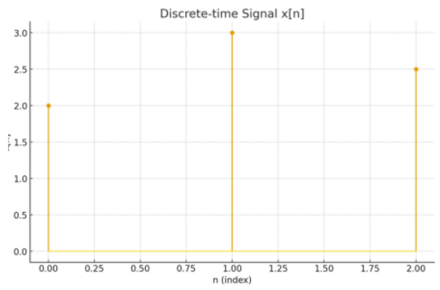
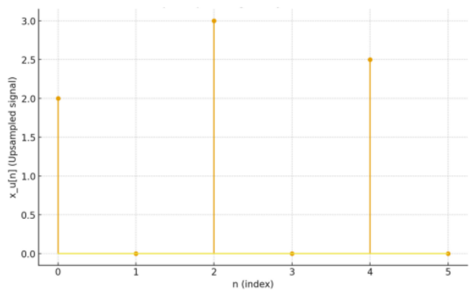
Signal Processing Algorithms & Architecture
Dr. Anirban Dasgupta
Department of Electronics & Electrical Engineering
Indian Institute of Technology Guwahati

Lec 15: Multi-rate Signal Processing - II

Hello everyone, welcome to a fresh new lecture on the topic of multirate signal processing 2, which is a continuation of the topic of multirate signal processing 1. I am Dr. Anirban Dasgupta and let us get started. So, we will start today's topic with half-band filters. So, what are half-band filters? So, when interpolating a signal, the interpolation filter will, in general, alter the samples of the signal $x[n]$ in addition to filling the zeros. Sounds complicated? Let us see.



So suppose we have a signal; say this value is 2, say this value is 3 at index 1, and say this value is 2.5 at index 2. Now this is my original signal $x[n]$. Now what we will do is first upsample the signal, say by a factor of 2.



What does it mean? So, I will insert a 0 in between. So, my zeroth index is 2. My second index will be 3, my index 4 will be 2.5, and I have two more values, which are zeros at index 1 and 3, and this is upsampling. Now we need to interpolate so that these zeros are removed, and for that, we will use a low-pass filter, which I have already discussed in the last class.

So, let me use a separate color; say, let me use this green color. So, what this interpolation filter will do is create something like this, probably, and these are the new interpolated values. Now, you can see that although this is a smoother curve, the green curve, these original values of xn are distorted. In the new interpolated signal, my resulting values are quite different from the original values. So, the question is, can I design a filter h of n such that, let me use another color, maybe blue, such that my original values of the signal should be intact, and the other values can be anything; it does not really matter.

Something like this. So, I get a smoother version, I get a low-passed version, but this interpolated signal, my original signal values that are 2, 3, and 2.5, are intact. So, such a filter which will keep these original signal values intact after interpolation is called a half-band filter. So, this is the question: can I design an interpolation filter that can preserve the original samples x[n]? So, when I am doing this upsampling by 2, what should the condition be? The condition is

$$y[n] = h[n] * [\uparrow 2]x[n]$$

that is even after the interpolation.

Of course, after up-sampling, the samples will remain at the same levels, or in general,

$$y[2n + n_o] = x[n]$$

Why n_0 ? Because if this is a zero-centered filter, it will not be a causal filter. So, to make it casual, I will give a shift n_o. So let us see what this filter looks like.

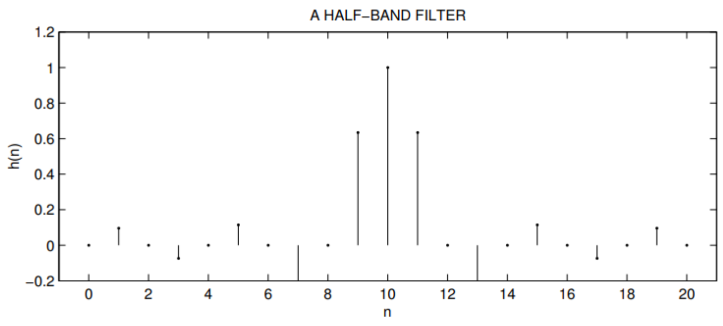
So this filter h[n] will be a half-band filter if it will not change the samples of x[n] or if it is centered at n_o. The criteria should be that h[n] should be

$$h[n] = \begin{cases} 1, & \text{for } n = n_0 \\ 0, & \text{for } n = n_0 \pm 2,4,6,\ldots \end{cases}$$

Now this is very much similar to the δ function, and this is the relation that

$$h[2n + n_0] = \delta[n]$$

This is just for the even values of n however; you are free to choose any value of h[n] for the odd indices. Let us take an example. Here $n_0 = 10$

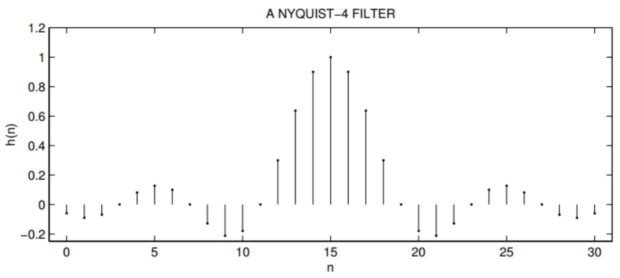


That is, it is centered at 10. By the structure, it looks like a sink filter. Sink means it will have a box-like structure in the Fourier domain. So, if this is my filter in the time domain. So, it will be this is in time, so in frequency it will be a box filter, kind of an ideal. However, because of this finite length, we have truncated it and this is a half-band filter; why? Because if you see, this has a value of 1 at n_0 , and the other even indices, which are n_0+2, n_0 + 4, n_0 + 6, and n_0 + 8, or if you look on this side, $n_0$is $-2, -4, -6$, and -8, these are all zeros and the remaining ones are kind of looking like a sink and again, if it is zero-centered, this is the condition; although this will not be a causal filter, it will depend on the future values of the input, and for that, you cannot go for real-time processing; you have to do offline processing.

Okay, so if the idea for a factor of 2 is clear, let us extend this for a factor L. When we extend this for a factor of L, such kinds of half-band filters are called Nyquist L filters.

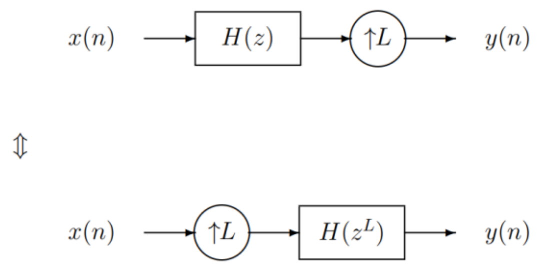
$$h[Ln] = \delta[n]; \text{ L is the interpolation factor}$$

This is the general criterion or the general condition for a zero-centered Nyquist L filter that $h[Ln] = \delta[n]$, where L is the factor by which we are up-sampling, equals to δ(n). If you look at this graph,



This is a Nyquist 4 filter that is centered at $n = 15$. So at n=15, you see the value is 1, and since this is a Nyquist 4 filter, after every 4 values, you will get a 0: like 1, 2, 3, then the fourth value is a 0.

So, what comes next after Nyquist filters? We will come up with some interesting identities. What do you mean by identities? So, these are similar in both ways, like we have the convolution multiplication identity. So, in the time domain, when we do convolution in frequency, we get multiplication and vice versa. So, these are called noble identities; I do not know why they are called noble. But let us try to understand why these identities were created in the first place.

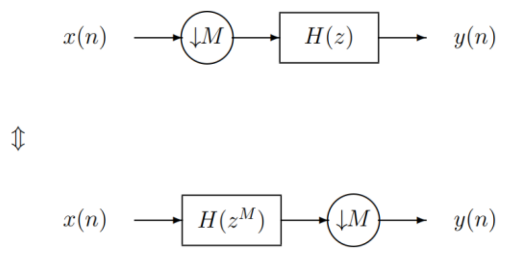


$$\boxed{[\uparrow L]\ (h(n) * x(n)) = [\uparrow L]\, h(n) * [\uparrow L]\, x(n)}$$

So, if you see this structure, what is it? This is an interpolator. So we are first up sample by a factor of L, and then we use the filter to remove the spectral images. Of course, if you are feeling this difficulty, you should go and watch the previous lecture, Multirate Signal Processing-I, because this is a continuation of that. So, what is the problem with this? The problem is that this filter operates at a higher sampling rate. So, if my input signal was, say, sampled at 100 Hertz, and if, say, the value of L is 5, then this filter will operate at 500 Hertz.

Now if a filter is operating at a higher sampling rate, then the cost of the filter will be higher because you need costlier components. But there is one more factor that I will discuss soon; for now, consider that this filter will be costlier because I have to work on it at a higher sampling rate. So, can I somehow bring this to a lower sampling rate? Well, the only way to bring it to the left of this upsampler is to place it before it. But will it be the same filter? Definitely not because now the sampling rate will be lower, L4 less. So, here if you see that if this filter response is $H(Z^L)$, then this filter in the noble identity will be just H(Z), which effectively means that if I do the convolution first and then upsample, this is equivalent to individually upsampling both the filter and the signal and then doing the convolution.

There is also a similar identity for the downsampler.



$$\boxed{h(n) * [\downarrow M]\, x(n) = [\downarrow M]\ ([\uparrow M]\, h(n) * x(n))}$$

So here, if you see, this is our down sampler; we have a pre-aliasing or anti-aliasing filter. So, this is again operating at a higher rate, M times the higher sampling rate. So, if we bring this after the downsampler, definitely the sampling rate will be reduced and here we will see that this has H(z) if the previous sampling rate was $H(z^M)$, not the sampling rate but the impulse response, the pulse transfer function of the system.

So, this is equivalent to convolving h[n], which in this case is a downsampled version of my signal, and is equivalent to downsampling the convolution of the signal with the upsampled version of the filter. This is an upsampled version of the filter. So, we are first convolving with the upsampled version of the filter and then we are downsampling it. Now these identities will be useful in the subsequent concepts. Which is polyphase decomposition.

Now polyphase decomposition is an interesting concept, and we will use this polyphase decomposition and the noble identities to create a very new and efficient way of implementing a filter. So, the first step is that I will decompose this x[n] into two sets.

$$x_0[n] = x[2n]$$
$$x_1[n] = x[2n+1]$$

Even odd samples. Does this sound familiar? So, we did something similar in the FFT derivation. So now we have X0 of n, which are even samples of $x_0[n]$, and $x_1[n]$, which are the odd samples of x[n].

Now, if you do the Z transform, we see that X(Z) will be

$$\boxed{X(z) = X_0(z^2) + z^{-1}\, X_1(z^2)}$$

Why is $X_0(Z^2)$ not Z^2 ? Because X is not operating at how much? Twice or half of the operating frequency. It is operating at half the sampling frequency of x at 2n. So, if we do the reverse, in terms of x, x will be $x[0]z^2$, and this will also be $x[1]z^2$, and this will appear as a delay. So, you just change the sides and see; this will be the relationship.

Let us take an example to clarify the concept. So, this is our signal and this is my 0 index. So, let us take the odd indices or odd values of the signal first to form $x_0[n]$, which are 3, 5, 2, and -3 and similarly, $X_1(Z)$ will be formed by 1, 6, 4, and 7.

$$x[n] = \{\underline{3},\ 1,\ 5,\ 6,\ 2,\ 4,\ -3,\ 7\}$$

$$x_0[n] = \{\underline{3},\ 5,\ 2,\ -3\}$$

$$x_1[n] = \{\underline{1},\ 6,\ 4,\ 7\}.$$

The Z-transforms for this example are given by

$$X(z) = 3 + z^{-1} + 5\,z^{-2} + 6\,z^{-3} + 2\,z^{-4} + 4\,z^{-5} - 3\,z^{-6} + 7\,z^{-7}$$

$$X_0(z) = 3 + 5\,z^{-1} + 2\,z^{-2} - 3\,z^{-3}$$

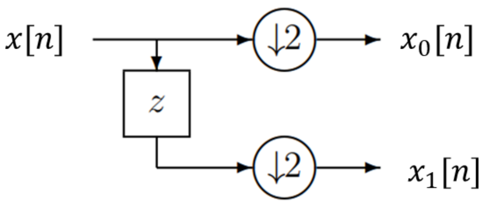
$$X_1(z) = 1 + 6\,z^{-1} + 4\,z^{-2} + 7\,z^{-3}.$$

This is simple. Now, if I do the Z-transform of the original signal, this is what I get from this, and this is just if I plug in the equation of the Z-transform. If I get the Z-transform of $x_0(n)$, I will get this. And for $x_1(n)$, I will get this. So, you can verify the relation that I just explained in the previous slide, this one. So, just plug in, and you will check that this is verified.

So, this is the block diagram in the time domain where we are downsampling the signal x[n] by 2 and also a shifted version of the signal. This is typically an advance; this is Z. If you take Z inverse, it will be a delayed version, and I am downsampling by 2, and I get $x_1(n)$.

$$x_0[n] = x[2n]$$

$$x_1[n] = x[2n+1]$$



These are the two signals that I have already explained. So, we will extend this concept from a factor of 2 to a factor of m.

$$x_0[n] = x[Mn]$$

$$x_1[n] = x[Mn+1]$$

$$\vdots$$

$$x_{M-1}[n] = x[Mn+M-1]$$

The Z-transform $X(z)$ is then given by

$$\boxed{X(z) = X_0(z^M) + z^{-1}\,X_1(z^M) + \cdots + z^{-(M-1)}\,X_{M-1}(z^M)}$$

What happens here is that I see I get m such components x_0 to x_{M-1}. Like if I have, say, 1, 2, 3, 4, 5, 6, and I want three phases, so 1 and 4 will be my x_0 as 0, 2 and 5 will be my x_1, and 3 and 6 will be my x_2 . This is also very similar to the divide and conquer FFT. So here I have three components. Now I will divide them similarly in this way.

So, this overall X(Z) is divided into smaller z transforms. So, the polyphase is an inherent implementation or application of downsampling. Let us try to understand an inherent implementation of up-sampling or interpolation and here we will use the noble identities as well as the polyphase decomposition to obtain an efficient structure and this is what I was talking about: what is the use of changing the order of the upsampler and the filter? So, this up-sampler is followed by a filter in an interpolation system; this you know, okay.

What does the upsampler do? It is inserting zeros between samples. If I have a sample, say of size 2, length 2, or amplitude 2, which is 3, then this is 5. So upsamplers, if I upsample by 3, I am inserting two zeros here. If I upsample by L, I insert $L-1$ zeros. Now, after that, if I am doing the filtering h[n], what am I doing? I am convolving this with the upsampled sequence, which is x[n] by 3, you could say.

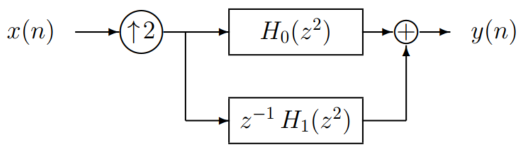
Now technically you know what the complexity of convolution is, but here I am unnecessarily multiplying a lot of samples by zeros and doing a lot of addition with zeros, which is technically useless and baseless. Although these are required, we are wasting some resources; that is, in other words, what I said: this is operating at a higher sampling rate, but this high sampling rate mainly consists of a lot of samples that are zeros. Most of the samples to the input are zero, and the filter is operating at a higher rate. So, can we do better? And that brings us to the concept of efficient implementation. So, we can use the polyphase representation and then apply the noble identities.

For example, this is my signal-

$$\begin{tikzcd} x(n) \arrow[r] & \uparrow 2 \arrow[r] & \boxed{H(z)} \arrow[r] & y(n) \end{tikzcd}$$

$$\begin{tikzcd} x(n) \arrow[r] & \uparrow 2 \arrow[r] & \boxed{H_0(z^2)+z^{-1}H_1(z^2)} \arrow[r] & y(n) \end{tikzcd}$$

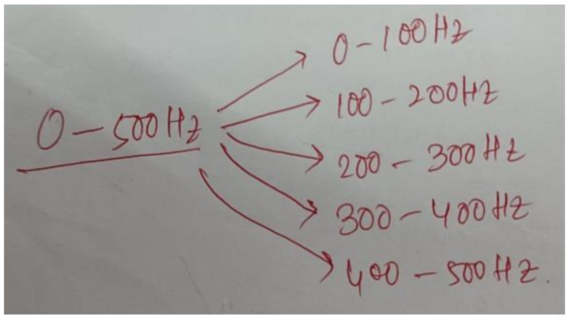
I am up-sample by a factor of 2, and then I do the filtering to remove the spectral copies to get my output y[n]. Again, this H(z) can be written as a polyphase decomposition and here, since the up-sampling is by a factor of 2, I will just do two terms H_0 and H_1. Next, what I do, since this is addition, is to say that this is a parallel structure with $H_0$and H_1. And why is it parallel? Because this is being decimated over time.



So, this is half of the components, which are the even samples, and these are odd samples, or whatever. These are the odd samples; these are the even samples. and these two get parallelly converged. So, this is once doing a parallel computation, so it will be faster.
That is fine. Now, what is the problem? This is operating at twice the sampling rate of course, the same thing is happening here, but how can we solve this issue? Let us see. So now if I just bring this up sampler to each block like this, I have brought the up sampler before each block separately. And why did I? Again, in this step, I just cascaded this because this is a product in z, so I can cascade the z inverse here. And now with this, I can implement the noble identities, which say that if I can bring this h before, and this h is a function of Z^2, if it comes before the half sampler, it becomes a function H(Z), and the same is true here.

This was the whole purpose, and now we see that H_0 and H_1 are operating at a lower sampling rate, which is a sampling rate of x[n], and this is an efficient implementation of interpolation. Now, extending this concept, we can form something called digital filter banks, and these are immensely useful in many cases, like even in the construction of discrete wavelet transforms. So, what is a digital filter bank? It is a collection of band-pass filters that divide the signal into multiple components. And each component will carry a different frequency band.

If you feel this is complex, say I have a filter that operates at 0 to 500 hertz, and since this has a high bandwidth, what I can do is reduce this to 5 filters, each operating at a 100

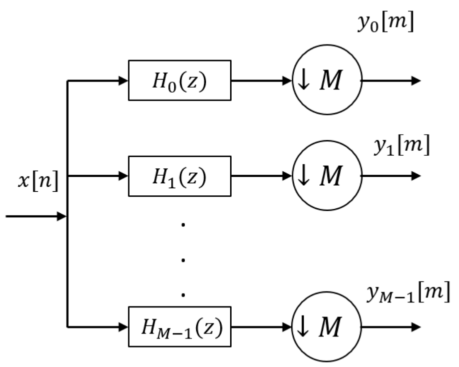


hertz bandwidth. So, this takes care of 0 to 100 Hertz, this is 100 to 200 Hertz, this is 200 to 300 Hertz, and this is 300 to 400 Hertz. This is a 400 to 500 Hertz kind of divide-and-conquer approach, you could say.

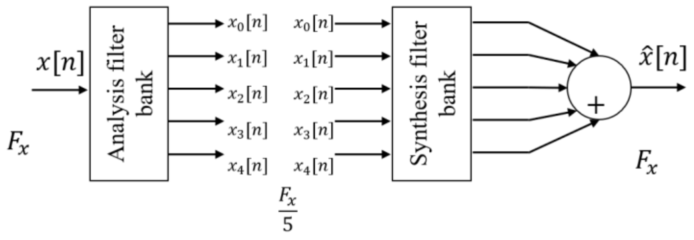
So, I am splitting the work into a lot of smaller tasks, and each filter is responsible for a specific bandwidth. This is like a prism; white light is coming to the prism, heating and splitting into different components, each responsible for a specific color. Now this is a structure where I have this X_n, and there are m such filters, each responsible for a small bandwidth.

So, if it is a digital domain, and if 0 to π is my operating digital frequency, you can see that my overall filter, instead of working from 0 to π, can have sections that are π/M. Each filter will operate in this bandwidth of 0 to π/M. So, this is what you get. So, this is an example of a filter bank.

Now we see that there are downsamplers. Why have we used down samplers? Because of the Nyquist theorem, we know that the Nyquist sampling rate for proper reconstruction is twice the bandwidth. So, if the bandwidth is reduced to half, then I can reduce my Nyquist sampling rate to half. So, if the bandwidth is reduced to m times, I can reduce my filter sampling rate, not the filter sampling rate, but the overall signal sampling rate to m times. So, this is an efficient implementation, and this is called a digital filter bank.



So, it can either have a common input like this; here I have a common input x[n], and this x[n] is common to all the filters of the filter bank, or a summed output, which I am going to show in the next slide.



That filter banks can be categorized into two types: analysis bank or synthesis bank. So, the analysis bank will split the signal into several signals that have a lower sampling rate, whereas the synthesis filter bank will take signals at a lower sampling rate and then sum them to form an overall signal. And on this side and this side, the sampling rates are the same; whereas, since I have used five such components, the sampling rates of each component will be $f_x/5$, considering this is a uniform filter bank. So, filter banks can be used for performing spectral analysis because now I can clearly see small segments, and this is kind of a time-frequency analysis because I not only have signals at different time intervals, but also it looks at small sections of frequencies and signal synthesis, like I can combine a signal from the component signals.

So, like Fourier, if I have a periodic signal, I am splitting it into its sinusoidal components. So, that becomes your analysis bank, kind of. Whereas the reconstruction, the inverse series involves taking all the components, summing them, and making the signal. This is the synthesis filter bank. So, the analysis bank, as I said, will break down the signal into several frequency bands.

These are parallel filters; see, these are all parallel filters, whereas the synthesis bank will reconstruct the original signals from the signals from the processed bands, which are obtained by summing them. So, it is basically a combination of processes. So, there are several types of filter banks, such as the uniform filter bank, which I mentioned, like the DFT filter bank. So, it will analyze a specific region of frequencies, and if there are, say, n number of filters in the filter bank, then each will take care of π/n frequency range in the digital domain. Then we have non-uniform filter banks, where the frequency bandwidth of each band may be different, and these are often used in audio applications.

Finally, we can have perfect reconstruction filter banks, where we first split the signals x and then reconstruct them into $\hat{x}$, and these two should be the same maybe there will be some phase lag because of the delays involved in the process, but overall, the signal should be the same. So this x[n] and $\hat{x}[n]$ should be the same or similar in some sense.

So that is it. Thank you so much. We will meet again in the new class.