LINEAR SYSTEMS
SECOND COURSE GSC/GSA/GTT/GT/IT

LABORATORY SESSION 4 - ACADEMIC YEAR 2023/2024

In this practice we will work with the Discrete Fourier Transform (DFT). We will use the popular Fast Fourier Transform (FFT) algorithm to calculate it. First, we will calculate the DFT of various signals and recover the original signals by means of the reverse DFT. Then, we will study the effect that extending the temporal signal with zeros has on the transform. Finally, we will discuss the circular convolution, either in its direct implementation or as a product of DFTs, and its relationship to the linear convolution.

1. The DFT and its implementation through the FFT

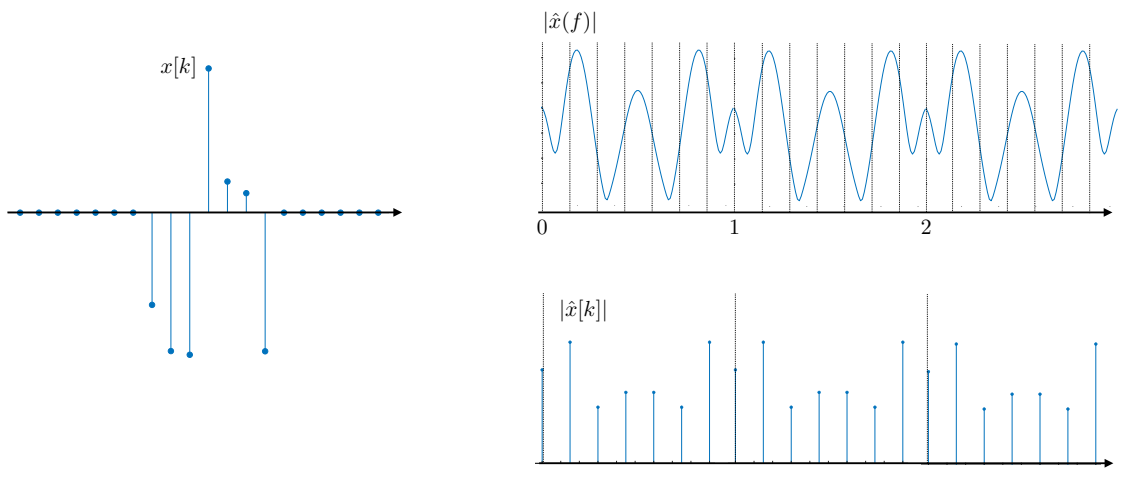
Goal: to understand the relationship between the Discrete-Time Fourier Transform (DTFT) and the DFT. Check the operation of the FFT algorithm and the Matlab function ***fft***.

The Fourier Transform of a discrete signal (i.e., the DTFT) is a continuous periodic signal with period 1. When considering the calculation of the DTFT computationally, it must be considered that Matlab only works with discrete signals. Therefore, although the Fourier Transform of a discrete signal is typically continuous, only discrete samples are obtained. Nevertheless, this can provide a good approximation if sufficient samples are taken. The resulting transform between a discrete-time signal and a discrete spectrum is called the *Discrete Fourier Transform (DFT)*.

The basic properties of DFT make it particularly convenient to analyze and design systems in the Fourier domain. Furthermore, there are efficient algorithms for calculating the DFT. Consequently, the DFT plays an important role in the analysis, design, and implementation of discrete signal processing systems and algorithms.

The DFT is equivalent to sampling the Fourier transform at uniform frequency intervals. That is, the calculation of the DFT of *N* points corresponds to calculating *N* samples of the Fourier Transform at frequencies $f_k = \frac{k}{N}$, $k = 0, \ldots, N-1$.

Example



There is a particularly efficient algorithm for calculating the DFT called the *Fast Fourier Transform*

(FFT). The number of mathematical operations employed in the calculations of an algorithm is sometimes used as a measure of its computational complexity. In the case of the FFT, its computational complexity is of the order of $N \log N$, where N is the number of points of the FFT. This is so low that, in many cases, the most efficient procedure for calculating a convolution between two discrete sequences is to calculate their respective FFTs, multiply them, and calculate the inverse FFT of the result.

Exercise 1: Direct DFT via FFT

The DFT is given by the following expression:

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \quad k = 0, \ldots, N-1$$

In this practice we will calculate the DFT by means of the FFT algorithm. To this end, we will use the ***fft*** function of Matlab. You just have to specify the signal you wish to transform and the number of points N. The Matlab function ***fft*** then calculates the transform for frequencies from 0 to 1. It is recommended to use the command ***fftshift*** to move and reorder the obtained values and be able to see the transform from –1/2 to +1/2. For more detailed instructions on the ***fft*** function, you can use Matlab's ***help*** functionality.

For a value of $N = 10$, calculate the FFT $\hat{x}[\cdot]$ for each of the following signals:

- $x[n] = \delta[n], n = 0, \ldots, 9$
- $x[n] = 1, n = 0, \ldots, 9$
- $x[n] = e^{j\frac{2\pi}{10}n}, n = 0, \ldots, 9$
- $x[n] = \cos\left(\frac{2\pi n}{10}\right), n = 0, \ldots, 9$

Draw the four signals in the same figure (Figure 1) using the ***subplot*** function. Plot the absolute values of the corresponding four DFTs in another figure (Figure 2). Modify the value of N, say, choose $N = 50$ and $N = 100$, and see how this affects the DFTs. Explain what you see.

Exercise 2: Reverse DFT using IFFT

To calculate the reverse DFT (or inverse DFT), make use of the ***ifft*** function of Matlab. Again, you can use the ***help*** functionality of Matlab to understand how the function works and consider using the ***ifftshift*** function. Use these functions to compute the reverse DFTs of the DFTs calculated in Exercise 1. For the value $N = 10$, plot the reverse DFTs and verify that the signals in the time domain correspond to the originals.

2. Zero Padding

One of the parameters that affects the behavior of the DFT is the number of points N with which it is calculated. If we have a sequence $x[\cdot]$ of length L ($L < N$), we can increase the length of the DFT by adding $N - L$ additional zeros. This is referred to as *zero padding.* If we interpret the DFT as a sampled version of the Fourier Transform of $x[\cdot]$ with N points (in each period), increasing the number of points of the DFT results in a finer sampling of the Fourier Transform.

Exercise 3: Inserting Zeros

To demonstrate the effects of zero padding, implement a Matlab function that does the following:

1. Define a signal $x[\cdot]$ of length N and store it in the form of a row vector ($1 \times N$).
2. Calculate the FFT of the signal and plot its absolute value.
3. Add a zero at the end of the sequence.
4. Repeat steps 2 and 3 and observe the effect of zero padding.

Additionally, you can check that the effect of zero padding is the same as that achieved by indicating the desired length to the function ***fft***.

```
% Code suggestion:
for n=N:2:4*N
    X = fftshift(fft(x));
    plot(abs(X),'b- .');
    axis tight;
    pause(0.2)
    x=[x 0];

end
```

3. Circular Convolution

The circular convolution of two sequence $x_1[\cdot]$ and $x_2[\cdot]$ can be computed as

$$x_3[n] = \sum_{m=0}^{N-1} x_1[m]x_2[(n-m)_N] = (x_1 \otimes x_2)[n], \quad n = 0, \ldots, N-1$$

Alternatively, one can first compute the DFTs of $x_1[\cdot]$ and $x_2[\cdot]$, multiply them sample by sample, and compute the inverse DFT of the result.

Exercise 4: Circular convolution

Implement a Matlab function that performs the circular convolution of two sequences. The format of the function is the following:

```
x3 = circ_conv(x1, x2, N, option)
% Circular Convolution
% x3 = Convolution of length N
% x1 = sequence of length <= N
% x2 = sequence of length<= N
% N = length of circular buffer
% Option
%     1 -> Convolution in time domain
%     2 -> Convolution in frequency domain
%           x3= ifft(fft(x1,N).*fft(x2,N))
...
```

Let $x_1[\cdot] = [1, 2, 2]$ and $x_2[\cdot] = [1, 2, 3, 4]$. Calculate the circular convolution for $N = 4$, $N = 5$, and $N = 6$ points. How does the result depend on N? Plot in one figure the signals $x_1[\cdot]$ and $x_2[\cdot]$ (using the function ***subplot***) and plot in another figure the three circular convolutions, calculated both directly in the time domain and indirectly by multiplying the signals' DFTs. Compare the resulting circular convolutions.

Exercise 5: Comparison with linear convolution

Now consider the sequences: $x_3[\cdot] = [1, 2, 2, 1]$ and $x_4[\cdot] = [1, -1, -1, 1]$. Calculate the value of N for which the circular convolution

$$x_5[n] = (x_3 \otimes x_4)[n], \quad n = 0, \ldots, N-1$$

is equal to the linear convolution

$$x_6[n] = (x_3 * x_4)[n], \quad n = 0, \ldots, N-1$$