DFT

Fourier Transform for Audio Processing and Noise Reduction

Raha Rahmanian

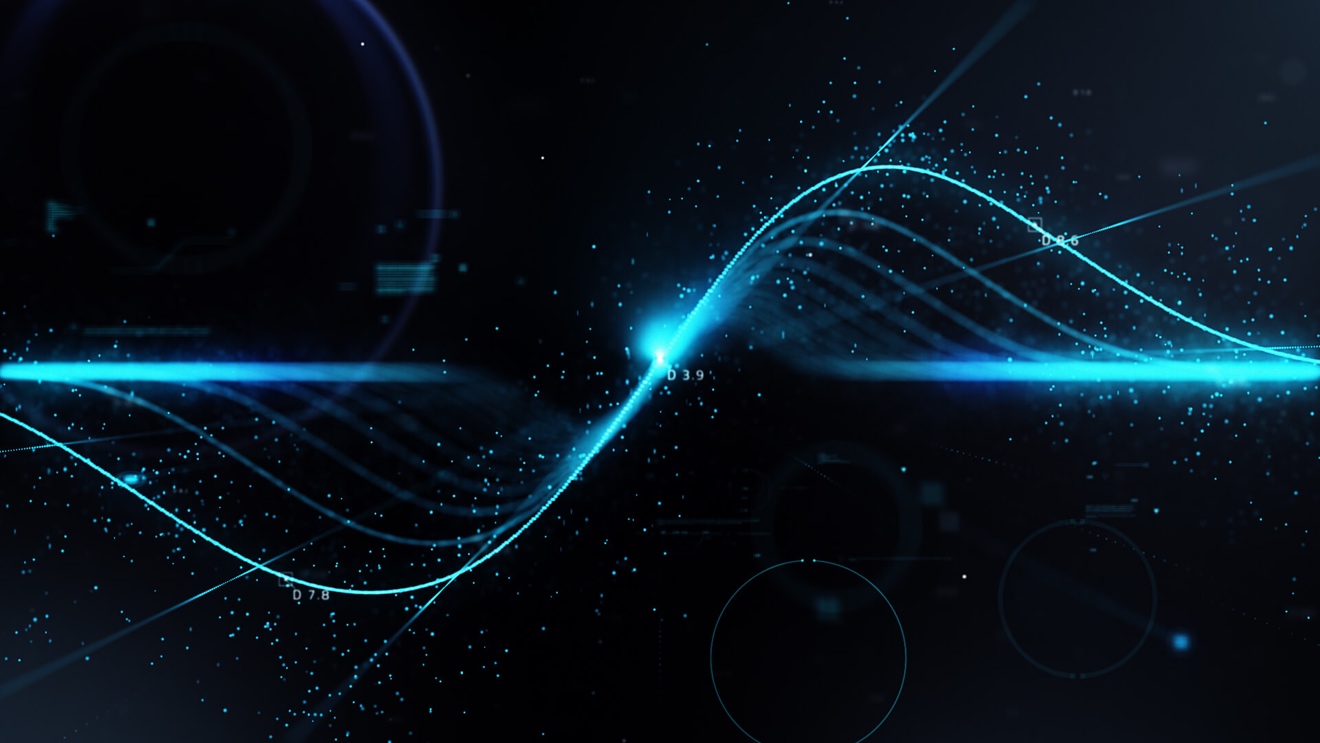


TAble of  
Contents

Introduction 1

How Does the DFT Work? 1

Why is the DFT Important in Signal Processing? 2

Project Purpose & Objective: 2

Methodology 3

Frequency Bins 4

Sampling Rate and Nyquist Frequency 4

Calculating the Frequency Axis 4

Filter Design and Application 5

Filter Design 5

1. Practicality of Data 8

Results 9

conclusion 10

## Introduction

So, if you are new to this just as I myself was before this project you must be wondering what is the difference between time domain and frequency domain? what is DFT? And more importantly what use does it have, so let me explain each briefly:

The time domain is the representation of a signal in terms of its time variation, while the frequency domain is the representation of a signal in terms of its frequency components

DFT:

The Discrete Fourier Transform (DFT) is a mathematical technique used to transform a sequence of values (usually a discrete signal in the time domain) into components of different frequencies (the frequency domain).

### How Does the DFT Work?

The DFT converts a sequence of N complex numbers   ​ into another sequence of N complex numbers.The formula for the DFT is:

Where:

* ​ is the output sequence (the frequency components).
* is the input sequence (the time-domain signal).
* is the imaginary unit.
* is a complex exponential function representing the sinusoidal basis functions.

### Why is the DFT Important in Signal Processing?

The Discrete Fourier Transform (DFT) is one of the most important tools in Digital Signal Processing.

1. **Calculating a signal's frequency spectrum.** This is a direct examination of information encoded in the frequency, phase, and amplitude of the component sinusoids. For example, human speech and hearing use signals with this type of encoding.
2. **finding a system's frequency response from the system's impulse response, and vice versa**. This allows systems to be analyzed in the frequency domain, just as convolution allows systems to be analyzed in the time domain.
3. **Filtering**: By transforming a signal to the frequency domain, we can easily filter out unwanted frequencies or noise.
4. **Compression**: Signals often have redundant information in the time domain. By transforming to the frequency domain, it is possible to identify and eliminate this redundancy.

### Project Purpose & Objective:

The purpose of this project is to gain a thorough understanding of Fourier analysis and its practical application in audio processing, specifically through the implementation of the Discrete Fourier Transform (DFT). By manually writing a DFT algorithm in MATLAB, analyzing the frequency components of audio signals, and designing low-pass, high-pass, and band-pass filters, the project aims to evaluate the effectiveness of these filters in noise reduction, culminating in the conversion of filtered signals back to the time domain for comprehensive analysis and evaluation.

## Methodology

It is quite a strait forward process to load and play sounds in MATLAB. There are several functions that can be used for this purpose the one I have used in this project is sound function:

% Read the audio file  
[audioData, sampleRate] = audioread(filePath);

Another alternative is using the player function:

% Create an audioplayer object  
player = audioplayer(audioData, sampleRate);  
  
% Play the audio  
 play(player);  
  
% To pause the audio  
 pause(player);  
  
% To resume the audio  
 resume(player);  
  
% To stop the audio  
 stop(player);

In the next step we apply the DFT function. I have used the DFT formula that I mentioned above to manually perform DFT on my function but this approach is not very efficient because the complexity of this algorithm is O(). There is an alternative function in MATLAB called FFT (Fast Fourier Transform).

function X = myDFT(x)  
N = length(x);  
 X = zeros(1, N);  
 for k = 0:N-1  
 for n = 0:N-1  
 X(k+1) = X(k+1) + x(n+1) \* exp(-1j \* 2 \* pi\*k\*n / N);  
 end  
 end  
 en

Now in order to find the frequency axis there are a couple of things to be considered:

### Frequency Bins

1. **DFT Output**: The DFT of a signal with NN samples will produce NN frequency bins.
   * Each bin corresponds to a specific frequency component in the signal.
   * The first bin (index 0) corresponds to the DC component (0 Hz).
   * Subsequent bins correspond to increasing frequencies.

### Sampling Rate and Nyquist Frequency

1. **Sampling Rate**: The sampling rate (fs) is the number of samples taken per second. It determines the highest frequency that can be accurately represented, known as the Nyquist frequency, which is half of the sampling rate (fs/2).

### Calculating the Frequency Axis

1. **Frequency Resolution**: The frequency resolution (Δf) is the spacing between adjacent frequency bins and is given by:
2. **Frequency Values**: The frequency for each bin can be calculated as:

where k is the bin index ranging from 0 to N−1.

The frequency bins are calculated in the code as instructed above

frequencies = (0:N-1) \* (sampleRate / N);

## Filter Design and Application

To identify the noise components in a signal we must distinguish the unwanted components (noise) from the actual signal based on their frequency characteristics; this can be achieved by plotting the signal after DFT has been applied on it. Noise often appears as peaks or spread-out energy in regions where there shouldn’t be any:

* **Low-Frequency Noise**: Common sources include power line hum (50/60 Hz) and mechanical vibrations. These components are usually found at the lower end of the frequency spectrum.
* **High-Frequency Noise**: This can be due to electronic interference, aliasing, or other high-frequency sources. These are located towards the higher end of the spectrum.
* **Broadband Noise**: This type of noise spans a wide range of frequencies and is often more challenging to filter out without affecting the desired signal.

We can block certain frequencies and allow certain frequencies to pass using filters. So we can remove noise from signals by applying the appropriate filter after recognizing the type of noise from the signal plot.

## Filter Design

The [Fourier transform](http://www.dsprelated.com/dspbooks/mdft/Fourier_Transform_FT_Inverse.html) can be defined for [signals](http://www.dsprelated.com/dspbooks/filters/Definition_Signal.html) which are

* discrete or continuous in time, and
* finite or infinite in duration.

This results in four cases. As you might expect, the [frequency domain](http://www.dsprelated.com/dspbooks/mdft/) has the same cases:

* discrete or continuous in frequency, and
* finite or infinite in [bandwidth](http://en.wikipedia.org/wiki/Bandpass).

When time is discrete, the frequency axis is finite, and vice versa.

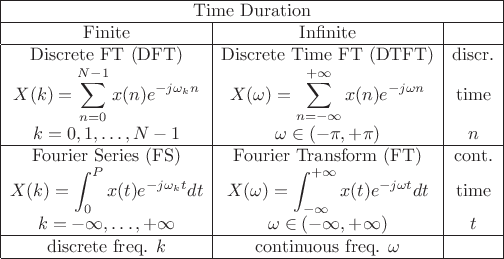


Figure 1.  Four cases of sampled/continuous finite/infinite time and frequency. (Often the FS coefficients are divided by the [period](http://en.wikibooks.org/wiki/Signals_and_Systems/Periodic_Signals)  .)

There are different kind of filters that can be applied on a signal, three of which have been implemented in this project:

* High-pass filter: This filter blocks low frequency parts of the signal and only allows high frequency parts to remain in the signal
* Low-pass filter: This filter blocks high frequency parts of the signal and only allows low frequency parts to remain in the signal.
* Band-pass filter: This filter only allows frequencies in range of two other given frequencies to remain and zeros all frequencies out of the given range.

These filters must be implemented differently on signals of different nature; this means that there is a fundamental difference in signals of continuous nature. The filtering is rather simple on continuous signals because the Fourier transform of these signals returns a frequency response that does not alternate. Filters for this type of transformation are shown in figure 1.

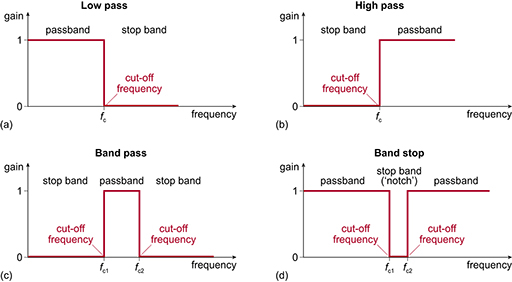


Figure 2.Continuous time signal filtering format.

But there is a difference when it comes to signals that are discrete in time domain because these signals are alternating in the frequency domain. Because of this we need our filters to be alternating as well;

The resulting filter will have the form shown in figure2.

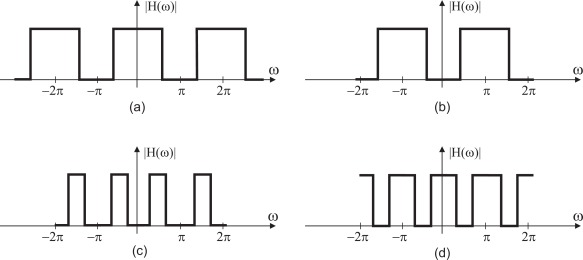


Figure 3.discrete time signals filtering format.

As described in the picture above the filtering for discrete time signals is different. So, you might be wondering why our filters in the project are not alternating although our signals are discrete in time domain after sampling and therefore it is expected that they would be alternating in frequency domain, the reason lies in the fact that we are using DFT instead of DTFT. So why do we use DFT?The Discrete Fourier Transform (DFT) is used in practice for analyzing signals like audio because it is suitable for finite, discrete data. Here's why the DFT is preferred over the Discrete-Time Fourier Transform (DTFT) in this context:

### 1. Practicality of Data

* **Voice Signal as Finite Sequence:** Although a voice signal can be considered theoretically infinite, in practice, we work with finite segments of the signal. When you record or analyze audio, you only have a finite portion of the signal available. The DFT is designed for finite-length sequences, which makes it practical for real-world applications.
* **Digital Storage and Processing:** Voice signals are sampled and stored digitally as finite sequences of numbers. The DFT operates on these discrete samples, making it the appropriate tool for digital signal processing (DSP).

Now you might also be wondering why this transformation is not creating an alternating result; The reason the DFT output doesn't alternate in a simple periodic manner is due to how it represents the frequency content of the signal:

1. **Nature of Frequency Components:**
   * The DFT output shows the amplitude and phase of different frequency components. These values do not inherently alternate between positive and negative.
2. **Magnitude and Phase Representation:**
   * When you look at the magnitude of the DFT output, you're looking at the strength of each frequency component, which is always non-negative and does not alternate.
   * The phase can alternate but is not typically presented as a simple alternation pattern; it depends on the specific signal being transformed.

Therefore, implementing our filters becomes quite straightforward:

function X = myHPF(signal, frequencies, cuttingFreq)  
 %finding the index of cutoff frequency  
 cutoffIndex = find(frequencies >= cuttingFreq, 1);  
signal(1:cutoffIndex) = 0;  
 X = signal;  
 end  
  
  
 function X = myLPF(signal,frequencies,cuttingFreq)  
 cutoffIndex = find(frequencies >= cuttingFreq, 1);  
 signal(cutoffIndex+1:end) = 0;  
 X = signal;  
 end

function X = myBPF(signal,frequencies,bandStartFreq,bandEndFreq)  
 cutoffIndex1 = find(frequencies >= bandStartFreq, 1);  
 cutoffIndex2 = find(frequencies >= bandEndFreq,1);  
 signal(1:cutoffIndex1) = 0;  
 signal(cutoffIndex2+1:end) = 0;  
 X = signal;  
 end

After applying filters on the signal, we need to transform it back to the time domain. This is achieved by performing IDFT on the signal.

IDFT:

function x = myIDFT(signalDFT)  
 N = length(signalDFT);  
 x = zeros(1, N);  
 for n = 0:N-1  
 for k = 0:N-1  
 x(n+1) = x(n+1) + (1/N) \* signalDFT(k+1) \* exp(1j \* 2 \* pi \* k \* n / N);  
 end  
 end  
end

## Results

To check the results, I have tested my code by creating a signal with frequencies of desired range and with noise applied on it, and after that I used my code above to filter out the range that previously belonged to my signal and I checked the result using sound function to actually test if the noise is reduced.

I also plotted my signals in frequency domain and time domain to check if my filters are working correctly and see the difference in noise reduction.

## conclusion

In this project, I gained a comprehensive understanding of the Discrete Fourier Transform (DFT) and its pivotal role in signal processing. By delving into the principles of the DFT, I learned to distinguish between time-domain and frequency-domain representations of signals, which is crucial for analyzing and manipulating audio data. I developed practical skills in MATLAB, including loading and playing audio files, and manually implementing the DFT algorithm to visualize the frequency spectrum of audio signals. This hands-on experience enabled me to identify and isolate noise components within the audio. Furthermore, designing and implementing various filters—low-pass, high-pass, and band-pass—deepened my understanding of frequency-selective filtering and its applications in noise reduction. Converting the filtered signals back to the time domain using the inverse DFT not only allowed me to listen to the effects of my filters but also to critically evaluate their effectiveness. Overall, this project provided valuable insights into digital signal processing, frequency analysis, and the practical implementation of audio filters.