INTRODUCTION TO UPSAMPLING & DOWNSAMPLING

What is Sampling Rate Conversion?

As the name suggests, the process of converting the sampling rate of a digital signal from one rate to another is Sampling Rate Conversion.

Increasing the rate of already sampled signal is Upsampling whereas decreasing the rate is called downsampling.

Why to do it?

Many practical applications require to transmit and receive digital signals with different sampling rates. So at many stages in application we need to do sampling rate conversion.

How to do it?

There are two ways in which we can achieve sampling rate conversion:

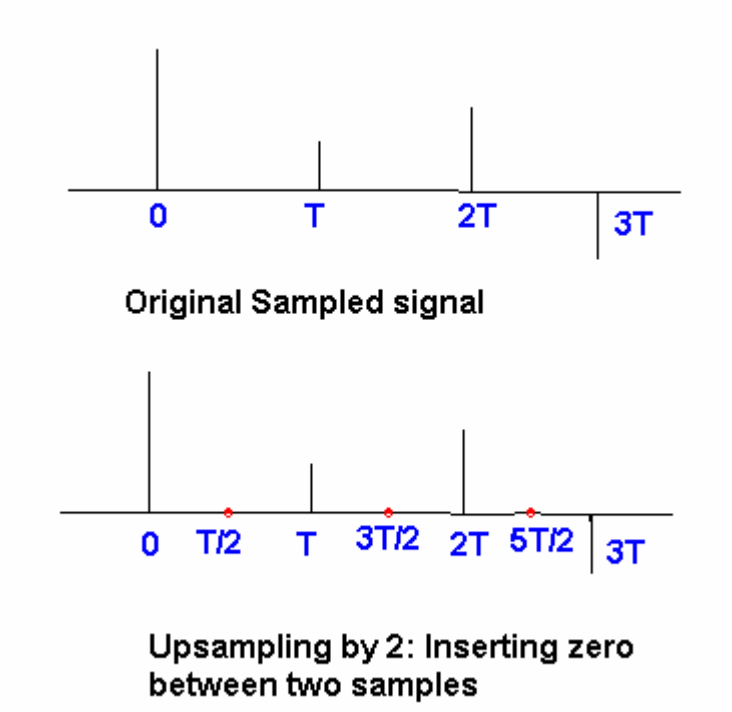
1. First approach is to do D/A conversion to recover back original analog signal. Then we can do A/D conversion with desired sampling rate. The advantage of this technique is that the second sampling rate need not hold any special relationship with old one. But the problem is signal distortion introduced by D/A and quantization effects of A/D.

2. Second approach is to work in digital domain only. So we'll have to predict the digital signal with desired rate using the already sampled signal in hand.

Now for this lecture, we'll look at the second choice of sampling rate conversion.

UPSAMPLING

Let's consider, simplest case of upsampling. We want to double the sampling rate of signal. So what we do is insert 0s in between two successive samples. As shown:

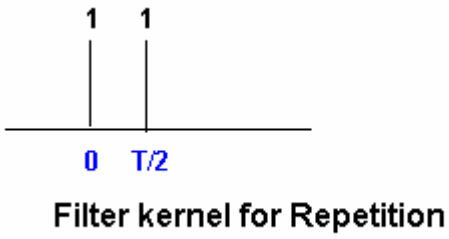


Obviously this is a bad approach. As we don't have data for intermediate samples, let's generate it.

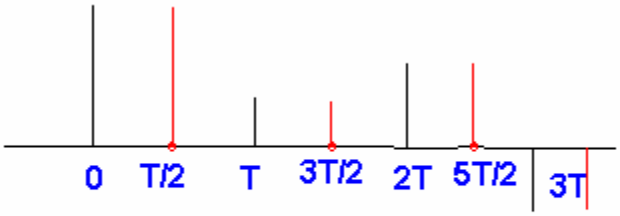
Method-1: Repetition

Repeat the current sample.

The corresponding filter kernel will be



The output waveform will be:

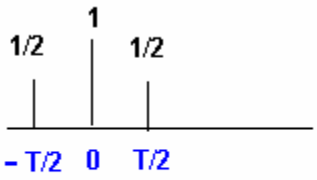


Upsampled by 2 using repetition

Method-2: Interpolation

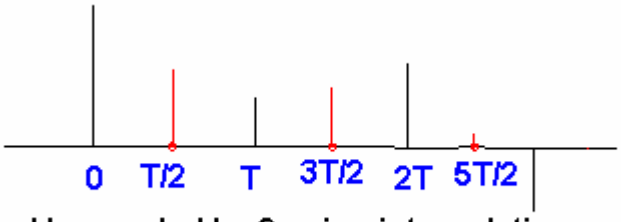
The intermediate sample value will be the average of its neighboring values.

The filter kernel will be:



Filter Kernel for interpolation

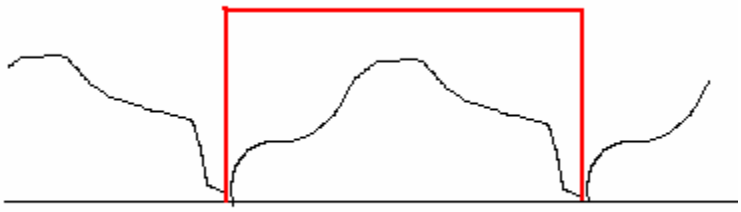
The output waveform will be:



Upsampled by 2 using interpolation

Now this a very good method but it produces, two aliases of each frequency in frequency domain . So we should cut the high frequency contents to avoid aliasing.

Why to cut?

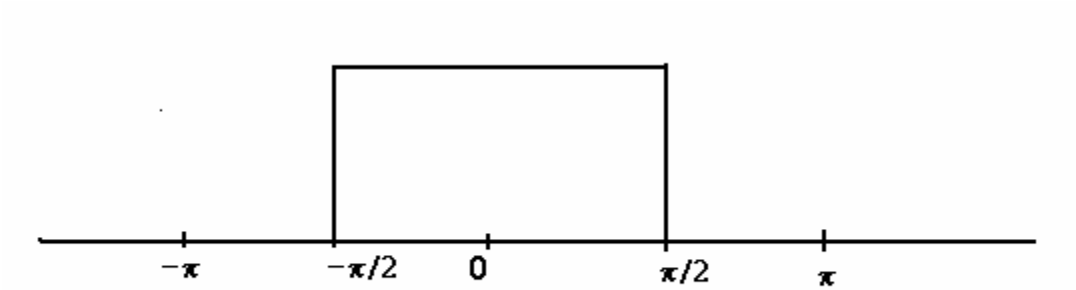


Use of rectangular window

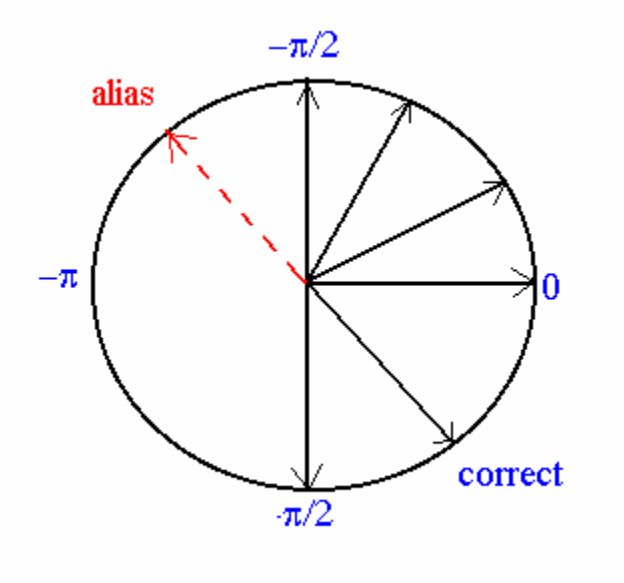
Because they don't contain any new information. They are just repeating information. We also know that maximum frequency content in original signal cannot be greater than **Fs/2** so there is definitely no more information in baseband. So we can afford to cut high frequency contents.

How to cut it?

Use a perfect low pass filter. That is we will keep slow moving spirals and reject fast moving spirals in frequency domain.



Low pass filter selecting frequecies between $-\pi/2$ to $\pi/2$



so filter kernel in frequency domain is as shown. Its amplitude is 1 for frequencies in the range - π/2 to +π/2 and zero for rest all frequencies. So we cut the high frequency aliases. So filter kernel in frequency domain is set of slow moving spirals having amplitude 1.

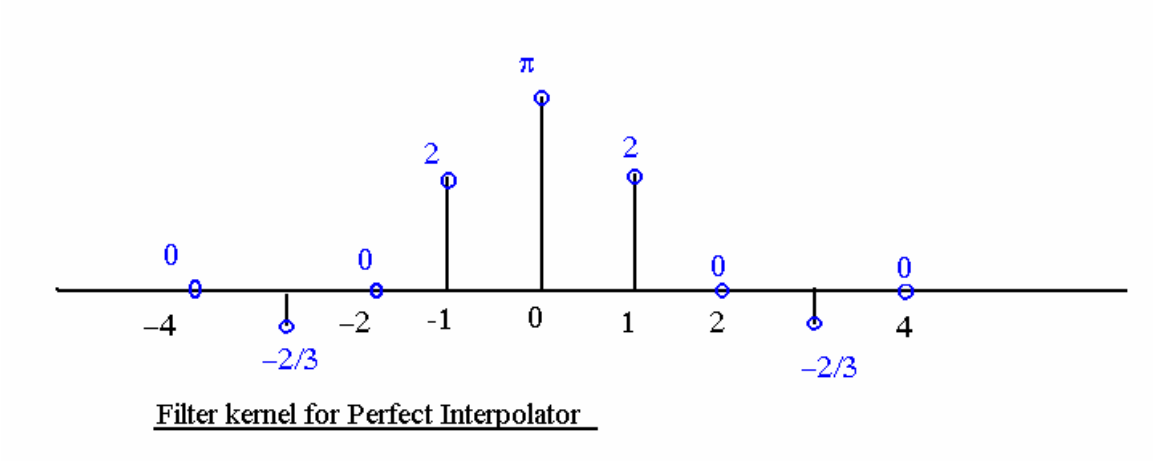
Let's figure out, what will be the corresponding filter kernel time domain.

$$\int_{-\pi/2}^{\pi/2} e^{0}\, d\omega = \pi/2 - (-\pi/2) = \pi$$

$$\int_{-\pi/2}^{\pi/2} e^{j\omega}\, d\omega = 1/j(e^{j\pi/2} - e^{-j\pi/2}) = 2$$

$$\int_{-\pi/2}^{\pi/2} e^{2j\omega}\, d\omega = 0$$

$$\int_{-\pi/2}^{\pi/2} e^{3j\omega}\, d\omega = -2/3$$



Filter kernel for Perfect Interpolator

Thus the perfect interpolator is an Infinite Impulse Response (IIR) filter. The filter is not causal hence cannot model using ARMA. We cannot implement the filter because it's infinite.

There is a class of functions called Analytic functions. According to Taylor, all the information in the analytical function is at zero, so you don't have to go far. We can express these functions as Lauriant series and model them. But for this, the function should be continuous at all points. But our perfect interpolator filter kernel is discontinuous at - $\pi/2$ and $\pi/2$.

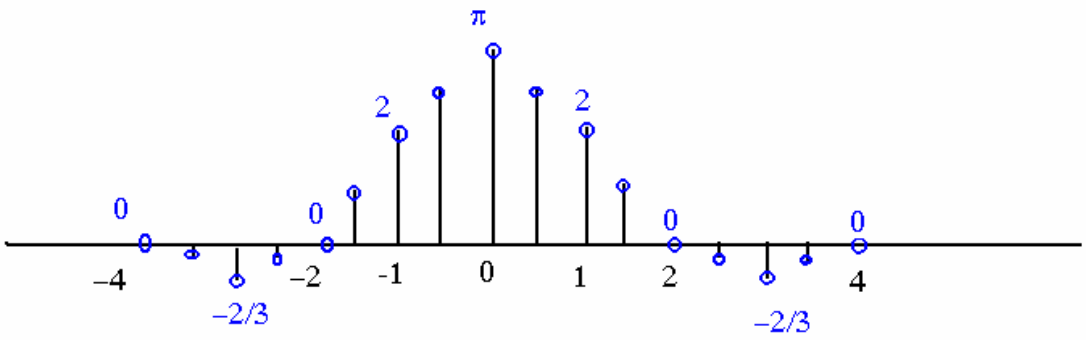
One interesting observation is that, we are getting zeros at previously sampled points in the filter kernel.

So to do upsampling faster we can use Multi-channel Polyphase Multi-sampler with two filter banks.

1. 1^{st} filter bank doing nothing (corresponding to already sampled points).
2. 2^{nd} filter bank with half sample delay.

UPSAMPLING BY 4

Here the filter kernel will look like this:



Filter kernel for Perfect Interpolator for upsampling by factor 4

In this case, we'll have to predict three new samples, between already present pair of samples. We now will have 4-filter banks.

1. Bank with 0 sample delay.
2. Bank with 1/4 sample delay.
3. Bank with 1/2 sample delay.
4. Bank with 3/4 sample delay.

PRACTICAL DESIGN OF FILTER:

Use of Linear Phase Filters:

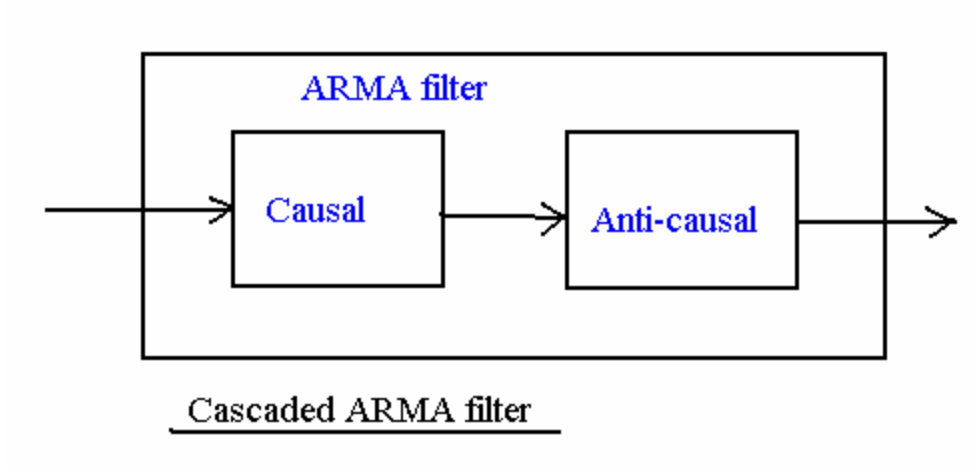
For practical design of upsampler filters, we use delayed symmetric filters.

Zero phase filters are symmetric about zero. Delayed symmetric filters have symmetry along a shifted point. They are also called as Linear phase filters. The advantage of this approach is the filtering can happen at real time. But disadvantage is that the output is not sharp.

Use of ARMA filters:

Using ARMA filters, we can get sharper output. But we cannot get real time processing using ARMA filters.

For offline processing, we can use two ARMA filters, one Causal and second Anti-causal filter. Then we just add the outputs and we'll have desired result. The filter kernel of Causal and Anti-causal filters will be just the flips of one another.

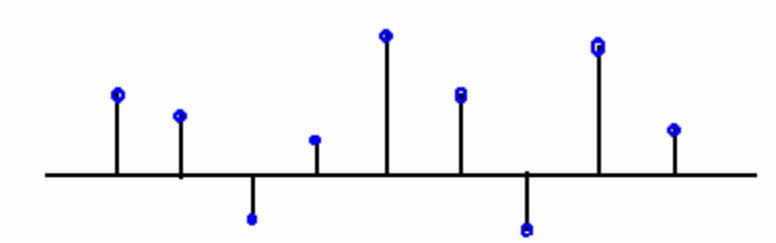


Cascaded ARMA filter

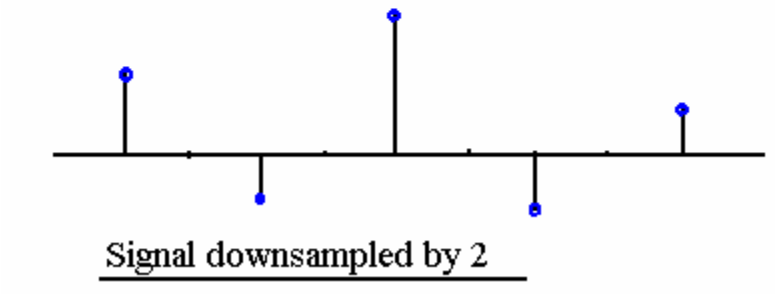
DOWNSAMPLING:

As said, Downsampling is decreasing the sampling rate of a signal. Let's consider a simple case of downsampling a signal to half of its original sampling rate.

Simplest way to do this is to forget every other sample and we'll have the desired sampling rate.

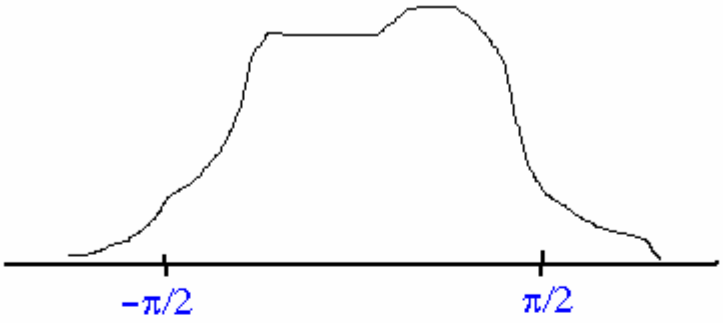


Original Signal

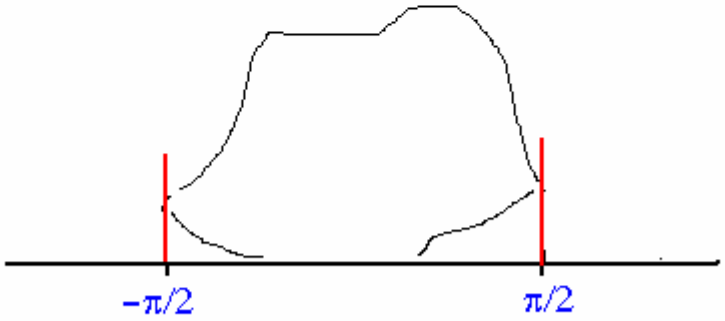


Signal downsampled by 2

But if we reduce the sampling rate just by selecting every other sample of x(n), the resulting with folding frequency Fs/2 . The frequency spectrum tries to spread up but it cannot do so. Hence it winds up on itself.



Original Spectrum



Spectrum wound up on itself

To avoid aliasing, we must first reduce the bandwidth of x(n) to $\omega_{max} = \pi/2$. So we should cut out high frequency contents to avoid aliasing.