LABORATORY 6
SAMPLING RATE CHANGE

6.1. Introduction

Many applications demand a change in sampling rate for the signal. Let $x[n]$ be a discrete signal obtained by sampling the continuous signal $x_c(t)$:

$$x[n] = x_c(nT) \tag{0.1}$$

where T represents the *sampling time*.

Changing the sampling rate for the digital signal $x[n]$ corresponds to acquiring a discrete sequence with the samples of the continuous signal taken with a period $T' \neq T$:

$$x'[n] = x_c(nT') \tag{0.2}$$

Usually, we will have at disposal only the discrete signal $x[n]$, not the continuous signal $x_c(t)$ and so, it will be necessary to modify the sampling rate only in discrete domain. This is possible through down-sampling (or decimation) or up-sampling (or interpolation), using appropriate circuits. Decimation and interpolation can be performed only with integer factors, and so, in order to achieve a rational change of the sampling rate it can be used a succession of decimation and interpolation circuits.

6.2. Decimation

Down-sampling represents a reduction of the sampling rate for a discrete-time signal with an integer factor M:

$$x_d[n] = x[nM] = x_c(nMT) \tag{0.3}$$

Let $F_{\max}$ be the maximum frequency of the continuous signal $x_c(t)$. In order to be possible to reconstruct the continuous signal $x_c(t)$ from the samples of the

decimated signal $x_d[n]$, it is necessary that the reduced sampling rate obeys the Nyquist law:

$$F_S' \geq 2F_{max} \tag{0.4}$$

where $F_S' = \frac{1}{T_S'} = \frac{1}{MT_S} = \frac{F_s}{M}$ is the reduced sampling rate by a factor *M*.

Figure 6.1 presents the amplitude spectrum of $x[n]$ in normalized angular frequencies and the amplitude-spectrum of $x_d[n]$, the decimated signal with the factor *M*.

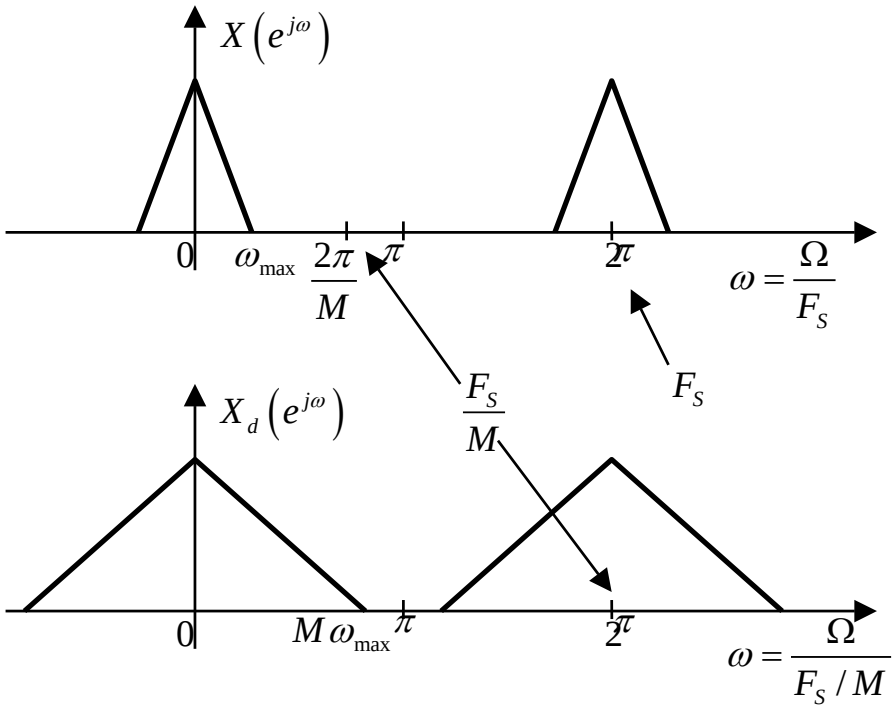


Figure 6.1. Signals spectra in normalized angular frequencies, before and after downsampling.

In case of not respecting the previous condition, after decimation the signal will be affected by aliasing (different spectrum periods overlap after decimation). In order to avoid aliasing, it is necessary to introduce a low-pass filtering before the elementary down-sampling circuit.

The normalized cutoff angular frequency for the low-pass filter is:

$$\omega_t = \frac{\pi}{M}. \tag{0.5}$$

Figure 6.2 shows the aliasing effect can be avoided by a low-pass filtering before the decimation.

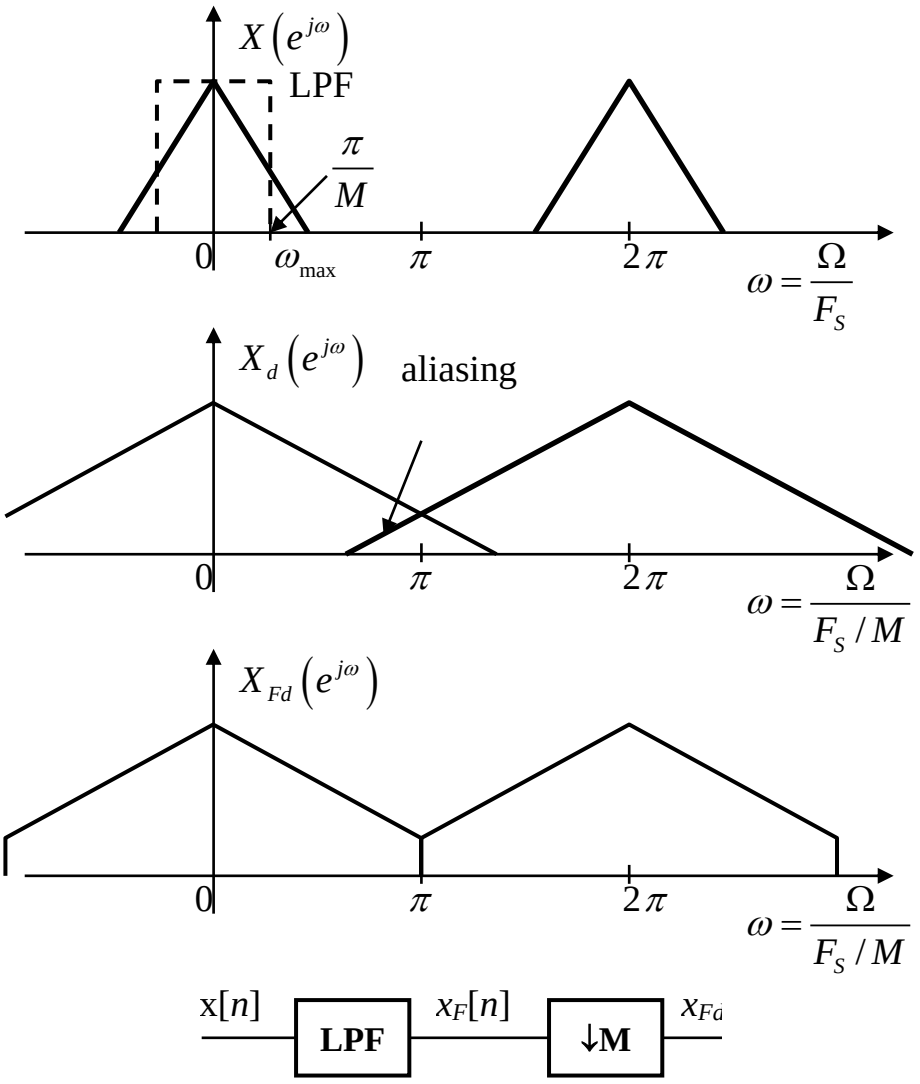


Figure 6.2. Spectral aliasing effect and the complete down-sampling circuit

E1. ***Exercise*****:**

a) Generate the following signal:

$$x[n] = \sin(2\pi f_1 n) + \sin(2\pi f_2 n), \qquad n = 0 : N-1 \tag{0.6}$$

where f_1 and f_2 are the normalized frequencies corresponding to the frequencies $F_1 = 1000Hz$, $F_2 = 3500Hz$ and the sampling frequency is $F_S = 20kHz$. The number of samples is $N = 64$.

b) Decimate this signal with $M = 2$ and $M = 4$. First, we reduce the sampling frequency using an elementary decimator which keeps the samples that are multiples of *M*:

$$xd[n] = x[nM]$$

This can be done in Matlab with:

```
xd = x(1:M:N);
```

c) Represent on the same figure, using subplot, the spectra calculated in $Nfft = 256$ points of the initial signal and of the decimated signal as function of the normalized frequency and in a second figure, the spectra as function of the un-normalized frequency.
How does the spectrum change by decimation? Fill in the table:

	$F_S' = \frac{F_S}{M}$	Normalized frequencies		Un-normalized frequencies	
		f_1'	f_2'	F_1' (Hz)	F_2' (Hz)
$M = 2$					
$M = 4$					

In order to avoid aliasing (observed for frequency F_2' if $M = 4$), the input signal must be filtered with a low-pass filter with the normalized cut-off angular frequency $\omega_t = \pi / M$, according to the **complete decimator circuit** (see figure 6.2).

d) Design using MATLAB the antialiasing filter as a **low-pass IIR filter** with the normalized cut-off angular frequency $\omega_t = \pi / M$. The filter must have an attenuation of minimum 30 dB in the stop band. Filter the input signal using the designed filter.

e) Decimate the filtered signal with $M = 2$ and $M = 4$. Represent the spectrum of the filtered signal and of the signal after filtering and decimation. Comment the differences with respect to the previously obtained results.

The Matlab function **`decimate`** implements a decimator, meaning that it filters with a low-pass filter the given signal, then it decimates it with the specified rate.

***Syntax*:**

`y = decimate(x,D)`

- the vector `x` contains the values of the samples of the input sequence $x[n]$ and `D` is the decimation factor. In order to avoid aliasing, use implicitly a low-pass filter of type Chebyshev I and order 8. The vector `y` results, which contains the values of the samples of the output signal $y[m]$.
- the length of the vector `x` must be at least 3 times higher than the order of the filter used to avoid aliasing. In this syntax, a filter with order 8 is used implicitly, so the minimum length of x is 25.

`y = decimate(x,D,n)`

- the same significance as in the previous syntax, except that a Chebyshev I filter of order `n` is used to avoid aliasing. In this case the vector `x` must have a length higher than `3n`. It is not recommended to choose an order higher than 13 due to the numerical instability (MATLAB outputs a warning in this case).

`y = decimate(x,D,'fir')`

- the same significance as in the previous syntax, except that a finite response filter (*FIR*) of length 30 is used to avoid aliasing.

`y = decimate(x,D,n,'fir')`

- the same significance as in the previous syntax, except that a finite response filter (*FIR*) of length `n` is used to avoid aliasing.

6.3. Interpolation

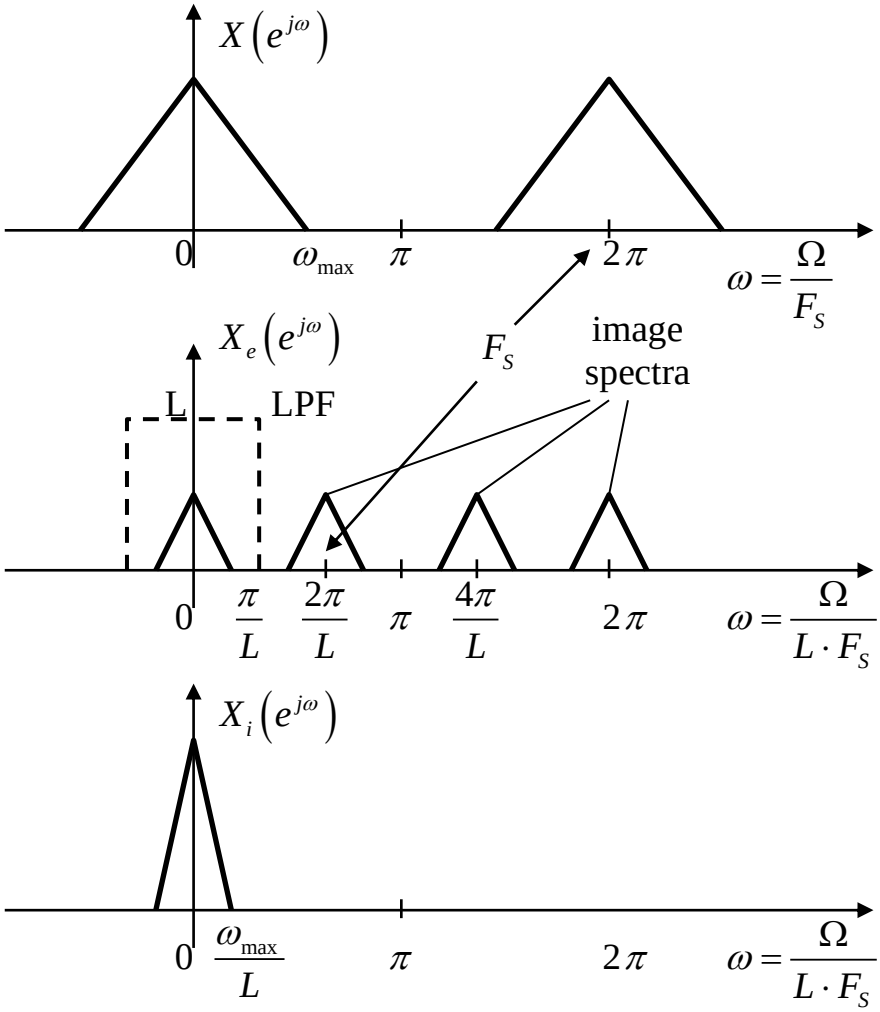
By up-sampling with a factor *L*, we increase the sampling rate with that factor. First the signal is expanded in discrete-time and then a low-pass filtering follows.

The basic method for expanding a signal is by introducing *L-1* zeros in between consecutive samples of the original signal $x[n]$:

$$x_e[n] = \begin{cases} x[n/L], & n = 0,\ \pm L,\ \pm 2L, \ldots \\ 0 & \text{în rest} \end{cases}, \tag{0.7}$$

After expanding, the low-pass filtering should reject the image spectra that appear in the base period range $\omega \in [-\pi, \pi]$. The image spectra are a result of increasing the sampling rate *L* times. The normalized angular cutoff frequency for the filter is:

$$\omega_t = \frac{\pi}{L}. \tag{0.8}$$



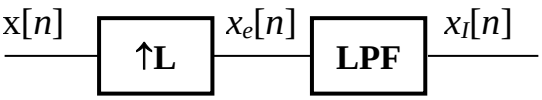


Figure 6.3. The complete interpolation circuit by a factor L.

Also, the filter gain should be equal to L, so that we have the following equality between input and output samples – $x[n]$ and $x_i[n]$:

$$x_i[m] = x[m/L], \text{ for } m = 0, \pm L \ldots \tag{0.9}$$

E2. ***Exercise*:**

a) Generate the following signal:

$$x[n] = \sin(0.4\pi n) \qquad n = 0 : N-1 \tag{0.10}$$

where $N = 64$.

b) Increase the sampling rate by up-sampling with $L = 3$.

$$x_e[n] = \begin{cases} x[n/L] & \text{pentru} \quad n = kL, \quad k \in \mathbf{Z} \\ 0 & \text{în rest} \end{cases}$$

This is done in Matlab using:

```
xe = zeros(1,L*N);
xe(1:L:L*N) = x;
```

c) Represent in figure 1 using subplot portions of the 2 signals:

```
figure(1)
subplot(311),stem(0:10,x(1:11)),grid;
subplot(312),stem(0:L*10,xe(1:L*10+1)),grid;
```

d) In figure 2 represent the spectra calculated in $Nfft = 256$ points of the initial signal and of the up-sampled signal as function of the normalized frequencies.

How does the spectrum change by up-sampling? Explain the occurence of the supplementary spectral components following the up-sampling.

The complete structure of the interpolation circuit is obtained by adding, after up-sampling, a low-pass filter with the normalized cut-off angular frequency $\omega_t = \pi / L$ and gain equal to L, which eliminates the image spectra (see figure 6.3). The effect of this time domain filtering operation is to restore the samples of the signal in the points where zeros were introduced, by obtaining a digital signal with a spectrum that is identical to the one of the analog signal.

e) **Design a low-pass FIR filter** of order 30 with the normalized cut-off angular frequency $\omega_t = \pi / L$ and gain equal to L. **Filter the up-sampled signal**.

f) Represent the signal **xi** obtained after the filtering in figure 1 (together with the signals **x** and **xe)** and the spectrum of this signal in figure 2 (together with the other spectra):

```
figure(1),
subplot(313),stem(0:L*10,xi(16:L*10+16)),grid;
```

g) Same exercise for $L = 6$.

The Matlab function `interp` implementes an interpolator, meaning that it up-samples the given signal with the specified rate, then it low-pass filters it.

***Syntax*:**

`y = interp(x,U)`

- the vector `x` contains the values of the samples of the input sequence $x[n]$ and `U` is the interpolation factor. The vector `y` results, which contains the values of the samples of the output signal $y[m]$.
- an anti-imaging filter of length 4 with the normalized cut-off frequency 0.5 is used implicitly. The length of `x` must be at least 2(length of the filter)+1. In this syntax a filter of length 4 is used implicitly, so the minimum length of `x` is 9.

`y = interp(x,U,l,ft)`

- the same significance as in the previous syntax, except that an anti-imaging filter of length `l` is used, having the normalized cut-off frequency `ft`. In this case the vector `x` must have the length at least `2l`+1.

`[y,b] = interp(x,U,l,ft)`

- `y`, `x`, `U`, `l`, `ft` have the same significance as in the previous syntax; the vector `b` is also returned, which contains the coefficients of the anti-imaging filter.

6.4. Sampling rate conversion by a rational factor L/M

We now consider the general case of sampling rate conversion by a rational factor *L*/*M*, since many practical applications require such a change. It is possible to achieve this sampling rate conversion by first performing interpolation with a factor *L* and then decimating the output with a factor *M*. These steps are realized by cascading an interpolator with a decimator, as it is presented in figure 6.4:

Figure 6.4. Circuit for sampling rate conversion by a rational factor.

It is important to perform the interpolation first and the decimation second, in order to preserve the desired spectral characteristics of $x[n]$. The low-pass filter in this circuit must incorporate the filtering operations necessary for both steps: interpolation and decimation. The frequency response of the filter is $H(e^{j\omega})$:

$$H(e^{j\omega}) = \begin{cases} L, & 0 \le |\omega| \le \min\left(\frac{\pi}{M}, \frac{\pi}{L}\right) \\ 0, & \text{in rest} \end{cases} \qquad (0.11)$$

E3. ***Exercise*:**
Start from a sine wave of $F_1 = 8kHz$, sampled with a sampling rate of $F_S = 32kHz$. The signal's length is $N = 256$. We want to convert the sampling rate to:

1. $F_{S1} = 48kHz$.
2. $F_{S2} = 20kHz$.
3. $F_{S3} = 12kHz$.

a) Establish the interpolation factor *L*, and the decimation factor *M* for each case.
b) Design the low-pass filter in the circuit for sampling rate conversion by a rational factor: establish the gain and the filter's cutoff frequency for each case.

The Matlab function **`resample`** implements the sampling rate conversion by a rational factor.

***Syntax*:**

`y = resample(x,U,D)`

- The vector `x` has the input signal's samples $x[n]$, `U` is the up-sampling factor, and `D` is the decimation factor needed to achieve a rational value U/D. The function output is a vector `y` that contains the output signal's samples $y[m]$;
- By default the procedure uses a low-pass filter implemented as a FIR filter (using `fir1` and a Kaiser window with the parameter `beta = 5`).

`[y,b] = resample(x,U,D)`

- This syntax has one more output value: the vector `b` that has the impulse response coefficients of the filter used by default.

`y = resample(x,U,D,b)`

- `y`, `x`, `U` and `D` have the same significance.
- the vector `b` that has the desired impulse response coefficients, used instead of the default filter.