto a magnetic tape using a fixed record head.
- **Speed of Rotation:** The pitch of the playback signal is altered by changing the speed of the rotating playback heads:
 - When the cylinder rotates faster than the recording speed (counter-clockwise), the playback heads read the tape more quickly, which raises the pitch.
 - When the cylinder rotates slower than the recording speed (clockwise), the playback heads read the tape more slowly, lowering the pitch.

Notice that the speed at which the tape moves past the playback heads is constant. It doesn't speed up or slow down, which means the duration of the track being read does not change. The playback heads are attached to a rotating cylinder. As this cylinder rotates, different heads come into contact with the tape at different positions.

5.7.2.2 Modified Tape Recorder: Time Scaling

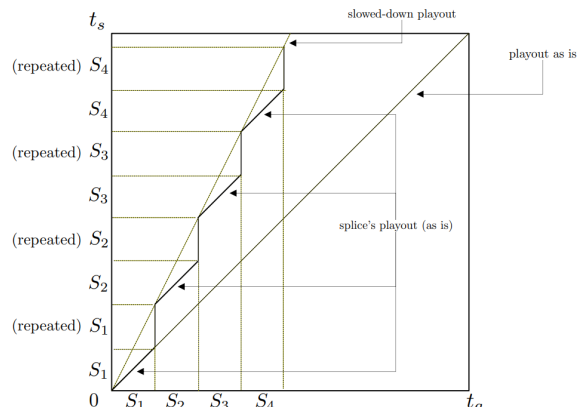
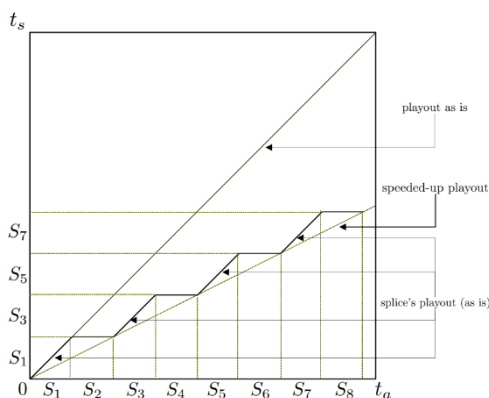
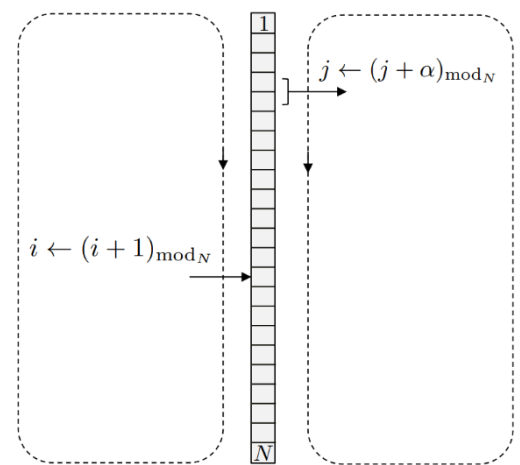
- **Rotary Motion:** The cylinder rotation direction relative to tape motion controls time scaling. The opposite direction accelerates playback, condensing the time-scale, while the same direction slows it down, expanding the time-scale.
- **Repeated Segments:** If the playback heads rotate in such a way that they leave the tape at one point and come back into contact at a previous point, the audio signal recorded between these points is played back multiple times, extending the duration of the audio.
- **Discarding Segments:** Conversely, if the cylinder rotates in the same direction as the tape motion and a playback head does not come back into contact with the tape at the previous recorded segment, that part of the audio signal is skipped, effectively shortening the duration.

The segments read by the playback heads have a fixed duration. This ensures consistent rhythm and timing during the pitch and time-scaling processes. The points at which segments are joined or spliced are evenly spaced, providing a regular rhythm to the playback. As playback heads move in and out of contact with the tape, the audio signal naturally fades in and out. This cross-fading avoids abrupt changes and creates smooth transitions between pitch-shifted segments.

5.7.2.3 Modified Tape Recorder: Digital Implementation

In digital signal processing, pitch scaling can be achieved through a digital buffer, a kind of storage register that has a pre-defined length, N . This buffer can be imagined as a circular array or a ring buffer. It has two main pointers or indices:

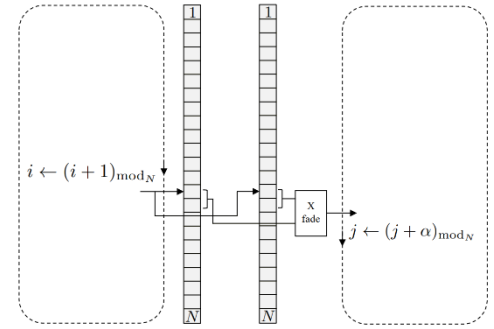
- **Writing Index (i):** This index corresponds to the record head in an analog tape recorder. It moves in discrete integer steps, simulating the tape moving past a record head.
- **Reading Index (j):** This index corresponds to the playback head. It moves in fractional steps because digital audio allows for very precise positioning within the buffer, unlike the discrete physical movement in analog devices.
- **Pitch Scaling (Scaling Factor α):** The rate of change of the reading index j is controlled by the scaling factor α . If $\alpha > 1$, the reading index moves faster than the writing index, meaning the frequencies go up (pitch scaling up). If $\alpha < 1$, it moves slower, meaning the frequencies go down (pitch scaling down).
- **Time Scaling:**
 - **Speed up:** When the reading index overtakes the writing index (or vice versa), this is equivalent to the playback head reading the same tape segment twice (or skipping a segment), which corresponds to a splice in the digital domain.
 - **Speed Down:** Because the reading index moves in fractional steps, there is a need for interpolation. Interpolation calculates the sample values between discrete data points in the buffer, allowing for smooth playback even when reading at a non-integer position in the buffer.



Crossfading is necessary whenever one index passes the other to prevent abrupt changes in the audio. This involves gradually decreasing the volume of one splice while increasing the volume of the next, creating a seamless transition between segments. In a digital system, this may involve blending the end of one buffer with the beginning of the next or merging two overlapping samples.

To ensure smooth transitions:

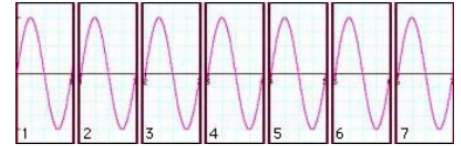
- A cross-fading algorithm is applied whenever the reading index overtakes the writing index, blending the contiguous splices to avoid any perceptible jumps or drops in the audio.
- This might involve doubling the buffer (replicating the circular register) to handle the cross-fade without disrupting the continuous flow of data.



5.7.3 SOLA

The SOLA method, or Synchronous Overlap and Add, is a nonparametric time-domain technique for modifying the time-scale of an audio signal without changing its pitch. It is particularly effective with signals that have some degree of periodicity, like speech or music.

SOLA works on the principle of waveform similarity. For periodic or nearly periodic signals, if you change the time-scale of the signal by repeating or skipping segments, the manipulation is less noticeable if these segments are an integer multiple of the period of the signal. In this way, the perceived pitch of the signal remains consistent, as pitch is related to the signal's periodicity.



The steps to perform in SOLA are:

1. **Analysis:** The original signal is divided into overlapping segments. These segments are called analysis blocks, and their length is related to the periodicity of the signal.
2. **Modification:** Depending on whether you want to stretch or compress the signal, you would either repeat or skip some of these blocks.
3. **CV**

To ensure smooth transitions between repeated or skipped segments, SOLA employs cross-correlation to determine the optimal overlap point. This involves shifting the segments relative to each other and calculating the similarity between overlapping portions. The shift that results in the highest cross-correlation value indicates the best alignment for splicing the segments together:

$$k_s = \operatorname{argmax}_k \{c_c(t_a^u, k)\} \text{ where } c_c(t_a^u, k) = \frac{(\sum_{i=1}^{N_c} x(t_a^u + i)x(t_a^u + i + k))}{\sqrt{\sum_{i=1}^{N_c} x^2(t_a^u + i)} * \sqrt{\sum_{i=1}^{N_c} x^2(t_a^u + i + k)}}$$

As an alternative to cross-correlation, the Average Magnitude Difference Function can be used. This function measures the average absolute difference between the amplitudes of two overlapping segments. Minimizing the AMDF value helps find the point of minimal difference, indicating a good splice point:

$$k'_s = \operatorname{argmin}_k \{f_c(t_a^u, k)\} \text{ where } f_c(t_a^u, k) = \frac{1}{N_c} * \sum_{i=1}^{N_c} |x(t_a^u + i) - x(t_a^u + i + k)|$$

k_s and k'_s are actually very similar.

5.7.4 PSOLA

PSOLA (Pitch-Synchronous Overlap and Add) is a technique used in signal processing to modify the pitch and duration of speech signals without introducing significant artifacts. It operates under the assumption that the pitch

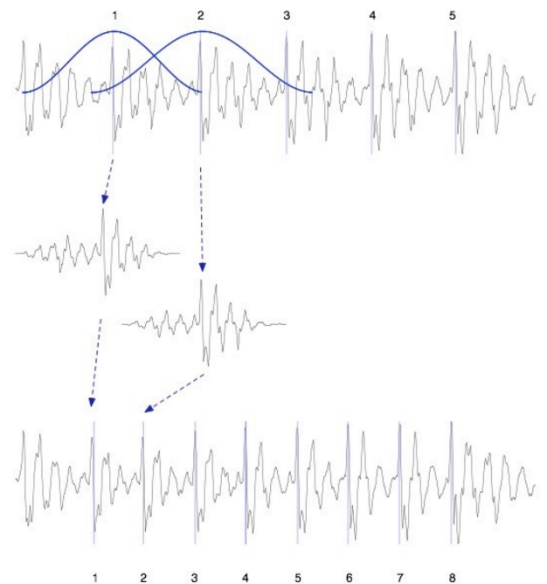
(fundamental frequency) of the speech is known. It also requires knowledge of the locations of the glottal pulses, which are the peaks of pressure resulting from the rapid closure of the vocal folds during speech production.

The process involves segmenting the speech signal into small sections around each glottal pulse and then overlapping and adding these sections at a different rate.

5.7.4.1 PSOLA Algorithm

Let's walk through the detailed steps of the PSOLA algorithm:

1. **Pitch Detection and Marking Glottal Pulses:** The first step in PSOLA is to detect the fundamental frequency (pitch) of the speech signal. This is typically done using a pitch detection algorithm. Once the pitch is known, the algorithm marks the glottal pulses, which correspond to the points in the signal where the vocal cords come together and create a peak in the waveform. These marks are crucial as they are the synchronization points around which the PSOLA manipulation will occur.
2. **Segmenting the Signal:** The speech signal is divided into small segments called "grains" or "windows," each centred around a glottal pulse mark. The length of these segments is generally a few pitch periods long to ensure they contain enough of the waveform to maintain the characteristics of the speech.
3. **Time or Frequency Scaling:**
 - **Overlap and Add for Time-Stretching or Compression:** For *time-stretching* (making the speech longer), the algorithm overlaps consecutive grains and adds them together. This overlapping can involve some degree of crossfading to ensure smooth transitions between grains. For *time-compression* (making the speech shorter), some grains are skipped over, and the remaining ones are overlapped and added. The key here is that overlapping grains maintain their relative pitch periods, preserving the original pitch characteristics of the speech.
 - **Overlap and Add for Pitch Shifting:** To increase pitch, grains are taken from the original signal and added together more frequently (shortening the period between grains). To decrease pitch, grains are added together less frequently (increasing the period between grains). The overlap ensures that the formant structure (which gives speech its characteristic quality) remains relatively unchanged, as abrupt changes to the formant structure can make the speech sound unnatural.
4. **Re-windowing:** After overlapping and adding grains, the signal may have some discontinuities due to phase and amplitude differences at the splice points. To smooth these out, the algorithm applies a window function to the overlapped grains to ensure that the added signal fades in and out smoothly. This re-windowing process minimizes audible artifacts.
5. **Reconstruction:** Finally, the modified grains are pieced together to form the continuous output signal. If pitch shifting was performed, the reconstructed signal will have a different pitch but the same duration as the original. If time-stretching was done, the pitch remains the same, but the duration will be altered.



Thanks to the duality theorem, using a time scaler followed by reverse time warping (which is not an exact process due to the complexity of speech signals) can produce a pitch scaler, but this can affect the voice formants.

High-quality PSOLA depends on accurate pitch marking. Errors in pitch detection can lead to artifacts in the processed speech. The chosen window length and shape also affect the quality of the output signal. Generally, Hann

or Hamming windows are used because of their smooth fade-in and fade-out characteristics. To avoid artifacts like "phasiness" or "robotic" sounds, PSOLA implementations often include additional steps like smoothing the transitions between windows or dynamically adjusting window lengths based on signal content.

5.7.5 Remarks on Time Domain Methods

Time-domain methods are generally computationally inexpensive, which means they require less processing power and can operate faster than their frequency-domain counterparts. This makes them suitable for applications that require real-time processing, where speed is essential. They perform well when the required modification to the pitch or time scale of the signal is small. For slight adjustments, these methods can provide satisfactory results without introducing significant artifacts.

Their computational efficiency makes them well-suited for real-time implementations. This implies that they can be used in systems that require immediate processing and output of the audio signal, such as live audio effects or interactive applications. Because of their simplicity and efficiency, time-domain methods can be easily incorporated into consumer products. This includes devices and software applications like telephone answering systems and smartphone apps, where the ability to quickly process audio is beneficial.

Despite these advantages, time-domain methods do not perform well in the presence of heavy reverberations, echoes, or background noise. These conditions can introduce complexities that time-domain methods are not robust against, leading to decreased performance and more noticeable artifacts. Moreover, when it comes to making significant changes to the time or pitch (for example, doubling the speed or significantly altering the pitch), time-domain methods may not be suitable. In such cases, more elaborate frequency-domain techniques are typically required because they offer finer control and can handle the complex interactions between different components of the signal.

5.8 NON-PARAMETRIC METHODS (FREQUENCY DOMAIN)

5.8.1 STFT Based Algorithm (Time Scaling)

The Short-Time Fourier Transform (STFT)-based methods for signal processing involve a combination of techniques that allow us to manipulate a signal in the frequency domain to achieve certain effects, such as time scaling or pitch scaling. The process is split into three key phases: Analysis, Scaling, and Synthesis.

5.8.1.1 Analysis

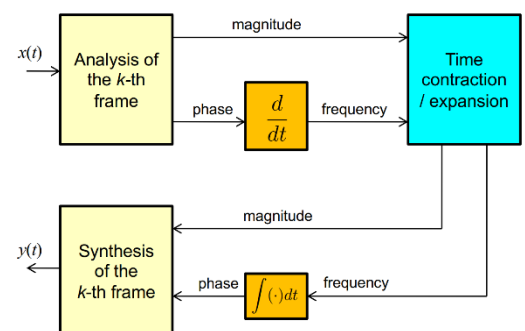
The analysis phase involves computing the STFT at a constant rate, which yields the magnitude and phase of the signal. The Short-Time Fourier Transform involves taking segments or 'frames' of the signal and applying the Fourier Transform to each. Mathematically, the STFT of a signal $x(t)$ at time t_u and frequency Ω_k is given by:

$$X(t_u, \Omega_k) = \sum_{n=-\infty}^{\infty} h(n) * x(t_u + n) * e^{-j\Omega_k * n} \quad \text{where} \quad \begin{cases} h(n) = \text{window function.} \\ \Omega_k = \frac{2\pi * k}{N} = \text{frequency bins.} \end{cases}$$

5.8.1.2 Scaling

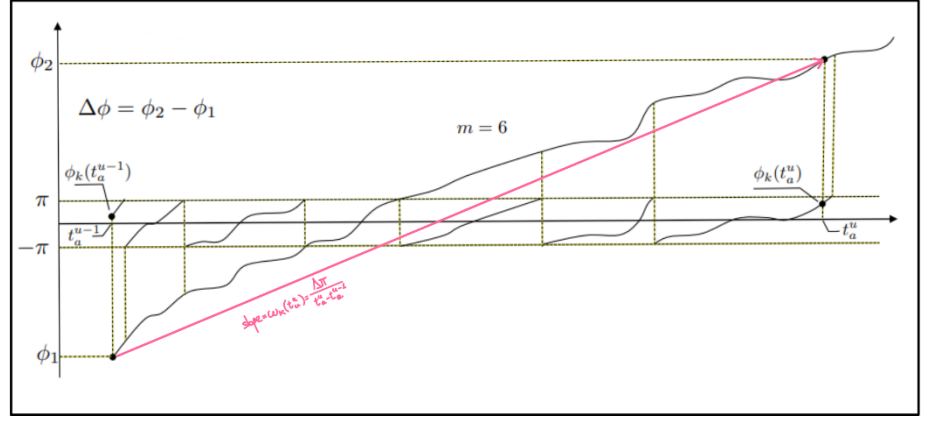
In the scaling phase, we perform the following operations:

- **Compute the STFT:** The Short-Time Fourier Transform (STFT) decomposes a signal into its constituent frequencies over short time intervals. For each frame (a segment of the signal multiplied by a window function), the STFT gives us complex numbers that represent the magnitude and phase of the signal at various frequencies. The magnitude tells us how much of a frequency is present in that frame, and the phase tells us the alignment of the wave at that frequency.
- **Apply time warping to the time axis,** which may require pitch scaling as per the duality theorem.



- **Perform phase unwrapping:**

The phase information provided by the STFT is wrapped within the range of $[-\pi, \pi]$. This means that if the true phase exceeds this range, it wraps around. To address these discontinuities, we unwrap the signal phase adding multiples of 2π to the wrapped phase differences to ensure a smooth phase transition from frame to frame.



The algorithm to do this is the following:

1. **Set initial variables:** The initial phases for each frequency bin Ω_k are set using the STFT of the initial signal segment at time zero.

$$\hat{\phi}(0, \Omega_k) = \angle X(0, \Omega_k)$$

We must also determine the next points in time for the synthesis (t_s^{u+1}) and analysis (t_a^{u+1}) where u is the current frame index. R represents the hop size or the time difference between consecutive frames for the synthesis.

2. ***Compute Instantaneous Frequency:** For each frequency channel, calculate the instantaneous frequency ($\omega_k(t_a^u + 1)$) at the next analysis instant. The calculation incorporates the unwrapped phase difference ($\Delta\phi$) and compensates for the wrapped phase by subtracting $2\pi m$ if needed, where m is an integer representing how many times the phase has wrapped.

The phase unwrapping formula is:

$$\Delta\phi = \phi_2 - \phi_1 = \Delta\phi_\omega + m * 2\pi \text{ where } \begin{cases} \Delta\phi = \text{actual phase difference between consecutive frames.} \\ \phi_2 = \text{phase at the current STFT frame.} \\ \phi_1 = \text{phase at the previous STFT frame.} \\ \Delta\phi_\omega = \text{wrapped phase difference directly obtained from the STFT.} \\ m = \text{represents the number } 2\pi \text{ added to wrapped phase difference.} \end{cases}$$

and we get the instantaneous phase with the formula:

$$\omega_k(t_a^u) = \frac{\Delta\phi}{t_a^u - t_a^{u-1}} \text{ where } \begin{cases} \omega_k(t_a^u) = \text{instantaneous frequency of } h_{k_{th}} \text{ component at the current frame.} \\ t_a^u = \text{The time corresponding to the current analysis frame.} \\ t_a^{u-1} = \text{The time corresponding to the previous analysis frame.} \end{cases}$$

or with the following formula (*implementation*):

$$\omega_k(t_a^u) = \Omega_k + \frac{1}{R(u)} (\Delta\phi - \Omega_k R(u) - 2m\pi) \text{ where } \begin{cases} \Omega_k = \text{center frequency of the } k_{th} \text{ Fourier analysis bin.} \\ R(u) = \text{time scaling factor (how much time has been wrapped)} \end{cases}$$

3. **Compute Instantaneous Phase:** Calculate the new instantaneous phase for the next synthesis frame, taking into account the phase evolution over time.

$$\hat{\phi}_k(t_s^u) = \hat{\phi}_k(t_s^{u-1}) + (t_s^u - t_s^{u-1})\omega_k(T^{-1}(t_s^u))$$

4. **Reconstruct Time-Scaled STFT:** Using the inverse mapping function $T^{-1}(t_s^u)$, which adjusts the time-scaling, and the newly computed phase $\phi^k(t_s^u)$, reconstruct the STFT for the next synthesis frame.

$$Y(t_s^u, \Omega_k) = |X(T^{-1}(t_s^u), \Omega_k)| e^{j\hat{\phi}_k(t_s^u)}$$

5.8.1.2.1 Synthesis

The inverse STFT is then used to reconstruct the signal from its Fourier representations. The reconstruction is perfect if the synthesis frames overlap-add (OLA) correctly, meaning the summation of the windowed segments is equal to one at every point in time.

The steps to perform are the following:

1. **Compute Output Term:** Generate the next output sample $y(n)$ by summing over all synthesized STFT frames using the overlap-add method. This step essentially synthesizes the time-scaled signal. A general synthesis formula for the signal $y(n)$ can be written as:

$$y(n) = \sum_u w(n - t_u^s) * \frac{1}{N} \sum_{k=0}^{N-1} Y(t_u^s, \Omega_k) * e^{j\Omega_k(n-t_u^s)} \quad \text{where} \quad \begin{cases} Y(t_u^s, \Omega_k) = \text{modified STFT coefficients} \\ w(n - t_u^s) = \text{synthesis window} \end{cases}$$

2. **Update Frame Index:** Increment the frame index u by one to process the next segment of the signal:
 $u = u + 1$
3. **Iterate:** Return to step * and repeat the process for the next frame.

5.8.2 STFT Based Algorithm (Pitch Scaling)

Starting from the Time Scaling STFT Based Algorithm, we can obtain a pitch scaling thanks to the duality theorem:

- **Time-Scaling:** First, we pretend to either stretch or squeeze the signal in time. If you stretch it, the pitch goes down (like slowing down a record player), and if you squeeze it, the pitch goes up (like speeding up a record player).
- **Resampling (Time-Warping):** After we've changed the pitch, the audio will be either longer or shorter in duration than we want. So, we use a "trick" resampling it to put it back to its original length without changing the new pitch we've just created.

$$x \xrightarrow[\alpha(t) = \frac{dT}{dt}]{\text{Pitch-Scaling}} y \quad \longleftrightarrow \quad x \xrightarrow[T(t)]{\text{Time-Scaling}} x' \xrightarrow[T^{-1}(t)]{\text{Time-Warping}} y$$

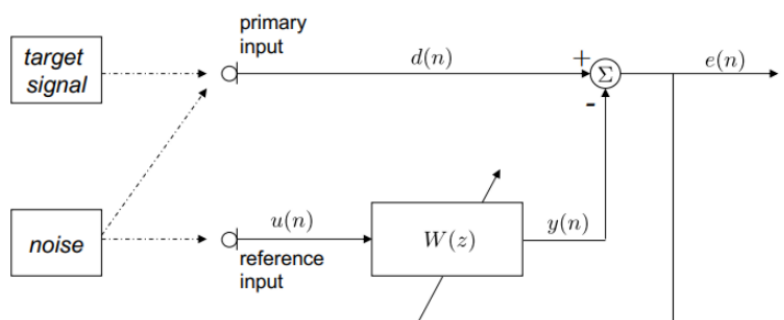
6 APPLICATIONS OF ADAPTIVE FILTERING (WIENER FILTER)

6.1 NOISE CANCELLATION

The process begins with a target signal that is corrupted by noise. The goal is to remove this noise. An adaptive filter, specifically a Wiener filter, is used for this purpose. The filter takes a noisy signal as input and produces an estimate of the noise which is then subtracted from the original noisy signal to yield the filtered signal.

Mathematically, this process involves several steps:

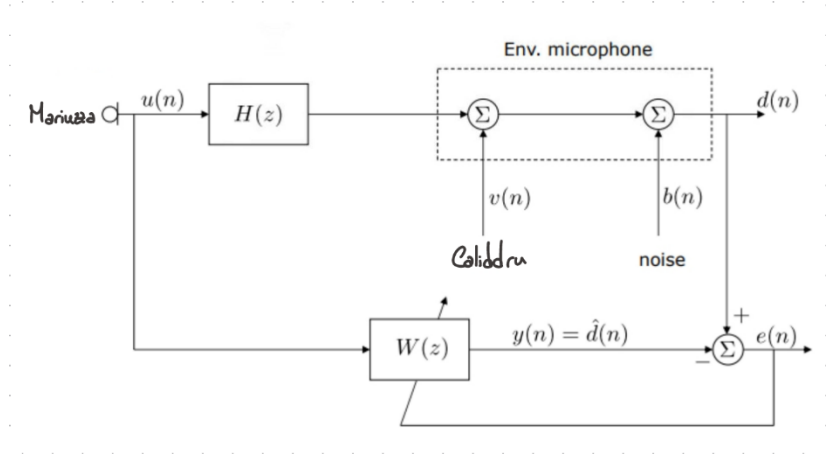
- The reference noise signal $u(n)$ is filtered through $W(z)$, resulting in $y(n)$.
- The target signal $d(n)$ has the estimated noise $y(n)$ subtracted from it, giving the error signal $e(n)$.
- The filter is updated iteratively with the formula $w(n+1) = w(n) + \mu u(n)e(n)$, where μ is a step size parameter for the update.



6.2 ACOUSTIC ECHO CANCELLATION

Acoustic echo cancellation is necessary when a signal is reflected, causing an echo that is picked up by microphones. This is addressed using Wiener filtering as well. The impulse response's length is crucial because it affects filtering performance.

In a scenario where one voice is live (Mariuzza) and another is through a speaker (Caliddru), their mixed signal is captured by an environmental microphone. By having a reference of Mariuzza's voice, Wiener filtering is used to remove her voice from the mixed signal, leaving Caliddru's echoed voice, which is also processed.



The effectiveness of this cancellation is measured by Echo Return Loss Enhancement (ERLE), which is the ratio between the energy of the echo before and after cancellation, given in decibels (dB).

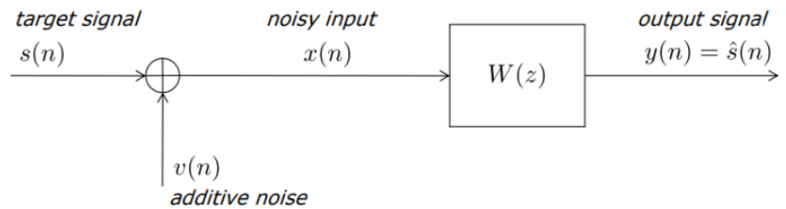
Mathematically:

- The filtered reference input $y(n)$ is equal to $W(z)$ applied to $u(n)$, Caliddru's voice.
- The residual error $e(n)$ is calculated by subtracting $y(n)$ from the total signal $d(n)$.
- ERLE is calculated using the formula $ERLE[dB] = -10 * \log_{10} \left(\frac{\sum [e(n) - v(n)]^2}{\sum [d(n)]^2} \right)$.

6.3 BACKGROUND NOISE REMOVAL

Let's assume that the noise $v(n)$ and signal $s(n)$ are independent and uncorrelated and that an estimate of the noise spectrum $P_v(\omega)$ is available.

The objective is to design a filter $W(z)$ that minimizes the Mean Square Error (MSE) between the signal and the filtered output. According to the orthogonality principle, the MSE is minimized when the expectation $E[\{s(n) - y(n)\}x(m)]$ equals zero.



In the frequency domain, the optimal filter $W(\omega)$ is given by $W(\omega) = \frac{P_{sx}(\omega)}{P_x(\omega)}$. It is computed based on the assumptions that the signal and noise are independent and the power spectral densities of the signal $P_s(\omega)$, noise $P_v(\omega)$, and the cross power spectral density $P_{sx}(\omega)$ are known.

Ultimately, the Wiener filter $W(\omega)$ is designed to adjust its frequency response according to the ratio of signal to noise power, acting as a spectral attenuator or mask: $W(\omega) = \frac{\hat{P}_s(\omega)}{\hat{P}_s(\omega) + \hat{P}_v(\omega)}$. This means that for each frequency, if the power of the noise is high we will apply a stronger filter (attenuating the power), if the power of the signal is very high we will let the total signal pass as it is.

7 FEATURES AND CLASSIFICATION

Nowadays, it's common to work with signal processing associated with artificial intelligence and machine learning. They usually are employed for music genre recognition and in general we would like to associate a label or a class to

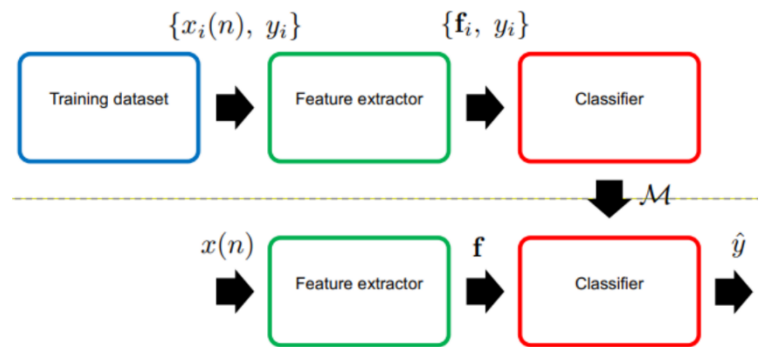
an audio track when using these systems. When labeling and comparing two different tracks with a reference one to find similarities, we can extract some features of each track, like loudness (using average signal amplitude) or fundamental frequency (using LPC). So, from a signal of thousands of samples we are extracting a few numbers that are strongly correlated to the signal. We can then plot, in an N-dimensional space, all these parameters to have some sort of coordinate, which means that signals that have similar characteristics will have similar coordinates. Sometimes, it may seem very obvious if one track belongs to one group, but other times we may need additional information to decide.

7.1 SOUND CLASSIFICATION

A common pipeline that we can use is by starting from a training dataset of known data, followed by a feature extraction block, which takes the training dataset, and computes a N-element feature vector. Finally, we will map these feature vectors into labels or classifiers.

After the training process, we will now consider a track which was not used for the training process, extract the features, compare it with the model and apply an estimated label (based on the area the feature vector is inside).

Some important points to consider is that we need to decide the way we separate each group (line or curve), and the importance or weight we give to each feature.



7.2 AUDIO FEATURES

This section lists the most common audio features:

- **Basic Descriptors:** Derived from audio statistics.
- **Harmonicity Descriptors:** Related to the harmonic components of a track.
- **Timbral Descriptors:** Capture timbral information.
- **Other Descriptors:** Vary based on application needs.

7.2.1 Basic Descriptors

Here below are listed various basic audio descriptors:

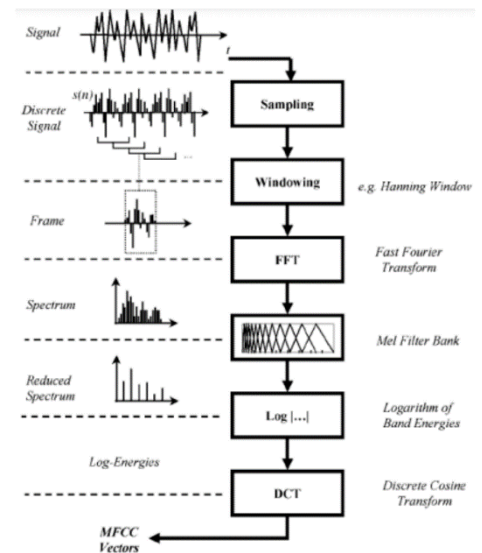
- **Audio WaveForm (AWF):** It's calculated using small frames and calculating the maximum and minimum value of each frame to extract the waveform of the track. So, with each frame we will extract two values, min and max.
- **Audio Power (AP):** Represents the power of each frame, calculated using the formula $AP(l) = \frac{1}{N_{hop}} * \sum_{t=0}^{N_{hop}-1} |s(t + l * N_{hop})|^2$, $0 < l < L - 1$ (non-overlapping frames).
- **Audio Spectral Envelope (ASE):** A reduced version of the spectrogram generated by summing the energy of the original power spectrum within a series of frequency bands.
- **Audio Spectrum Centroid:** Averages frequencies based on the energy to find the center of gravity of the power spectrum. Related to this is the Audio Spectrum Spread.

7.2.2 Harmonicity Descriptors

About audio harmonicity we have the Harmonic Ratio, that measures the proportion of the harmonic components in the power spectrum. For each frame we will calculate the normalized autocorrelation function of the signal and consider the maximum value.

7.2.3 Timbral Descriptors

- **Log Attack Time:** About the timbre we can extract the envelope using the ADSR model, and calculate the Log Attack Time, which is the time the sound envelope takes to reach the maximum amplitude of a signal from a minimum threshold time.
- **MFCC (Mel Frequency Cepstrum Coefficients):** Widely used in speech recognition, it applies a non-linear Mel-frequency scale that approximates the auditory system's response. To convert a frequency in hertz into its equivalent in Mel value, the following formula is used: $f_{mel} = 1127.0148 * \log \left(1 + \frac{f_{HZ}}{700} \right)$. The MFCCs are based on the extraction of the signal energy within critical frequency bands by means of a series of triangular filters whose center frequencies are equally spaced according to the Mel scale. The log-energy of the spectrum is measured within the pass-band of each filter, resulting in a reduced representation of the spectrum. The cepstral coefficients are finally obtained through a Discrete Cosine Transform (DCT) of the reduced log-energy spectrum.



7.2.4 Other Descriptors

One of the main descriptors that are used generally is the zero-crossing rate, to see how many times we have a change of sign of the signal: $ZCR(l) = \frac{F_s}{2N_\omega} * \sum_{t=1}^{N_\omega-1} |sign[s_l(t)] - sign[s_l(t-1)]|$.

7.3 CLASSIFICATION METHODS

With these descriptors we can define some classification methods.

- **KNN (K Nearest Neighbor):** is the most basic classifier, because we consider the k nearest points and determine the number of elements of each class within that region, the majority wins. Depending on the number k of nearest elements we can have different results, usually a small k is more flexible.
- **Linear Classifier:** Provides a simple linear separator that decides which region belongs to each class. However, we need a training process to find the best linear equation to separate the groups.
- **SVM (Support Vector Machine):** It considers the concept of margin between classes, where, for example, points should ideally lie far away from the line. The one that maximizes the margin separating the classes is selected. Usually, if there are more than two classes, we could use a multi-class solution, where we combine binary classifiers (i.e., predicting only two classes) trained multiple times on different combinations of the classes.

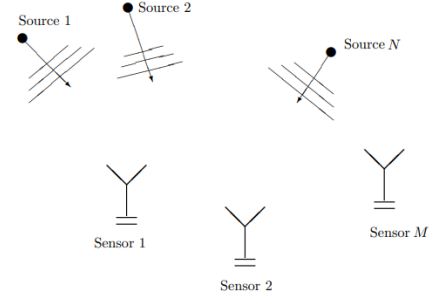
8 MICROPHONE ARRAYS: SPATIAL METHODS FOR DOA ESTIMATION (NARROW BAND MODELS)

We want to detect and locate multiple (N) radiating acoustic sources using an array of multiple (M) passive microphones.

We could achieve this determining the acoustic energy distribution over space, which likely involves spatial spectral analysis techniques.

Consider the scenario in the figure on the right. There are:

- **N sources of sound**, indicated as Source 1, Source 2, ..., Source N.
- **M sensors** (microphones), indicated as Sensor 1, Sensor 2, ..., Sensor M.



Each source emits sound waves represented by lines emanating outwards, and these waves are captured by the microphone array.

To make the problem tractable, several simplifying assumptions are made:

- **Sources located in the far-field:** This means that the sources of sound are far enough away from the microphone array such that the sound waves arrive as planar wavefronts, which simplifies the math involved in DOA estimation.
- **Only directions of arrival (DOAs) characterize the sources:** It is assumed that the sources can be characterized solely by the direction from which their sound arrives at the microphone array, without needing other information such as the distance to the source.
- **Sources and microphones all lie on the same plane:** This implies a 2D geometry for the problem, which significantly simplifies the calculations as compared to a 3D scenario.
- **No reverberation:** Reverberation occurs when sound reflects off surfaces before reaching the microphones, complicating DOA estimation. Assuming no reverberation means that the sound waves reach the microphones without any reflections.
- **No dispersion:** Dispersion would cause different frequencies of sound to travel at different speeds, changing the wave's shape over distance. Without dispersion, the shape of the sound wave does not change, simplifying the model.
- **Homogeneous propagation:** This assumes that the sound waves propagate through a uniform medium without any variations that could affect the speed of sound or cause refraction.

8.1 DATA MODEL

Let's simplify for a moment the problem considering having a single source. The angle at which this sound source impinges on the sensor array is referred to as the Direction of Arrival (DOA), denoted by the Greek letter theta (θ).

8.1.1 Single Source & Single Sensor

We can model the output of one of the sensors with the following equation:

$$y_{k(n)} = h_{k(n)} * s(n - \tau_k) + \varepsilon_{k(n)} \text{ where } \begin{cases} s(n) = \text{source signal in absence of noise or other distortions.} \\ h_{k(n)} = \text{impulse response of the } k_{\text{th}} \text{ sensor.} \\ \tau_k = \text{propagation delay between the reference point and the } k_{\text{th}} \text{ sensor.} \\ \varepsilon_{k(n)} = \text{additive noise at the } k_{\text{th}} \text{ sensor.} \end{cases}$$

The information about the source's position θ (Problem of Direction of Arrival DOA Estimation) is encoded in the propagation delays $\tau_1, \tau_2, \dots, \tau_M$ for each sensor. By determining these delays, one can infer the direction the sound came from. This means that we must estimate the time delays $\tau_1, \tau_2, \dots, \tau_M$ given that the impulse responses $h_{k(n)}$ of the sensors are known.

If we consider the signal emitted by the single source as narrowband, we can further simplify the equation considering the signal as a cosine wave with a known carrier frequency:

$$\bar{s}(n) = a(n) * \cos[\omega_c * n + \phi(n)] \text{ where } \begin{cases} \bar{s}(n) = \text{source signal.} \\ a(n) = \text{time varying amplitude.} \\ \omega_c = \text{carrier frequency.} \\ \phi(n) = \text{time varying phase.} \end{cases}$$

Frames of length L are extracted from the signal using a rectangular window function, which isolates segments of the signal without overlap:

$$S_{t(n)} = s(n) * w(n - t) \text{ where } \begin{cases} w(n) = 1 \text{ for } 0 < n < L - 1 \\ w(n) = 0 \text{ otherwise.} \end{cases}$$

And for each frame we compute the DTFT:

$$S_t(\omega) = \sum_{n=-\infty}^{\infty} s_t(n) e^{-j\omega n}$$

Practically we are performing STFT.

We can simplify the Short-Time Fourier Transform (STFT) under the assumption that the amplitude and phase vary slowly over the window ($\alpha(t) \approx \alpha(t + n)$ and $\phi(t) \approx \phi(t + n)$), effectively treating them as constant. This allows the STFT of the windowed signal to be expressed as a product of a complex exponential and the nearly constant amplitude and phase:

$$S_t(\omega_c) = \alpha(t) e^{j\phi(t)}$$

We can further simplify the model by focusing on the signal at the carrier frequency. The signal can be represented solely (*solamente*) as a function of time due to the narrowband assumption. This results in a notation where $s(t)$ denotes the STFT at ω_c , and is equal to the amplitude times a complex exponential of the phase:

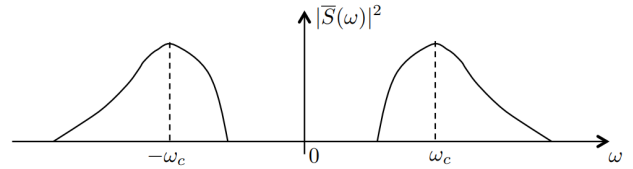
$$s(t) \cong S_t(\omega_c) = \alpha(t) e^{j\phi(t)}$$

The narrowband signal can be demodulated to baseband by multiplying the original signal with a complex exponential of the negative carrier frequency. This shifts the frequency spectrum down to baseband (centered at 0 Hz). After demodulation, low-pass filtering is applied to remove the high-frequency component, leaving the baseband signal represented as an amplitude-modulated and phase-modulated component at DC (0 Hz):

band-pass signal (modulated)

$$\bar{s}(n) = \alpha(n) \cos[\omega_c n + \phi(n)]$$

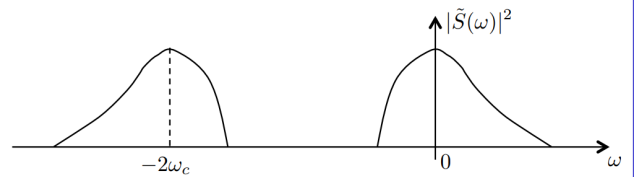
$$= \frac{1}{2} \alpha(n) \left\{ e^{j[\omega_c n + \phi(n)]} + e^{-j[\omega_c n + \phi(n)]} \right\}$$



de-modulated signal

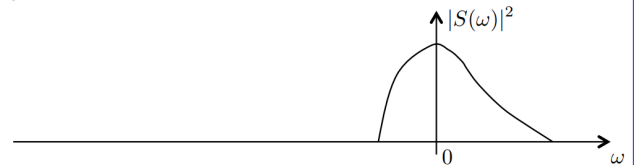
$$\tilde{s}(n) = \bar{s}(n) \times 2e^{-j\omega_c n}$$

$$= \alpha(n) \left\{ e^{j\phi(n)} + e^{-j[2\omega_c + \phi(n)]} \right\}$$



base-band signal (de-modulated and low-pass filtered)

$$s(n) = \alpha(n) e^{j\phi(n)}$$



8.1.2 Single Source Multiple Sensors (Array)

In an array of sensors, the output of the k_{th} sensor in the array is given by the STFT of the microphone signals, where each sensor output, $\backslash(y_k(t) \backslash)$, is represented as:

$$y_k(t) = H_k(\omega_c)s(t)e^{-j\omega_c\tau_k} + e_k(t) \text{ where } \begin{cases} H_k(\omega_c) = \text{frequency response of the sensor } k. \\ s(t) = \text{source signal.} \\ \tau_k = \text{time delay of signal "impact" on the sensor.} \\ e_k(t) = \text{additive noise.} \end{cases}$$

Is useful to combine the signals from the array into a vector representation:

$$y(t) = a(\theta)s(t) + e(t) \text{ where } \begin{cases} y(t) = \text{array vector} = [y^1(t); y^2(t); \dots; y_m(t)]^T \\ a(\theta) = \text{propagation vector} = [H_1(\omega_c)e^{-j\omega_c\tau^1}; H_2(\omega_c)e^{-j\omega_c\tau^2}; \dots; H_M(\omega_c)e^{-j\omega_c\tau_M}]^T \\ e(t) = \text{noise vector} = [e_1(t); e_2(t); \dots; e_M(t)]^T \end{cases}$$

8.1.3 Multiple Sources Multiple Sensors

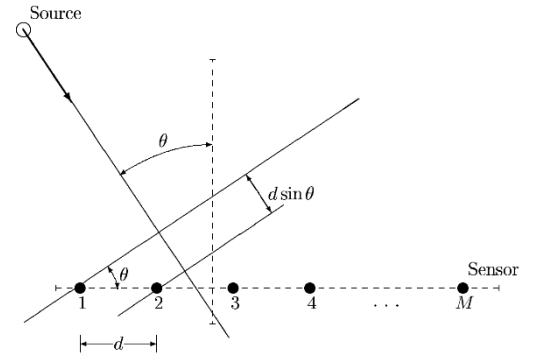
Consider now multiple signals $s_1(t), s_2(t), \dots, s_N(t)$ coming from directions $\theta_1, \theta_2, \dots, \theta_N$ with corresponding propagation vectors $a(\theta_1), a(\theta_2), \dots, a(\theta_N)$. Assuming ideal linear acoustic propagation, the signals from all sources add linearly. This means that the total signal $y(t)$ at the array is the sum of the signals from all sources plus noise $e(t)$:

$$y(t) = As(t) + e(t) \text{ where } \begin{cases} A = [a(\theta^1), a(\theta^2), \dots, a(\theta_n)] \text{ is the } M \times N \text{ propagation matrix.} \\ s(t) = [s_1(t); s_2(t); \dots; s_n(t)]^T \text{ is the source vector.} \end{cases}$$

8.2 UNIFORM LINEAR ARRAY (ULA)

A Uniform Linear Array (ULA) is a configuration of multiple (M) sensors, typically microphones, that are arranged in a straight line with equal spacing (d) between them. It is used to capture and process sound signals for various applications like direction finding, beamforming, and spatial filtering.

Considering a single source, the sound wave arrives at each sensor in the ULA with a certain time delay relative to the first sensor, which is chosen as the reference point. The angle of arrival of the sound wave from the source to the array is denoted by θ . Due to the geometry of the ULA and the angle θ , there is a path difference between the sound reaching each sensor. This path difference translates into a time delay, τ_k , for the $h_{k_{th}}$ sensor in the array and we can calculate the time delay for each sensor using the formula: $\tau_k = (k - 1) * \frac{d \sin(\theta)}{c}$ where c is the speed of sound in the medium (approximately 340 m/s in air).



If we assume that:

- All microphones are identical and have a flat frequency response across the band of interest, so we set $H_k(\omega_c) = 1$ for each sensor.
- The direction of arrival (DOA) θ is within the range $[-90^\circ, 90^\circ]$.

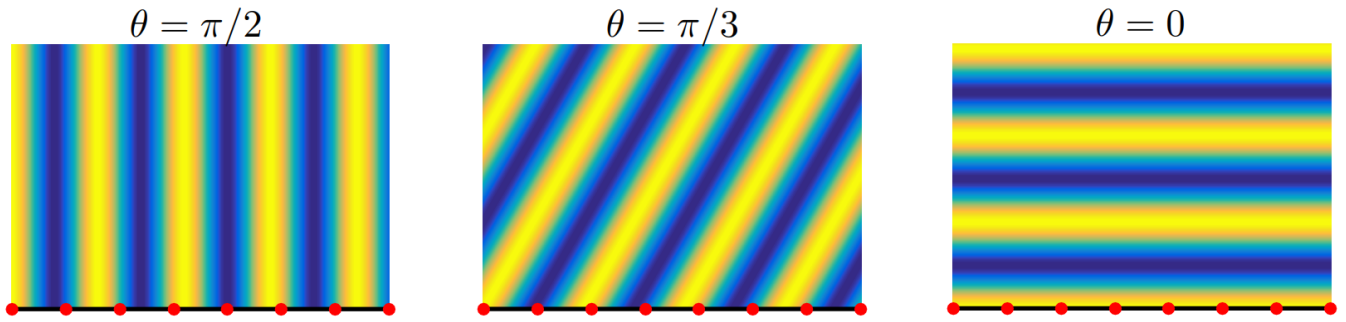
we can simplify the propagation vector $a(\theta)$ (which represents the relative phases that a plane wavefront from the source induces across the sensors of the ULA) in the following way:

$$\begin{bmatrix} 1 \\ e^{-j\omega_c \frac{d \sin \theta}{c}} \\ \vdots \\ e^{-j(M-1)\omega_c \frac{d \sin \theta}{c}} \end{bmatrix}$$

The exponent term $-j\omega_c \frac{d \sin \theta}{c}$ corresponds to the phase shift due to the time delay in reaching the k_{th} microphone. The phase shift for the first microphone (the reference point) is zero, hence its vector element is 1.

8.2.1 Spatial Frequency

The frequency of the complex sinusoids in the propagation vector is referred to as **spatial frequency**, denoted by ω_s . It is calculated as $\omega_s = \frac{\omega_c d \sin \theta}{c}$, linking the concept of spatial frequency to the angle of arrival θ .



8.2.2 Anti-aliasing Condition in ULA

The propagation vector is sampled at a unitary sample period, resulting in a vector $a(\theta)$ with terms representing different phases based on the spatial frequency. To ensure no aliasing occurs, the absolute value of the spatial frequency must be less than or equal to π , which is a standard sampling theorem condition. The condition translates into a requirement for microphone spacing d : for no aliasing, the spacing d must be less than or equal to half the wavelength λ of the sound:

$$d \leq \frac{\lambda}{2}$$

This prevents two different DOAs from producing the same phase differences, which could lead to incorrect DOA estimation.

8.2.3 DOA Estimation

The DOA of a sound source is related to the spatial frequency sampled by the ULA. DOA estimation can be approached as a spatial frequency estimation problem. Multiple sources with different DOAs will have different spatial frequencies.

In the next paragraph, we see different methods to do that.

Estimate DOA without assuming any covariance structure of the array data.

8.2.4 Spatial filtering (Non-parametric methods)

Spatial filtering is a technique used to manipulate signals based on the direction from which they arrive at a sensor array. It is analogous to temporal filtering, but instead of one-dimensional time, it operates across spatial dimensions. The goal is to enhance signals coming from a specific direction, called the direction of arrival (DOA), while attenuating signals from other directions.

The process involves linearly combining microphone signals using a set of weights, which are represented as a vector $h = [h_1, h_2, \dots, h_M]^H$ (where H denotes the Hermitian transpose, and M is the number of microphones in the array). The combined signal, $y_F(t)$, is calculated as:

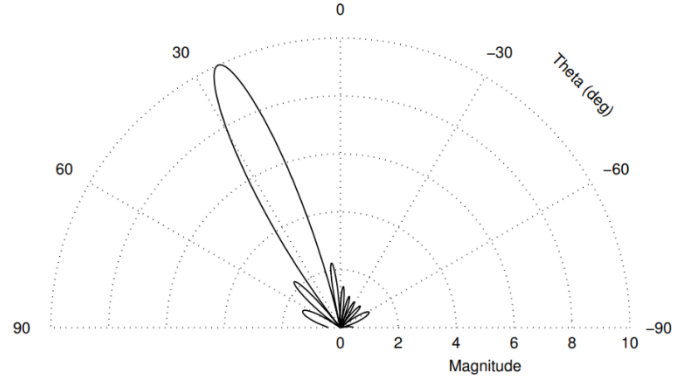
$$y_n(t) = \sum_{k=1}^M h_k \cdot y_k(t) = h^H y(t)$$

Assuming that the filter h is band-pass, centered at θ in the spatial domain, the power of the filtered signal should give a good indication of the energy coming from the direction θ . The power of the filtered signal is:

$$E|y_n(t)|^2 = h^H * R * h \text{ where } R = y(t)y(t)^H \text{ covariance matrix of the array data}$$

8.2.4.1 Delay-and-Sum Beamformer

Beamforming (*formazione di fasci*) is a specific type of spatial filtering where the weights are designed such that the spatial response of the filter is unitary at the target DOA (θ). This means the filter passes the signal from the target direction without distortion. This is typically done using the delay-and-sum technique, where the weights correspond to time delays that align the signals from the target direction to



constructively interfere (band-pass characteristic), while signals from other directions are misaligned, leading to destructive interference (minimize the power from directions other than θ).

The power of the output signal from the beamformer provides an indication of the energy coming from the direction θ . The DOAs of the sources can be estimated by evaluating the pseudo-spectrum function $p(\theta)$ across all spatial directions, identifying the peaks corresponding to different source locations.

The ability of the beamformer to distinguish between two closely spaced sources (resolution) depends on the width of the main lobe of the spatial response. For a uniform linear array, this width is approximately given by:

$$\frac{\lambda}{l * |\cos(\theta)|} \text{ where } \begin{cases} l \text{ is the array aperture.} \\ \lambda \text{ is the wavelength of the signal.} \end{cases}$$

Given two sources with angular separation of $\Delta\theta$ (located around $\theta \cong 0$) they can be solved if $\Delta\theta > \frac{\lambda}{l}$.

8.2.4.2 Capon Method

The Capon method, also known as the Minimum Variance Distortionless Response (MVDR) method, is a spatial filtering technique used in signal processing.

The objective of the Capon method is to minimize the output power of the array while maintaining a distortionless response in the direction of interest (θ). This is mathematically formulated as a minimization problem:

$$h(\theta) = \arg \min_h h^H * R * h \text{ where } \begin{cases} h^H * a(\theta) = 1. \\ h = \text{weight vector (spatial filter).} \\ R = \text{spatial covariance matrix.} \\ a(\theta) = \text{steering vector.} \\ H = \text{conjugate transpose operation.} \end{cases}$$

Unlike conventional beamformers that assume R to be the identity matrix I , the Capon method focuses on minimizing the power of signals that are actually impinging on the array from directions different from the target. This leads to better interference rejection.

The solution to the optimization problem is given by $h(\theta) = \frac{R^{-1}a(\bar{\theta})}{a^H(\bar{\theta})R^{-1}a(\bar{\theta})}$, which is the expression for calculating the filter weights.

The output power of the filter is given by the expectation:

$$E\{|y_F(t)|^2\} = h^H(\theta) * R * h(\theta) = \frac{1}{a^H(\theta)\hat{R}^{-1}a(\theta)} = p(\theta) \text{ for } \theta = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$$

This expression also represents the Capon pseudo-spectrum $p(\theta)$.

8.2.4.2.1 DOA Estimation

Direction of arrivals (DOAs) are estimated by finding the N highest peaks of the Capon pseudo-spectrum $p(\theta)$.

8.2.4.2.2 Data Dependence and Performances

The performance of the Capon method is stated to be slightly superior to that of conventional beamforming techniques. The superiority comes from the method's ability to more effectively suppress interference and noise, leading to a cleaner output and more accurate DOA estimation.

On the other hand, we must take in account that in Capon method the filter weights, the pseud-spectrum and the spatial response are dependent on the data covariance matrix R , which is estimated from the array data. This implies that the resolution of the Capon method is not constant and depends on the data and the performance may vary with different signal scenarios.

8.2.5 Parametric Methods

Parametric methods are a powerful alternative to spatial filtering for DOA estimation. Unlike non-parametric methods, parametric methods assume a certain model about the data which can lead to more accurate results if the model fits well to the actual data.

Parametric methods utilize a model of the covariance structure of the array data. The covariance matrix represents the variance within the signals and the correlation between different signals received by an array of sensors.

For applying this technique it's necessary to have:

- **Known Number of Sources:** The number of sources N must be known exactly in advance, and it must be less than M (the number of sensors in the array).
- **Non-Singular Covariance Matrix:** The signals $s_1(t), s_2(t), \dots, s_N(t)$ are such that their covariance matrix is non-singular (invertible). This means each signal contributes unique information. The matrix R_s is the covariance matrix, calculated as:

$$R_s = E\{s(t)s^H(t)\}$$

- **Spatially White Sensor Noise:** The noise $e(t)$ affecting the sensors is spatially white, meaning it is the same across all sensors and uncorrelated over time. Its expected value is an identity matrix I_M multiplied by the variance σ^2 , indicating identical and independent distribution (i.i.d.) of the noise at all sensors:

$$E\{e(t)e^H(t)\} = \sigma^2 I_M$$

- **Uncorrelated Noise and Signals:** The sensor noise is uncorrelated with the source signals, which means that the noise does not carry any information about the source signals and thus does not distort the covariance structure of the signals.
- **Unique DOAs:** Each Direction of Arrival is unique, resulting in different spatial frequencies $\omega_{s1}, \omega_{s2}, \dots, \omega_{sN}$. This implies that each source has a distinct direction relative to the sensor array, and no two sources come from the same direction.

8.2.5.1 Covariance Model

Under the assumptions specified in the previous paragraph, the input covariance matrix R can be expressed as the sum of two components:

$$R = A * R_s * A^H + \sigma^2 * I_M \text{ where } \begin{cases} A \text{ is the array manifold matrix.} \\ R_s \text{ is the signal covariance matrix.} \\ \sigma^2 \text{ is the noise variance.} \\ I_M \text{ is the identity matrix of size } M \end{cases}$$

The covariance matrix R can be diagonalized using eigenvalue decomposition:

$$\mathbf{R} = [\mathbf{U} \quad \mathbf{V}] \underbrace{\begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \ddots & & \\ & & \lambda_N & \\ \vdots & & & \ddots \\ & & \lambda_{N+1} & \\ & & & \ddots & 0 \\ 0 & \dots & 0 & 0 & \lambda_M \end{bmatrix}}_{\Lambda} \begin{bmatrix} \mathbf{U}^H \\ \mathbf{V}^H \end{bmatrix}$$

where: $\begin{cases} V \text{ contains the orthonormal eigenvectors associated with the noise.} \\ U \text{ contains the orthonormal eigenvectors associated with the source signals.} \end{cases}$

Moreover, as the source signals and the noise are uncorrelated, we can rewrite the diagonal eigenvalue matrix Λ as:

$$\Lambda = \underbrace{\Lambda_s}_{\text{eigenvalues of } \mathbf{A}\mathbf{R}_s\mathbf{A}^H} + \underbrace{\Lambda_e}_{\text{eigenvalues of } \sigma^2\mathbf{I}_M} = \Lambda_s + \sigma^2\mathbf{I}_M$$

In this equation, Λ_s is a diagonal matrix with the N largest eigenvalues, which are the eigenvalues of $A * R_s * A^H$. Λ_e contains the remaining $M - N$ eigenvalues, which are all equal to σ^2 , the noise power.

The eigenvalues are sorted in decreasing order $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_M$.

As the spatial frequencies $\omega_{s_1}, \omega_{s_2}, \dots, \omega_{s_N}$ are distinct for each source signal, the matrix A is full rank, and therefore, $A * R_s * A^H$ has N non-zero eigenvalues and $M - N$ null eigenvalues.

Finally, the eigenvalues of R can be expressed as:

$$\lambda_k = \begin{cases} \bar{\lambda} + \sigma^2 & \text{if } 1 \leq k \leq N \\ \sigma^2 & \text{if } N + 1 \leq k \leq M \end{cases}$$

This indicates that the first N eigenvalues correspond to the signal sources (augmented by the noise power), and the remaining eigenvalues correspond to the noise power.

8.2.5.1.1 Properties of V and U

Properties of V : the matrix V contains the eigenvectors associated with the noise subspace. It can be derived from the eigenvalue decomposition of R , focusing on the noise-related eigenvalues:

1. The eigenvalue decomposition shows that: $RV = V \begin{bmatrix} \lambda_{N+1} & & \\ & \ddots & \\ & & \lambda_M \end{bmatrix} = \begin{bmatrix} \sigma^2 & & \\ & \ddots & \\ & & \sigma^2 \end{bmatrix} = \sigma^2 * V$.
2. When we replace R with its expression $A * R_s * A^H + \sigma^2 * I_M$, we get $A * R_s * A^H * V + \sigma^2 * V = \sigma^2 V$.
3. This simplifies to $A * R_s * A^H * V = 0$ which implies that the noise subspace is orthogonal to the signal subspace represented by $A * R_s * A^H$.
4. It directly follows that: $A^H * V = 0_{N \times (M-N)}$ indicating that the eigenvectors in V are orthogonal to the steering matrix A , reinforcing that V represents the noise-only part of the data.

Properties of U : the matrix U contains the eigenvectors associated with the signal subspace:

1. The eigenvalue decomposition implies:

$$RU = U \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_N \end{bmatrix} = U * \begin{bmatrix} \bar{\lambda}_1 & & \\ & \ddots & \\ & & \bar{\lambda}_N \end{bmatrix} + U * \begin{bmatrix} \sigma^2 & & \\ & \ddots & \\ & & \sigma^2 \end{bmatrix} = U * \bar{\lambda} - \sigma^2 * U$$

2. Replacing R with $A * R_s * A^H + \sigma^2 * I_M$ gives: $A * R_s * A^H * U + \sigma^2 * U = U * \Lambda + \sigma^2 * U$
3. Simplifying the above expression yields: $A * R_s * A^H * U = U * \bar{\Lambda}$, which indicates that the eigenvectors in U span the space that is influenced by the signal-related covariance structure.
4. Noting that Λ is non-singular (invertible), we can write: $U = A * R_s * A^H * U * \Lambda^{-1}$. Here Λ^{-1} is the inverse of the non-singular diagonal matrix of the largest N eigenvalues, meaning that U can be expressed in terms of the signal covariance structure: $U = A * C$, where C is a non-singular matrix.

8.2.5.2 Summary of the Algorithm

1. Start with the received signal's covariance matrix R .
2. Perform eigenvalue decomposition on R to obtain the eigenvalues and eigenvectors.
3. Sort the eigenvalues in descending order.

4. Identify U as the matrix of eigenvectors associated with the N largest eigenvalues (signals), and V as the matrix of eigenvectors associated with the $M - N$ smallest eigenvalues (noise).
5. The DOA estimation can be refined using the information in U while V can be used for noise estimation and suppression.

8.2.6 MUSIC – Multiple Signal Classification (Parametric method)

The MUSIC (Multiple Signal Classification) algorithm is an approach used in signal processing to estimate the direction of arrival (DOA) of multiple signals.

MUSIC is based on the orthogonality principle, which states that the signal subspace is orthogonal to the noise subspace. This is mathematically represented as:

$$A^H * V = 0_{N \times (M-N)} \text{ where } \begin{cases} A^H \text{ is the conjugate transpose (Hermitian transpose) of the signal matrix.} \\ A \text{ and } V \text{ represents the noise eigenvectors.} \end{cases}$$

This orthogonality can also be expressed in a scalar form:

$$A^H * V * V^H * A = 0$$

Here, $V * V^H$ forms a projection matrix onto the noise subspace, and this expression means that when the signal matrix is projected onto the noise subspace, it yields zero.

The solution to the MUSIC algorithm is given by the propagation vectors, which correspond to the directions of arrival (DOAs) of the signals. This is expressed as:

$$a^H(\theta_k) * V * V^H * a(\theta_k) = 0 \text{ where } a(\theta_k) = \text{steering vector corresponding to the } k_{th} \text{ source direction } \theta_k.$$

8.2.6.1 MUSIC Algorithm Steps

The algorithmic steps for MUSIC are:

- **Step 1: Covariance Matrix Estimation** → First, estimate the covariance matrix of the received signal. This is given by: $\hat{R} = \frac{1}{K} \sum_{t=1}^K y(t)y^H(t)$ where $y(t)$ represents the received signal at time t , K is the number of time samples, and $y^H(t)$ is the conjugate transpose of $y(t)$.
- **Step 2: Eigenvalue Decomposition** → Using eigenvalue decomposition on the estimated covariance matrix ($\hat{R}$), you obtain the eigenvectors that correspond to the noise subspace, denoted ($\hat{V}$).
- **Step 3: DOA Estimation** → Finally, estimate the DOAs by finding the N highest peaks of the MUSIC spectrum function: $p(\theta) = \frac{1}{a^H(\theta)\hat{V}\hat{V}^H a(\theta)}$. The search for the DOAs is done within the range of $\theta \in [-\pi/2, \pi/2]$.

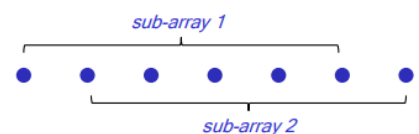
The MUSIC spectrum $p(\theta)$ will have peaks at the angles where the incoming signals are coming from, due to the orthogonality between signal and noise subspaces. The N highest peaks correspond to the estimated DOAs of the N signals.

8.2.7 ESPRIT (Parametric method)

The ESPRIT (Estimation of Signal Parameters via Rotational Invariance Techniques) algorithm is a parameter estimation method used in signal processing to determine the directions of arrival (DOA) of multiple signals. The method takes advantage of the invariance properties of signal subspaces under rotational transformations.

ESPRIT ALGORITHM:

- **Step 1: Signal Subspace Decomposition:**
 1. Start with a sensor array and divide it into two identical sub-arrays with M and M' elements each, such that $M' = M - 1$.
 2. The array data model $y(t) = As(t) + e(t)$ represents the sensor outputs, where $y(t)$ are the measurements, A is the matrix related to the sources, $s(t)$ is the signal, and $e(t)$ is the noise at time t .



3. Derive subarray models $y_1(t)$ and $y_2(t)$ such that $y_1(t) = A_1 s(t) + e_1(t)$ and $y_2(t) = A_2 s(t) + e_2(t)$, with A_1 and A_2 representing the relationship of each subarray to the sources.
- **Step 2: Estimate the Covariance Matrix:**
 1. Compute the sample covariance matrix $\hat{R}$ from the sensor array outputs. This involves averaging the product of measurements $y(t)$ and their conjugate transpose $y^H(t)$ over K time samples.
 - **Step 3: Estimate Subspace Matrices U_1 and U_2 :**
 1. For each subarray, compute their covariance matrices R_1 and R_2 .
 2. Perform eigenvalue decomposition on these matrices to obtain eigenvectors associated with the signal sources, denoted as U_1 and U_2 .
 - **Step 4: Rotational Invariance Property:**
 1. Utilize the fact that the subarray matrices A_1 and A_2 are related by a rotational invariance property, such that $A_2 = A_1 D$, where D is a diagonal matrix containing exponential terms related to the DOAs.
 2. Understand that $U = AC$, where C is an invertible matrix, leading to $A_1 = U_1 C^{-1}$ and $A_2 = U_2 C^{-1}$.
 - **Step 5: Solve for Matrix D :**
 1. Knowing $A_2 = A_1 D$, solve for $U_2 = U_1 C^{-1} D C = U_1 \Phi$ where $\Phi = C^{-1} D C$.
 2. Note that Φ has the same eigenvalues as D , which are directly related to the DOAs.
 - **Step 6: Estimate Spatial Frequencies:**
 1. Compute the least squares estimate of Φ , denoted $\hat{\Phi} = (U_1^H U_1)^{-1} U_1^H U_2$.
 2. Calculate the eigenvalues of $\hat{\Phi}$, which correspond to the spatial frequencies.
 - **Step 7: Estimate the DOAs:**
 1. Estimate the spatial frequencies as $\omega_k = -\arg(v_k)$, where v_k are the eigenvalues of $\hat{\Phi}$, and $\arg(\cdot)$ represents the argument (or phase) of the eigenvalue.
 2. Finally, estimate the DOAs of the signals from the spatial frequencies.

9 MICROPHONE ARRAYS: LOCALIZATION OF WIDE-BAND SOURCES

In this section we will consider a specific scenario where the following conditions are present:

- There is a single acoustic source to be located.
- The environment has small to moderate reverberation effects.
- The source emits a wide-band signal.

In such a scenario, traditional spatial filtering or parametric methods might not be the most efficient solution for determining the location of the sound source.

9.1 DATA MODEL

When dealing with wide-band sources in the field of audio signal processing, the sinusoidal model commonly used for narrow-band signals is not adequate. Wide-band sources, like many natural sounds, contain energy across a wide range of frequencies, making the simplistic sinusoidal assumption insufficient for accurate modelling.

To simplify the explanation without sacrificing generality, a system with just two microphones is considered. The model for each microphone in the context of a wide-band source takes into account three main components:

- $s(n)$ represents the source signal. This is the original sound that is being captured by the microphones.
- $g_k(n)$ denotes the channel response from the source to the k_{th} microphone. This includes the effects of propagation through the space, such as distance-related attenuation and reverberation.
- $e_k(n)$ is the additive noise at the k_{th} microphone. Noise is an inevitable part of any real-world signal acquisition and can come from a multitude of sources, such as electrical noise in the microphone circuitry or ambient noise in the environment.

The microphone signal $y_k(n)$ is then the combination of the source signal after it has been modified by the channel response, plus any noise that has been added. Mathematically, for each microphone indexed by k , this relationship is given by the convolution of the source signal with the channel response, plus noise:

$$y_k(n) = g_k(n) * s(n) + e_k(n)$$

Here, $*$ denotes the convolution operation.

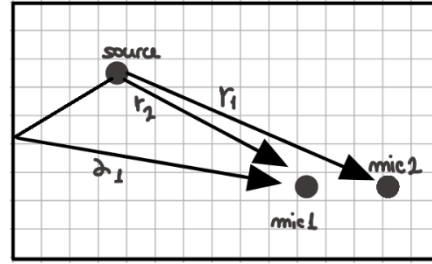
9.1.1 Time Difference of Arrival

In acoustic signal processing, Time Difference of Arrival (TDOA) involves determining the location of a sound source often measuring the time at which sounds arrive at different microphones. The TDOA approach is particularly relevant when the source emits signals that span a wide frequency range (wide-band signals).

The impulse responses of the channels leading to two microphones can be modeled to include both the direct path signal and the early reflections from the environment. These channel models can be expressed as follows:

$$g_1(n) = \underbrace{\frac{1}{r_1}\delta(n - \tau_1)}_{\text{direct signal}} + \underbrace{\sum_i \alpha_i \delta(n - \tau_{1i})}_{\text{early reflections}}$$

$$g_2(n) = \underbrace{\frac{1}{r_2}\delta(n - \tau_2)}_{\text{direct signal}} + \underbrace{\sum_i \beta_i \delta(n - \tau_{2i})}_{\text{early reflections}}$$



For the first microphone, the impulse response $g_1(n)$ is a function representing the direct signal arriving after a time delay τ_1 , attenuated by the factor $\frac{1}{r_1}$, plus a sum of early reflections. Each reflection arrives after a different delay τ_{1i} and is attenuated by a factor α_i .

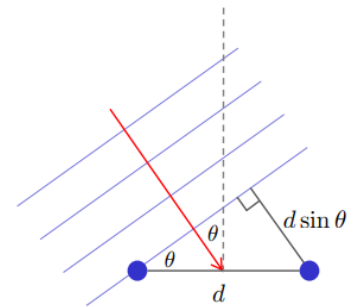
Similarly, for the second microphone, the impulse response $g_2(n)$ includes the direct signal with a time delay τ_2 and attenuation factor $\frac{1}{r_2}$, as well as the sum of early reflections with their respective delays τ_{2i} and attenuation factors β_i .

The critical information used to locate the source is encoded in the propagation delays τ_1 and τ_2 , which are the times taken for the sound to travel from the source to the first and second microphones, respectively.

Since the exact moment when the source emitted the signal is typically unknown, the focus is on estimating the TDOA, represented as $\tau_{12} = \tau_2 - \tau_1$. This is the difference in time between the arrivals of the sound at microphone 2 relative to microphone 1. By calculating the TDOA, one can infer the relative position of the sound source concerning the microphone array without needing to know the precise emission time of the sound.

9.1.2 From TDOA to DOA

In the context of sound source localization, this concept involves calculating the Time Difference of Arrival (TDOA) between two microphones to determine the Direction of Arrival (DOA) of the sound. If we assume that the sound source is sufficiently far from the microphone pair (far-field condition), the sound waves arrive at each microphone with a time delay that is a function of the distance between the microphones and the angle of arrival (θ) of the sound wave.



The TDOA, denoted as τ_{12} , can be described by the equation:

$$\tau_{12} = \frac{d \sin(\theta)}{c}, \text{ where } \begin{cases} d \text{ is the distance between the two microphones} \\ \sin(\theta) \text{ represents the sine of the angle of arrival} \\ c \text{ is the speed of sound} \end{cases}$$

This equation captures the relationship between the spatial arrangement of the microphones, the angle of the sound's arrival, and the observed time delay.

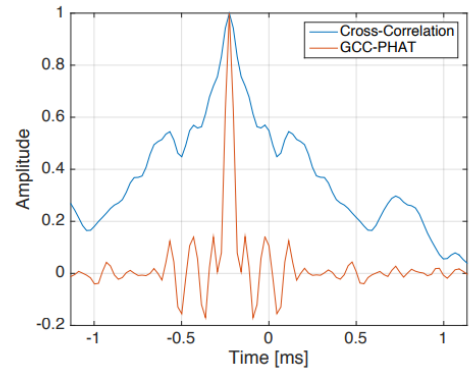
Once the TDOA is known or estimated, we can calculate the DOA. Since the TDOA corresponds to the path length difference divided by the speed of sound, we can rearrange the formula and use the arcsine function to solve for θ . Thus, the DOA $\hat{\theta}$ is found using the formula:

$$\hat{\theta} = \arcsin\left(\frac{\hat{\tau}_{12}c}{d}\right)$$

9.1.3 TDOA Estimation: Cross-Correlation

Cross-correlation and generalized cross-correlation (GCC) are methods used for estimating the Time Difference of Arrival (TDOA) between signals received by a pair of microphones.

In the cross-correlation method, the idea is that the two microphones pick up the same sound signal at slightly different times, along with some disturbances like environmental noise and reverberation. If the noise and reverberation are not too strong compared to the signal, it's possible to cross-correlate the signals from the two microphones to estimate the TDOA. The cross-correlation function can be obtained by taking the inverse Fourier Transform of the cross-spectrum of the two signals. This function gives the degree of similarity between the two signals as a function of the time lag between them. The TDOA corresponds to the time lag where this cross-correlation function reaches its maximum value.



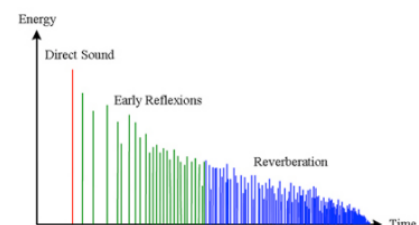
$$R_{12}(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X_1(\omega) \cdot X_2^*(\omega) \cdot e^{j\omega t} d\omega \longrightarrow \hat{\tau}_{12} = \underset{\tau \in D}{\operatorname{argmax}} (R_{12}(\tau)) \text{ with } D \in \left[-\frac{d}{c}, \frac{d}{c}\right]$$

However, the cross-correlation function may be affected by periodicities in the signal, which can produce additional peaks due to the auto-correlation of the signal. These spurious peaks may obscure the true TDOA peak. To address this, GCC methods can be employed. One such method involves whitening the signal spectrum, which aims to make the signal behave more like a white signal, where all frequencies are equally represented. This is done by introducing a weight function in the cross-correlation function to mitigate the effect of periodicities.

$$R_{12}(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \Psi_{12}(\omega) \cdot X_1(\omega) \cdot X_2^*(\omega) \cdot e^{j\omega t} d\omega \quad \text{where} \quad \Psi_{12}(\omega) = \frac{1}{|X_1(\omega) \cdot X_2^*(\omega)|}$$

9.1.4 TDOA Estimation: Adaptive Eigenvalue Decomposition (AED)

Adaptive Eigenvalue Decomposition (AED) is a signal processing technique utilized to estimate the Time Difference of Arrival (TDOA) of a signal reaching two spatially separated microphones. TDOA estimation is fundamental in applications such as sound source localization and acoustic environment mapping.



The AED technique follows several key steps:

1. **Channel Response Estimation:** AED begins by aiming to estimate the channel responses of the environment through which the sound propagates. The rationale is that the first significant peak in these impulse responses corresponds to the direct path of sound traveling from the source to each microphone. By identifying these peaks, we can estimate the TDOA as the difference in time-lags related to these direct paths.
2. **Mathematical Modelling:** In an idealized noise-free setting, the signals $y_1(n)$ and $y_2(n)$ captured by the microphones are the convolution of the source signal $s(n)$ with each microphone's channel impulse response, $g_1(n)$ and $g_2(n)$, respectively. Assuming the source signal is the same for both paths, we have

$y_1(n) * g_2(n) = s(n) * g_1(n) * g_2(n) = y_2(n) * g_1(n)$. This relationship can be expressed in a matrix form where the signal vectors are framed in M -long segments.

$$\mathbf{y}_1^T(n)\mathbf{g}_2 = \mathbf{y}_2^T(n)\mathbf{g}_1 \implies \mathbf{y}_1^T(n)\mathbf{g}_2 - \mathbf{y}_2^T(n)\mathbf{g}_1 = 0$$

where:

$$\mathbf{y}_k(n) = [y_k(n) \quad y_k(n-1) \quad \cdots \quad y_k(n-M+1)]^T$$

$$\mathbf{g}_k = [g_k(0) \quad g_k(1) \quad \cdots \quad g_k(M-1)]^T$$

3. **Statistical Representation:** Using the framed signal vectors, a covariance matrix R is constructed. This matrix is the statistical representation of the system and is used to find the vector u that satisfies $Ru = 0$, where u is the eigenvector associated with the zero eigenvalue of R .
4. **Eigenvalue Decomposition:** The eigenvalue decomposition of R is typically resource-intensive, scaling cubically with the matrix dimensions $O((2M)^3)$. To handle this, AED adopts an iterative search strategy to find the smallest eigenvalue and its associated eigenvector, thus bypassing direct decomposition.
5. **Iterative Optimization:** The objective is to minimize the norm $\|Ru\|$ of the vector Ru , subject to the constraint that $u^T u = 1$ to avoid the trivial solution of $u = 0$. The optimization problem is hence: $\hat{u} = \arg \min_u u^T Ru$ subject to $u^T u = 1$.
6. **LMS-like Procedure:** AED applies a method akin to the Least Mean Squares algorithm for this optimization. An error signal $e(n) = \frac{u^T(n)y(n)}{\|u(n)\|}$, where $y(n) = [y_1^T y_2^T]^T$, is defined, and the update step for u uses the gradient of this error with respect to u , leading to the update equation: $u(n+1) = u(n) - \mu e(n) \nabla e(n)$.
7. **Convergence and TDOA Estimation:** With successive iterations, the algorithm converges, and the final vector $u(\infty)$ can be proven to give the smallest eigenvalue of R . This converged vector is used to extract the estimated channel responses $\hat{g}_1(n)$ and $\hat{g}_2(n)$. The propagation delays $\hat{\tau}_k$ are found by locating the maximum of $\hat{g}_k(\tau)$, for $k = 1, 2$, and the TDOA τ_{12} is then calculated as $\tau_{12} = \hat{\tau}_2 - \hat{\tau}_1$.

In this methodology, AED transforms the problem of estimating TDOA into a solvable eigenvalue problem. By employing statistical signal processing techniques and iterative optimization algorithms, it achieves an efficient and accurate estimation of TDOA, crucial for applications that rely on precise source localization.

10 MICROPHONE ARRAYS: ACOUSTIC SOURCE SEPARATION

Imagine having several musical instruments playing in a room, and you're trying to record each one separately using multiple microphones. Each microphone captures a mixture of sounds from all the instruments, and your task is to separate the sounds to get a clean recording of each instrument.

The problem starts with N acoustic sources emitting signals in an environment. These sources could be anything that produces sound. The signals from these sources are denoted by $s_1(n), s_2(n), \dots, s_N(n)$, where n represents the sample index over time. The environment also has M microphones, where $M > 1$, capturing the sounds. The signals captured by the microphones are denoted by $y_1(n), y_2(n), \dots, y_M(n)$. The microphones pick up a mixture of all the source signals.

The goal is to estimate the original source signals $s_i(n)$ from the mixed signals recorded by the microphones $y_i(n)$.

There are different approaches to this problem, which can be categorized based on several criteria:

1. **How much information is available about the mixing process?**
 - **Blind Source Separation (BSS)** assumes little to no prior information about how the source signals were mixed together by the microphones. The aim is to infer the mixing process parameters solely from the mixed signals and any known statistics of the source signals. This method has some level of uncertainty in the de-mixing process.
 - **Non-Blind Source Separation**, on the other hand, assumes that some information about the mixing process, like the channel or impulse responses, is known. This makes the separation process more a matter of inverting a known system.

2. Is the environment reverberant or not?

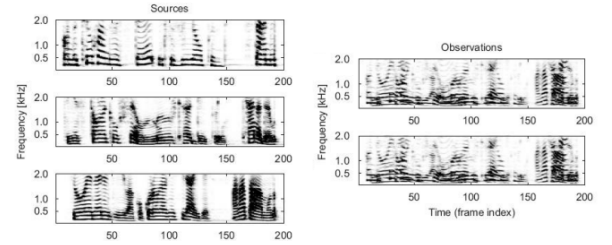
- **Instantaneous mixing** assumes no or very low reverberation. The mixing process is simply a linear combination of the source signals, which can be represented in a matrix form. The challenge here is to invert the mixing matrix to recover the original signals.
- **Convulsive mixing** considers a reverberant environment where the mixing process is more complex. The recovery of the original signals involves inverting a convolutive system, which is typically more challenging than instantaneous mixing due to the effects of reverberation.

3. Is the number of microphones greater than the number of sources (overdetermined) or vice versa (underdetermined)?

- If there are more microphones than sources $M > N$, the problem is overdetermined, and typically there are more equations than unknowns, which might make the problem easier to solve in theory.
- If there are fewer microphones than sources $M < N$, the problem is underdetermined. This requires estimating the mixing model with the help of some a priori knowledge about the sources because there are more unknowns than equations.

10.1 SOURCE SEPARATION : BINARY MASKS

We are in an underdetermined problem where you have fewer microphones ($M = 2$) than sources ($N = 3$) to record speech signals in a low-reverberation environment. The sources are stationary in space, and we need to separate the sources without prior information about the mixing process, which is known as Blind Source Separation (BSS).



The strategy is to convert the signals to the time-frequency domain using the Short-Time Fourier Transform (STFT). This conversion is motivated by the fact that speech signals are typically more sparse in the time-frequency domain than in the time domain, which means that at any given time and frequency, each time-frequency point is likely to be dominated by just one source.

In the time-frequency domain, the microphone signals are denoted by $Y_1(n, \omega_k)$ and $Y_2(n, \omega_k)$, and the source signals by $S_1(n, \omega_k)$, $S_2(n, \omega_k)$, and $S_3(n, \omega_k)$. It's assumed that for each time-frequency pair n, ω_k , the microphone signals are mainly associated with just one of the sources.

The objective is to find three binary masks M_1, M_2, M_3 such that the STFTs of the sources can be estimated by applying these masks to one of the microphone signals (e.g., $Y_1(n, \omega_k)$).

$$\hat{S}_i(n, \omega_k) = Y_1(n, \omega_k) \cdot M_i(n, \omega_k), \quad i = 1, 2, 3$$

To achieve this, you can exploit spatial information inherent in the signals. In the absence of reverberation, the microphone signals will contain attenuated and delayed versions of the source signals. For instance, if you consider the i_{th} source, the two microphone signals can be represented as scaled and phase-shifted versions of the source signal in the frequency domain.

- example for bins associated to the i -th source:

$$\begin{cases} Y_1(n, \omega_k) = \alpha_{1i} \cdot S_i(n, \omega_k) e^{-j\omega_k \tau_{1i}} \\ Y_2(n, \omega_k) = \alpha_{2i} \cdot S_i(n, \omega_k) e^{-j\omega_k \tau_{2i}} \end{cases}$$

Each time-frequency bin associated with the i_{th} source will exhibit a constant time-delay T_i and a constant attenuation ratio A_i . By comparing the two microphone signals, you can calculate these constants for each time-frequency bin.

- constant time-delay: $T_i = \tau_{i2} - \tau_{i1}$
- constant attenuation ratio: $A_i = \alpha_{2i} / \alpha_{1i}$

For each time-frequency bin, you compute the amplitude ratio $A(n, \omega_k)$ and the time delay $T(n, \omega_k)$. These values will form clusters in the amplitude-time delay space, typically with one cluster per source.

$$\begin{aligned} \circ \quad A(n, \omega_k) &= \frac{|Y_2(n, \omega_k)|}{|Y_1(n, \omega_k)|} \rightarrow \text{amplitude ratio} \\ \circ \quad T(n, \omega_k) &= \frac{\arg Y_2(n, \omega_k) - \arg Y_1(n, \omega_k)}{\omega_k} \rightarrow \text{time delay} \end{aligned}$$

Using a clustering algorithm like k_{means} , you can assign each time-frequency bin to one of the three sources based on the proximity to the cluster centroids, which are represented by the pairs (A_i, T_i) .

After associating each time-frequency bin with a source, binary masks $M_i(n, \omega_k)$ are computed, where each mask takes the value 1 if the bin belongs to the cluster associated with that source, and 0 otherwise.

$$M_i(n, \omega_k) = \begin{cases} 1 & \text{if } [A(n, \omega_k), T(n, \omega_k)] \text{ belongs to cluster } [A_i, T_i] \\ 0 & \text{otherwise} \end{cases}$$

To refine the process, you can employ multi-dimensional clustering techniques that consider additional features beyond just the amplitude ratio and time delay. For example, features that might be more suitable for clustering could include the normalized amplitude ratio and the phase difference, which help to account for potential imbalances in feature variance that could affect the clustering process.

10.2 SOURCE SEPARATION: INDEPENDENT COMPONENT ANALYSIS

Now consider the case there are more microphones than sources, there is no information about the mixing matrix and no reverberant environment ($y = As$). We will estimate the inverse matrix $W = A^{-1}$ that will de-mix the signals ($s = Wy$). To do so we will assume that the sources are independent (no correlation, thus correlation matrix is diagonal) and the sources have non-Gaussian statistics (this will make sense at the end). There are two main ambiguities: we are not able to find the variances of the sources that are independent, and we cannot determine the order of independent components, we can freely change the order of the terms in the sum without influencing the final result. The key concept is the one of independence, which means that the joint probability is the product of the two (knowing one of the two signals doesn't tell us anything about the other signal), and the signals are uncorrelated (the covariance is zero). These assumptions were valid before the mixing process, but after it we are giving some sort of correlation to the signals because they are mixed together and are not independent anymore (in other words the signals are now sort of correlated because they are all multiplied by the same matrix). If we know the edges of the joint distribution, we can estimate the de-mixing matrix.

Before we said that we don't want a Gaussian distribution because the mixing matrix will not have any effect on it, therefore separation is not possible with this method (remember that in general, statistical operations made on Gaussians will always return a Gaussian, so we can't de-mix the signals). We can apply the central limit theorem to estimate the mixing matrix (the theorem said that the distribution of a sum of independent random variables tends toward a Gaussian distribution). If we sum up all independent variables we will return a distribution closer to a Gaussian distribution. Because the microphoned signal is more Gaussian than the original signals (because as said before we are combining the signals together, thus more Gaussian distribution), we will find the matrix that minimises the Gaussianity (the original matrix A was making the mixing Gaussian, therefore we need to apply a matrix that reduces this effect). We can achieve perfect separation if $W = A^{-1}$. There are different methods of measuring the non-Gaussianity (Kurtosis, Negentropy). We can also have some measures of independency, like mutual information. Once we found this cost function we will estimate the de-mixing matrix maximizing or minimizing it (it depends on the independence metric used). The de-mixing model is typically estimated through LMS-like iterative algorithms because the cost function is strongly not-linear, therefore the iterative algorithm could be trapped into a local minimum/maximum, so the solution may depend on the initial conditions adopted in each realization.

11 MATLAB

11.1 DEFINITION OF A SINUSOIDAL SIGNAL

```

%
% Define the time vector (t) and the sampling frequency (fs)
fs = 1000; % Sampling frequency in Hz
T = 1/fs; % Sampling period in seconds
t = 0:T:0.005; % Time vector in seconds

% Define the amplitude (A) and the phase (phi) of the sinusoid
A = 2; % Amplitude
phi = 0; % Phase in radians

% Generate the sinusoidal signal (x)
x = A * sin(2 * pi * fs * t + phi);

% Plot the sinusoidal signal
plot(t, x);
xlabel('Time (s)');
ylabel('Amplitude');

```

```

%
% Define the number of samples (N) and the sampling frequency (fs)
N = 1000; % Number of samples
fs = 1000; % Sampling frequency in Hz

% Define the amplitude (A) and the phase (phi) of the sinusoid
A = 2; % Amplitude
phi = 0; % Phase in radians

% Generate the sinusoidal signal (x)
x = A * sin(2 * pi * fs * (0:N-1)/N + phi);

% Plot the sinusoidal signal
plot(0:N-1, x);
xlabel('Sample');
ylabel('Amplitude');

```

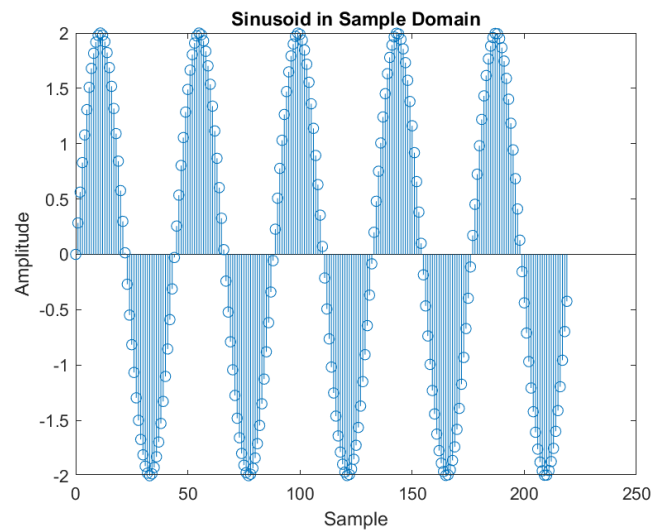
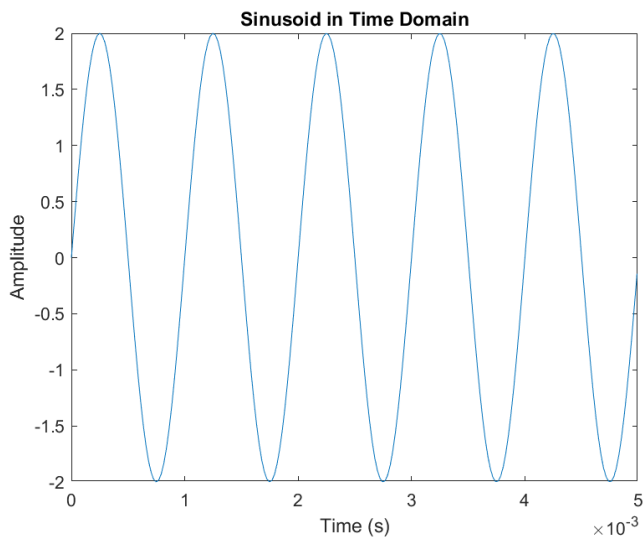


Figure 31

11.2 MODELLING THE NOISE

% Modeling the noise

% Create a sin with $F_s = 44100$, $A = 2$, $f = 1000$, $\phi = \pi/2$ and duration = 0.01s.

Amplitude = 2; $F_s = 44100$; $f = 1000$; $\phi = \pi/2$; duration = 0.01;

time = 0 : 1/ F_s : duration-1/ F_s ;

$x = \text{Amplitude} * \sin(2 * \pi * f * \text{time} + \phi)$;

% Create some noise with SNR = 10.

SNR_dB = 10;

std = Amplitude/($10^{(\text{SNR_dB}/20)}$);

noise = std*randn(size(x));

% Add the noise to the signal.

$s = x + \text{noise}$;

% Plot

figure;

subplot(1,2,1);

plot (time, x);

title("Clean Signal");

subplot(1,2,2);

plot(time,s);

title("Noisy Signal");

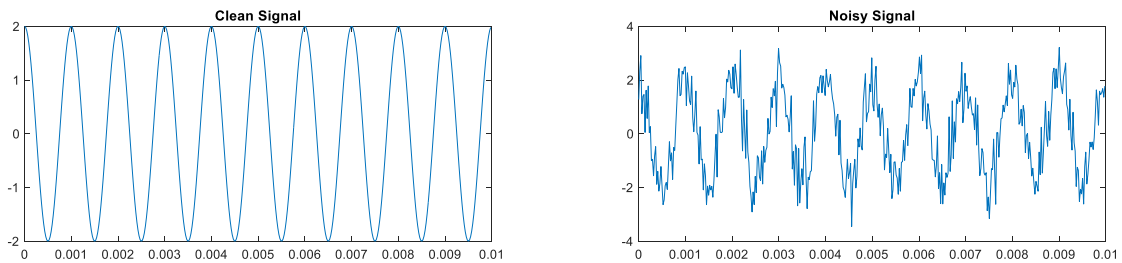


Figure 32

Note: $SNR = \frac{P_{\text{signal}}}{P_{\text{noise}}}$, and in signal processing (in which we consider electrical power) the power is defined as $P =$

$\frac{\text{Amplitude}^2}{R}$, so the Signal Noise Ratio formula becomes $SNR = \frac{P_{\text{signal}}}{P_{\text{noise}}} \rightarrow \left(\frac{A_{\text{signal}}^2}{R} * \frac{R}{A_{\text{noise}}^2} \right) = \frac{A_{\text{signal}}^2}{A_{\text{noise}}^2} = \left(\frac{A_{\text{signal}}}{A_{\text{noise}}} \right)^2$.

And if we express the SNR in decibel we get

$$SNR_{dB} = 10 * \log_{10} \left(\frac{A_{\text{signal}}}{A_{\text{noise}}} \right)^2 = 20 * \log_{10} \left(\frac{A_{\text{signal}}}{A_{\text{noise}}} \right)$$

When we want to model noise and the problem gives us the SNR (like in the code snippet above), we must extract

the amplitude of the noise signal (std of the gaussian): $A_{\text{noise}} = \frac{A_{\text{signal}}}{10^{\frac{SNR_{dB}}{20}}}$.

11.3 FIR FILTERING

11.3.1 FIR filtering via Time-Domain Convolution

A way of computing FIR filtering is through direct computation of the linear convolution of an input signal $x(n)$ with an FIR filter represented by $h(n)$.

The filter is depicted as a series of delay elements ($z - 1$), each followed by a multiplier for the filter coefficients $h(0), h(1), \dots, h(N)$, and the outputs of these multipliers are then summed to produce the output $y(n)$.

The convolution is mathematically expressed as $y(n) = \sum_{i=0}^N h(i)x(n-i)$.

The complexity of this operation is $O(N^2)$, which means the number of computations required grows quadratically with the number of taps in the filter, and this could be a computational issue for large filters.

For implementation in MATLAB, the convolution can be performed using the **conv(x, h)**; command.

11.3.2 FIR filtering via Frequency-Domain Convolution

For the convolution theorem, convolution in the time domain corresponds to multiplication in the frequency domain.

Frequency-domain convolution can be implemented efficiently using the Fast Fourier Transform (FFT):

1. Compute the FFT of both x and h with complexity $O(N \log(N))$.
2. Multiply the two FFTs point-wise, which is an $O(N)$ operation.
3. Perform the Inverse FFT (IFFT) to get the convolved signal with complexity $O(N \log(N))$.

It's important to consider that when we perform the convolution in frequency domain through FFT, "we are sampled and periodic in time and frequencies", and so we convolution is a cyclic convolution. For this reason, it's important to zero-pad the signals in FFT-based convolution to ensure acyclic (linear) convolution (emulate a linear convolution through cyclic convolution).

For a signal $x(n)$ and an impulse response $h(n)$, the length of the resulting signal is $N_y = N_x + N_h - 1$, so we must pad the signals with zeros until they reach that length.

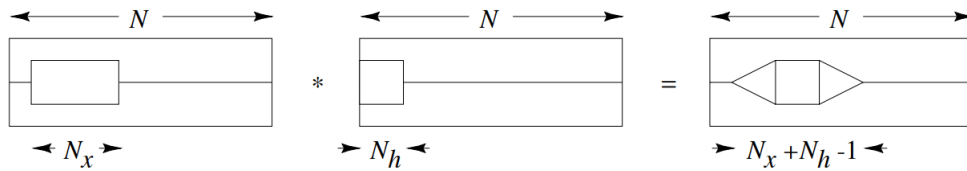


Figure 33

11.3.3 Filtering in Time/Freq domain (MATLAB)

To understand the difference between the two methods, let's implement an LPF:

```
X
%% Filtering in Time Domain
% Generate two signals.
Fs = 8000; f1 = 500; f2 = 1000; duration = 0.06;
time = 0 : 1/Fs : duration-1/Fs;
x1 = sawtooth(2*pi*f1*time);
x2 = square(2*pi*f2*time);

% Design a LPF in time domain (it is a rectangle in the frequency domain).
FilterLength = 512;
freq_cut = 1500;
n_axis = -(FilterLength-1)/2 : (FilterLength-1)/2; % Centre the filter around
zero.
h_ideal = sin(2*pi*freq_cut/Fs*n_axis) ./ (pi*n_axis); % sinc function.
window = hamming(FilterLength);
h = h_ideal' .* window;

% Filter in time domain
y1 = filter (h,1, x1); % or y1 = conv (h, x1);
y2 = filter (h,1, x2); % or y2 = conv (h, x2);

% Plot
figure(1);
title("Time Domain Filtering");
subplot(2,2,1); plot(time,x1); title('x1');
subplot(2,2,3); plot(time,x2); title("x2");
subplot(2,2,2); plot(time, y1); title("y1 = x1 * h");
subplot(2,2,4); plot(time, y2); title("y2 = x2 * h");

%% Filtering in Frequency domain

H = fft(h,2048);
X1 = fft(x1,2048); X2 = fft(x2,2048);
Y1 = H' .* X1; Y2 = H' .* X2;
y1 = ifft(Y1); y2 = ifft(Y2);

y1 = y1(1:length(x1));
y2 = y2(1:length(x2));

figure(2);
title("Frequency Domain Filtering");
subplot(2,2,1); plot(time,x1); title('x1');
subplot(2,2,3); plot(time,x2); title("x2");
subplot(2,2,2); plot(y1); title("y1 = x1 * h");
subplot(2,2,4); plot(y2); title("y2 = x2 * h");
```

11.3.4 Overlap And Add

When signals are of indefinite length or very large, using a single Fast Fourier Transform (FFT) becomes impractical. The OLA method overcomes this by processing the signal in smaller sections or frames. OLA is suitable for real-time processing as it does not require the entire signal to be present before starting the filtering process.

The algorithm steps are the following:

1. **Block subdivision:** the input signal is divided into overlapping blocks or frames. These frames are typically windowed (multiplied) by a function to manage discontinuities at the block edges.
2. **Frequency Domain Convolution:** each block is transformed into the frequency domain using the FFT. The filter is also transformed into the frequency domain once and reused for each block. Block-wise convolution is then performed by multiplying the transformed block with the transformed filter.

3. **Overlap-Add:** after the convolution of each block, the result is transformed back to the time domain using the Inverse FFT (IFFT). The overlapping parts of these blocks are then added together to reconstruct the continuous output signal.

11.3.4.1 COLA condition

The Constant Overlap-Add (COLA) condition is a crucial requirement in the Overlap-Add method, ensuring that segmented processing of signals does not distort the signal when segments are overlapped and added together.

When you process a signal in segments, as in the OLA method, you apply a window to each segment, and then overlap these segments by a certain amount when you reconstruct the signal. For the reconstruction to be perfect, the overlapped windows must add up to a constant value at every point in time where they overlap. This constant value ensures that the original signal's amplitude is preserved when the windowed segments are added back together.

x
 $i x \tilde{a} \frac{1}{2} \tilde{U} \tilde{L} \tilde{I} \tilde{u} x \quad l p x \quad p p x [\tilde{E} \tilde{L} \tilde{s} \tilde{U} \tilde{E} \tilde{L} \tilde{D} x$
 $i x ? \frac{1}{2} \tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{x} \tilde{s} \tilde{e} \tilde{x} \tilde{O} \tilde{E} \tilde{D} \tilde{L} \tilde{I} \tilde{L} x$
 $A x \rightarrow x \frac{1}{2} \tilde{U} \tilde{O} \tilde{W} \quad U \tilde{C} U U U \tilde{U} \tilde{Z} x$
 $A \quad U \tilde{C} U \tilde{C} \frac{1}{2} \tilde{l} p \Sigma x \rightarrow x U \tilde{Z} x$
 $i x ? \frac{1}{2} \tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{x} \tilde{s} \tilde{e} \tilde{x} [\tilde{E} \tilde{L} \tilde{s} \tilde{U} \tilde{x} x$
 $\tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{U} \tilde{x} \frac{1}{2} l x \rightarrow x U \tilde{U} \tilde{Z} x$
 $e x \rightarrow x \tilde{c} \tilde{E} \tilde{U} \tilde{U} \quad \tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{U} \tilde{x} \frac{1}{2} l \quad U \tilde{C} x U \tilde{T} \tilde{O} \tilde{Z} x$
 $i x ? \frac{1}{2} \tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{x} \tilde{s} \tilde{e} \tilde{x} \tilde{A} \tilde{E} \tilde{l} p \tilde{O} \quad x \tilde{I} l p x \tilde{s} \tilde{e} \tilde{x} \tilde{c} \tilde{O} \tilde{u} \tilde{O} \tilde{E} \tilde{L} \tilde{s} x$
 $\tilde{E} l x \frac{1}{2} l x \rightarrow x U \tilde{U} \tilde{Z} x$
 $\tilde{E} l x \rightarrow x \tilde{e} \tilde{I} l l \tilde{E} \tilde{L} \tilde{D} " \quad \tilde{E} l x \frac{1}{2} l \Sigma \tilde{I} \tilde{Z} x$
 $\tilde{e} \tilde{O} \tilde{u} \tilde{O} \tilde{E} \tilde{L} \tilde{s} x \rightarrow x \frac{1}{2} \tilde{E} \tilde{L} " \quad \tilde{E} l x \frac{1}{2} l \quad \Sigma \tilde{Z} x i x \tilde{c} \tilde{O} \tilde{u} \tilde{O} \tilde{E} \tilde{L} \tilde{s} x \rightarrow x \tilde{O} \tilde{U} i x \tilde{E} l x \frac{1}{2} l x$
 $\tilde{c} \tilde{c} s x \frac{1}{2} l x \rightarrow x \quad x = x \quad \frac{1}{2} \tilde{E} \tilde{L} \quad \tilde{L} \tilde{O} \tilde{D} \quad \tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{U} \tilde{x} \frac{1}{2} l x + x \tilde{E} l x \frac{1}{2} l x \quad x U \Sigma \Sigma \tilde{Z} x x$
 $i x \tilde{a} \quad \frac{1}{2} \tilde{U} \tilde{L} \tilde{I} \tilde{u} x \tilde{I} l p x \quad p p x$
 $B x \rightarrow x \frac{1}{2} \tilde{U} \tilde{O} \tilde{W} \quad U \tilde{C} x \tilde{L} \frac{1}{2} l \tilde{D} \tilde{s} \tilde{e} \quad A \Sigma x + x \tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{U} \tilde{x} \frac{1}{2} l x \quad x U \Sigma \tilde{Z} x$
 $C x \rightarrow x \tilde{c} \tilde{c} s \quad \tilde{e} \tilde{C} x \tilde{c} \tilde{c} s x \frac{1}{2} l \Sigma \tilde{Z} x x$
 $l \tilde{L} l [\tilde{U} \tilde{I} l \frac{1}{2} \tilde{W} x \rightarrow x \tilde{L} \tilde{O} \tilde{O} \tilde{U} \quad \tilde{L} \frac{1}{2} l \tilde{D} \tilde{s} \tilde{e} \quad A \Sigma x \quad x \quad \tilde{E} l x \frac{1}{2} l \Sigma x \quad x \tilde{e} \tilde{O} \tilde{u} \tilde{O} \tilde{E} \tilde{L} \tilde{s} \Sigma \tilde{Z} x$
 $\tilde{c} \tilde{O} \tilde{U} x \tilde{E} x \rightarrow x U x \tilde{C} x l \tilde{L} l [\tilde{U} \tilde{I} l \frac{1}{2} \tilde{W} x$
 $x x x x \tilde{l} \tilde{U} \tilde{U} [\tilde{U} \tilde{I} l \frac{1}{2} f ? - x \rightarrow x \quad \tilde{E} \quad U \Sigma ' \tilde{e} \tilde{O} \tilde{u} \tilde{O} \tilde{E} \tilde{L} \tilde{s} + U x \tilde{C} x \quad \tilde{E} \quad U \Sigma ' \tilde{e} \tilde{O} \tilde{u} \tilde{O} \tilde{E} \tilde{L} \tilde{s} x + x \tilde{E} l x \frac{1}{2} l \tilde{Z} x$
 $x x x x A [\tilde{U} \tilde{I} l \frac{1}{2} x \rightarrow x A \quad \tilde{l} \tilde{U} \tilde{U} [\tilde{U} \tilde{I} l \frac{1}{2} f ? - \Sigma x \tau ' x \tilde{E} l \tilde{Z} x$
 $x x x x - [\tilde{U} \tilde{I} l \frac{1}{2} x \rightarrow x \tilde{c} \tilde{c} s \quad A [\tilde{U} \tilde{I} l \frac{1}{2} \tilde{C} x \tilde{c} \tilde{c} s x \frac{1}{2} l \Sigma \tilde{Z} x$
 $x x x x " [\tilde{U} \tilde{I} l \frac{1}{2} x \rightarrow x - [\tilde{U} \tilde{I} l \frac{1}{2} x \tau ' x \tilde{c} \tilde{Z} x$
 $x x x x B [\tilde{U} \tilde{I} l \frac{1}{2} x \rightarrow x \tilde{E} \tilde{c} \tilde{c} s \quad " [\tilde{U} \tilde{I} l \frac{1}{2} \Sigma \tilde{Z} x$
 $x x x x B [\tilde{U} \tilde{I} l \frac{1}{2} x \rightarrow x B [\tilde{U} \tilde{I} l \frac{1}{2} \quad U \tilde{C} x \tilde{E} l x \frac{1}{2} l x + x \tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{U} \tilde{x} \frac{1}{2} l x \quad x U \Sigma \tilde{Z} x$
 $x x x x B f ? - x \rightarrow x \tilde{E} \quad U \Sigma ' \tilde{e} \tilde{O} \tilde{u} \tilde{O} \tilde{E} \tilde{L} \tilde{s} x + x U x \tilde{C} x \quad \tilde{E} \quad U \Sigma ' \tilde{e} \tilde{O} \tilde{u} \tilde{O} \tilde{E} \tilde{L} \tilde{s} x + x " \quad \tilde{E} l x \frac{1}{2} l + \tilde{c} \tilde{E} \tilde{L} \tilde{s} \tilde{U} \tilde{x} \frac{1}{2} l \quad U \Sigma \tilde{Z} x$
 $x x x x B \quad B f ? - \Sigma x \rightarrow x B \quad B f ? - \Sigma + B [\tilde{U} \tilde{I} l \frac{1}{2} \tilde{Z} x$
 $\frac{1}{2} l p x$
 $i x \tilde{c} \tilde{L} \tilde{O} \tilde{s} x$
 $\tilde{c} \tilde{E} \tilde{D} \tilde{L} \tilde{U} \frac{1}{2} \quad U \Sigma \tilde{Z} x$
 $\tilde{W} \tilde{L} \tilde{Z} \tilde{u} \tilde{L} \tilde{O} \tilde{s} \quad U \tilde{C} \quad \tilde{C} U \Sigma \tilde{Z} x$
 $\tilde{u} \tilde{L} \tilde{O} \tilde{s} \quad A \Sigma \tilde{Z} x$
 $\tilde{s} \tilde{E} \tilde{s} \tilde{L} \frac{1}{2} \quad \tilde{I} \tilde{A} \tilde{I} \Sigma \tilde{Z} x$
 $\tilde{W} \tilde{L} \tilde{Z} \tilde{u} \tilde{L} \tilde{O} \tilde{s} \quad U \tilde{C} \quad \tilde{C} \quad \Sigma \tilde{Z} x$
 $\tilde{u} \tilde{L} \tilde{O} \tilde{s} \quad B \Sigma \tilde{Z} x$
 $\tilde{s} \tilde{E} \tilde{s} \tilde{L} \frac{1}{2} \quad \tilde{I} \tilde{B} \tilde{I} \Sigma \tilde{Z} x$