Signal Processing Systems (521279S)
Part 4(a) : Multirate techniques

Version 1.2

November 18, 2025

Goals of the study. Multirate algorithms allow efficient implementation of signal processing applications in many cases. This document introduces the basic ideas behind those methods, noble identities and polyphase decomposition. They are used in implementation of efficient decimators and interpolators.

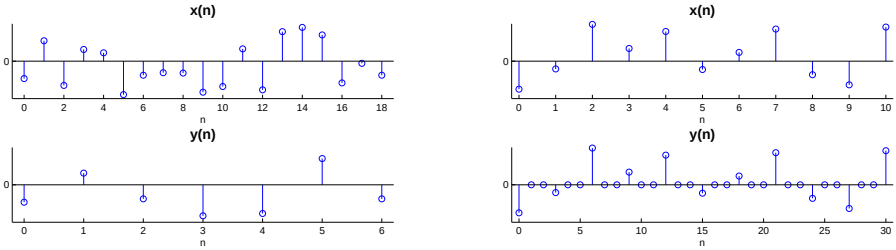
Contents

History
V1.0 11.10.2019 .
V1.1 21.11.2019 Add missing references.
V1.2 18.11.2025 Change on reference.

1 Preliminaries

1.1 Downsampling/upsampling

Basic operations in multirate processing are *downsampling* and *upsampling*. In downsampling by M, every Mth value of the original signal is retained (Fig. 1a), and the sample rate F_s is reduced to F_s/M. In upsampling by L, $L-1$ zeros are inserted between the original signal samples (Fig. 1b), which increases the sample rate to LF_s.



(a) Downsampling by $M = 3$.

(b) Upsampling by $L = 3$.

Figure 1: Rate changing operations.

1.2 Representations: signal flow and z-transform

The symbols used for the down- and upsampling operations in the signal flow graphs are provided in Table 1 along with symbols for other common operations. Also corresponding difference equations for the operation outputs, and Z-domain representations are given. The last symbol represents a system H. For example, FIR filter was specified before as a difference equation

$$y(n) = \sum_{k=0}^{N-1} c_k x(n-k).$$

If we denote the z-transform of $y(n)$ with $Y(z)$ and the z-transform of $x(n)$ with $X(z)$, then the FIR filter can represented as a system

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{k=0}^{N-1} c_k z^{-k}.$$

Note also that $H(z^M)$ means that z^M is substituted for z in $H(z)$. For example, in the case of FIR

$$H(z^M) = \sum_{k=0}^{N-1} c_k z^{-Mk}.$$

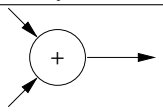
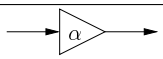
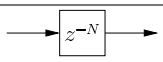
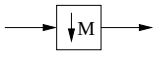
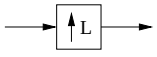
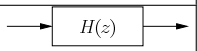
Name	Symbol	Output $y(n)$	In Z-domain
Addition		$x_1(n) + x_2(n)$	$Y(z) = X_1(x) + X_2(z)$
Multiplication by α		$\alpha x(n)$	$Y(z) = \alpha X(z)$
Delay of N time units		$x(n-N)$	$Y(z) = z^{-N}X(z)$
Downsampling by factor M		$x(Mn)$	$Y(z^M) = \frac{1}{M}\sum_{k=0}^{M-1} X(zW_M^k)$, where $W_M = e^{-j2\pi/M}$
Upsampling by factor L		$\begin{cases} x(n/L) & \text{if } n = kL \\ 0 & \text{otherwise} \end{cases}$	$Y(z) = X(z^L)$
System H			$Y(z) = H(z)X(z)$

Table 1: Symbols for signal flow graphs.

2 Noble identities

In implementation of signal processing algorithms we prefer to do arithmetic operations at a lowest possible sample rate which leads to savings in computations and also in memory. Noble identities provide us ways for changing the order of operations in signal flow graphs containing downsampling/upsampling operations so that these savings can be achieved. There are total of six identities.

2.1 Downsampling: identities 1-3

First three identities are related to the downsampling operation. They are illustrated in Table 2 using signal flow graphs. The first identity says that if multiplication and/or addition operations are followed by a downsampling operation, then the order of these operations can be changed. The effect of this reordering is that the arithmetic operations are now performed at a lower sample rate.

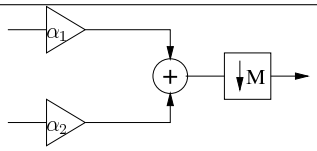
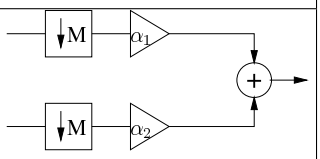
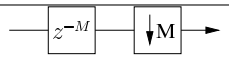
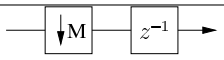
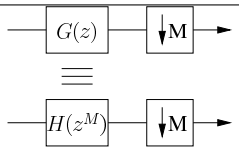
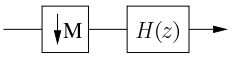
Name	Original structure	Transformed structure
Noble identity 1		
Noble identity 2		
Noble identity 3		

Table 2: Noble identities related to the downsampling operation.

The second identity considers delaying of signals. Delay of M units before downsampling is equal to a one unit delay after downsampling as shown in Table 2. Implementation of M-unit delay requires a first-in-first-out (FIFO) buffer of M words. So, application of the second identity leads to reduced memory costs.

The third identity combines the ideas from the identities 1 and 2. When only every Mth coefficient of a system (e.g. FIR filter) is non-zero, and the system is followed by a downsampling operation (factor M), then the order of these operations can be switched.

Example 1. For the filter

$$\begin{aligned} y(n) &= \sum_{k=0}^{4} g_k x(n-k) \\ &= g_0 x(n) + g_2 x(n-2) + g_4 x(n-4) \end{aligned}$$

only every 2nd coefficient (g_0, g_2, g_4) is non-zero. Using the z-transform, we can write this filter as

$$Y(z) = G(z)X(z),$$

where $Y(z)$ corresponds to the output $y(n)$, $X(z)$ corresponds to the input $x(n)$,

and $G(z) = g_0 + g_2 z^{-2} + g_4 z^{-4}$ represents the filter. We note that

$$\begin{aligned} G(z) &= g_0 + g_2 z^{-2} + g_4 z^{-4} \\ &= g_0 + g_2 (z^2)^{-1} + g_4 (z^2)^{-2} \\ &= H(z^2), \end{aligned}$$

where $H(z) = g_0 + g_2 z^{-1} + g_2 z^{-2}$. Now, if the output of $G(z)$ is downsampled by factor 2, we can do the downsampling first, and then apply the filter $H(z)$ as shown in Table 2. ■

2.2 Upsampling: identities 4-6

The identities related to the upsampling operation resemble the identities 1-3, and are provided in Table 3. The upsampling operation inserts zeros to the signal, and it is wasteful to do that before any operations. In the case of the fourth identity, we avoid arithmetic operations, where those zero samples are involved. In the case of the fifth identity, we avoid the buffering of zero samples. The sixth identity, where the system $G(z)$ can be represented as $H(z^L)$, avoids both storage and processing of zero samples.

It may look strange that zeros are inserted to the sequences as the last step in the transformed structures. The answer is that upsamplings will appear in contexts, where implementations do *not* actually insert zeros. In the Section 4 efficient implementation of signal interpolation will be discussed, which provides an example of this (especially, see Fig. 6).

3 Polyphase representation

Noble identities provide some tools for finding efficient implementations of systems. Note that the noble identity 3 shown in Table 2 involves a block $H(z^M)$, and $H(z^L)$ appears in the identity 6 (Table 3). To apply those identities, we should have structures where the transfer functions have these forms. For that purpose, we can use polyphase decompositions, which are described now.

3.1 Discrete sampling of signals

Polyphase representation is based on the concept of *discrete sampling*. In discrete sampling by M, every Mth value of a discrete signal $x(n)$ is retained in the result, and other values are set to zero. This can be done for different phase offsets λ. Resulting signals, $x_\lambda^{(p)}(n)$, are called the *polyphase components* of $x(n)$, and the

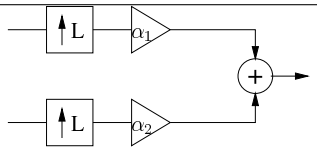
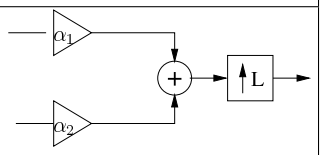
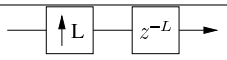
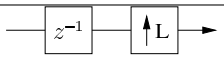
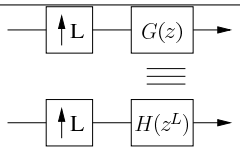
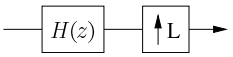
Name	Original structure	Transformed structure
Noble identity 4		
Noble identity 5		
Noble identity 6		

Table 3: Noble identities related to the upsampling operation.

original signal can be represented as a sum of the polyphase components:

$$x(n) = \sum_{\lambda=0}^{M-1} x_{\lambda}^{(p)}(n). \tag{1}$$

An example of discrere sampling is provided in Fig. 2.

Based on this decomposition, the Z-transform of $x(n)$ can be written as

$$X(z) = \sum_{\lambda=0}^{M-1} z^{-\lambda} X_{\lambda}^{(p)}(z^M), \tag{2}$$

where

$$X_{\lambda}^{(p)}(z^M) = \sum_{m=-\infty}^{\infty} x(mM+\lambda) z^{-mM}. \tag{3}$$

This is called the polyphase representation of the Z-transform $X(z)$.

3.2 System decomposition

Polyphase decompositions can also be done for filter impulse responses, and corresponding transfer functions $H(z)$:

$$H(z) = \sum_{\lambda=0}^{M-1} z^{-\lambda} H_{\lambda}^{(p)}(z^M). \tag{4}$$

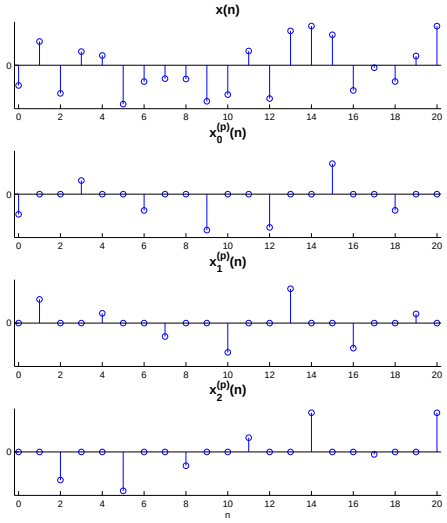


Figure 2: Example of discrete sampling by factor $M = 3$. The uppermost plot shows the signal $x(n)$, and below it are its polyphase components (for $M = 3$).

For matrix notations, the terms of the sum (4) can be collected to a vector

$$\mathbf{h}^{(p)}(z) = \begin{bmatrix} H_0^{(p)}(z^M) \\ z^{-1}H_1^{(p)}(z^M) \\ \vdots \\ z^{-(M-1)}H_{M-1}^{(p)}(z^M) \end{bmatrix}. \tag{5}$$

Example 2. As an example of the polyphase decomposition of systems, let us consider a FIR filter having a transfer function

$$H(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4} + a_5 z^{-5}.$$

Its polyphase representation for $M = 2$ is

$$H(z) = \underbrace{(a_0 + a_2 z^{-2} + a_4 z^{-4})}_{H_0^{(p)}(z^2)} + z^{-1} \underbrace{(a_1 + a_3 z^{-2} + a_5 z^{-4})}_{H_1^{(p)}(z^2)},$$

and for $M = 3$, it is

$$H(z) = \underbrace{(a_0 + a_3 z^{-3})}_{H_0^{(p)}(z^3)} + z^{-1} \underbrace{(a_1 + a_4 z^{-3})}_{H_1^{(p)}(z^3)} + z^{-2} \underbrace{(a_2 + a_5 z^{-3})}_{H_2^{(p)}(z^3)}.$$

For $M = 2$, the vector-based representation is

$$\mathbf{h}^{(p)}(z) = \begin{bmatrix} a_0 + a_2 z^{-2} + a_4 z^{-4} \\ z^{-1}(a_1 + a_3 z^{-2} + a_5 z^{-4}) \end{bmatrix}.$$

and for $M = 3$, it is

$$\mathbf{h}^{(p)}(z) = \begin{bmatrix} a_0 + a_3 z^{-3} \\ z^{-1}(a_1 + a_4 z^{-3}) \\ z^{-2}(a_2 + a_5 z^{-3}) \end{bmatrix}.$$

■

We note that the polyphase representation consist of terms, which are based on the powers z^M, namely, $H_\lambda^{(p)}(z^M)$. This fact allows us to exploit the noble identities 3 and 6 in filter implementations, as will be shown Section 4. As the superscript $^{(p)}$ in $H_\lambda^{(p)}(z^M)$ makes the notation complicated, we leave it out and use just $H_\lambda(z^M)$ in the following.

4 Efficient decimation and interpolation

4.1 Modifying FIR based structures

Basic applications of the down- and upsampling operations are the *decimation* and *interpolation*. In decimation, downsampling is preceeded by a filter, whose purpose is to prevent aliasing. An efficient structure for decimation is found by reversing the order of anti-aliasing and downsampling. The solution is based on the noble identity 1 and is illustrated in Fig. 3. In Fig. 3a, the anti-aliasing filter is represented in the direct form[1]. In the efficient implementation (Fig. 3b), the downsampling operation is embedded to this structure, which reduces the number (rate) of multiplication and addition operations.

It is not easy to see from Fig. 3b what would be the implementation of the efficient decimation. Alternatively, we can derive the polyphase decomposition of the original filter $H(z)$:

$$H(z) = H_0(z^M) + z^{-1}H_1(z^M) + ...z^{-(M-1)}H_{M-1}(z^M)$$

Corresponding dataflow is shown in Fig. 4a. To this structure we apply the noble identity 3, which leads to the dataflow shown in Fig. 4b.

[1]Direct form of an FIR filter is based on direct implementation of the convolution, where delayed versions of the input are multiplied by filter coefficients. In the *transposed* direct form, outputs of the multiplications are delayed.

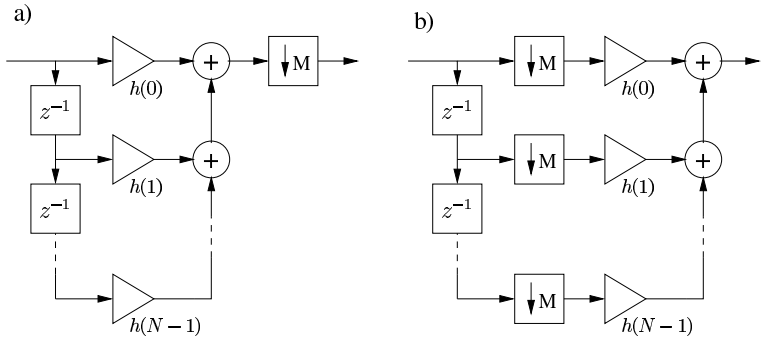


Figure 3: Efficient structure for decimation: (a) original decimator, where downsampling is done after anti-alias filtering; (b) modified structure.

The delay/downsampling structure of Fig. 4b splits the input signal to M lower rate subsequences which can be represented with a *commutator* (Fig. 4c). The structure outputs the next value when M input values have been fed to the corresponding filters $H_i(z)$ by the commutator, and the sum of filter outputs has been computed.

In the case of interpolation, upsampling operation is followed by anti-imaging filter. If we represent the filter in the transposed direct form (Fig. 5a), we can see how noble identity 4 leads to a structure where multiplications are done at a lower rate (Fig. 5b).

Again, it is easier to understand the implementation if we perform polyphase decomposition of the anti-imaging filter $H(z)$:

$$H(z) = H_0(z^L) + z^{-1}H_1(z^L) + ...z^{-(L-1)}H_{L-1}(z^L).$$

Fig. 6 shows the derivation of the commutator-based solution. When a new input sample arrives the output of each filter $H_i(z)$ can be updated. Then, those values are passed forward one by one using the commutator.

4.2 Cascaded integrator-comb filters

For decimation/interpolation using FIR filters, noble identities can be applied to improve efficiency as discussed above. However, there are other strategies for implementing antialiasing/-imaging filters for sample rate changers. For example, *cascaded integrator-comb (CIC) filters* (Hogenauer 1981, Donadio 2000, Lyons 2020) are well suited to hardware implementation as they are free of multipliers and include efficient recursive computations. These characteristics are very important in high data-rate communications systems, for example.

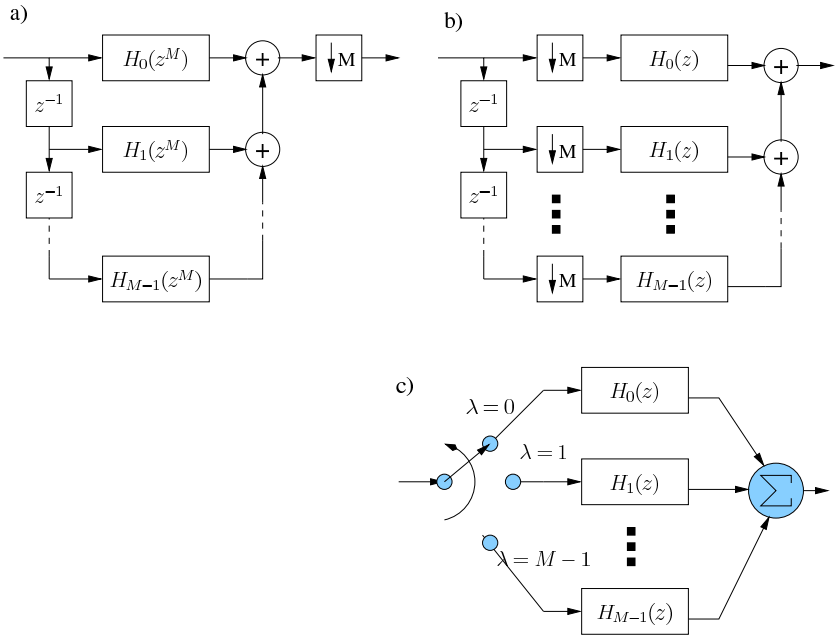


Figure 4: Structure of a decimator based on polyphase decomposition: (a) polyphase decomposition of antialias filter, (b) after application of the noble identity 3, (c) the delay/downsampling structure replaced with a commutator.

The CIC filters are constructed by cascading two basic components, *integrators* and *combs*. The integrator (Fig. 7(a)) is an IIR filter with a transfer function

$$I(z) = \frac{1}{1 - z^{-1}} \tag{6}$$

and the transfer function of the *comb* (Fig. 7(b)) is

$$C_D(z) = 1 - z^{-D}. \tag{7}$$

In the cascade, these filters make up a recursive running sum filter (up to the scaling factor $1/D$), which is equivalent to the nonrecursive D-point moving average filter. That is,

$$\frac{1}{D} C_D(z) I(z) = \underbrace{\frac{1}{D} \frac{1 - z^{-D}}{1 - z^{-1}}}_{\text{running sum filter}} = \underbrace{\frac{1}{D} \sum_{d=0}^{D-1} z^{-d}}_{\text{moving average filter}} . \tag{8}$$

$C_D(z)I(z)$ is the first-order CIC filter. It is a low-pass filter with a sinc-shaped frequency response. Several (N) integrator/comb pairs can be cascaded. For example, Fig. 8(a) illustrates a three-stage CIC filter. As the comparison of Fig. 9(a)

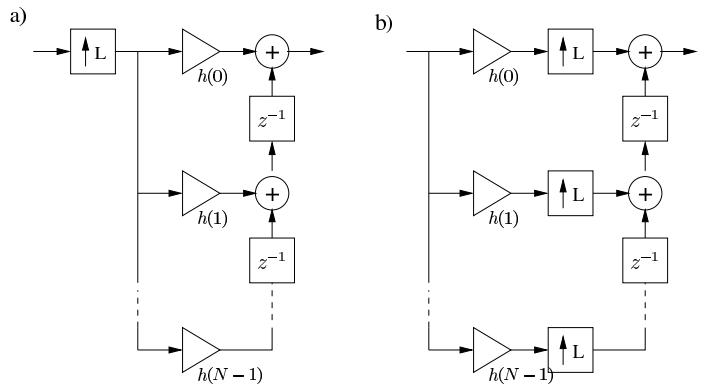


Figure 5: Structures for interpolation: (a) original interpolator, where upsampling is done before anti-image filtering; (b) efficient structure.

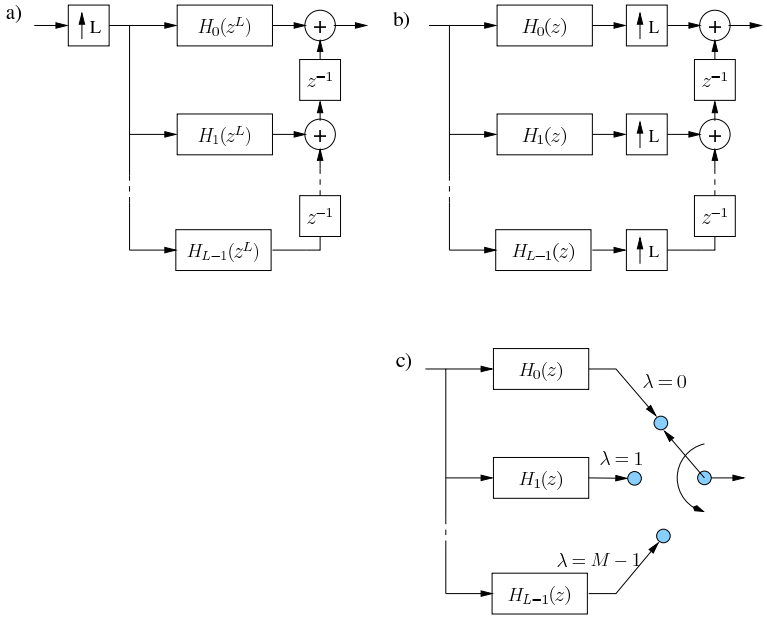
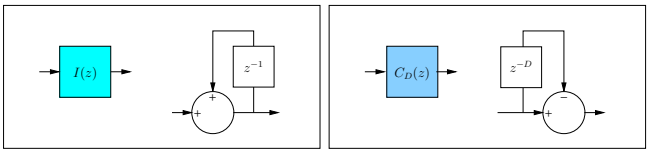


Figure 6: Polyphase interpolation: (a) structure showing polyphase decomposition of the anti-imaging filter, (b) the structure after application of the noble identity 6, (c) the delay/upsampling/sum structure replaced with a commutator.

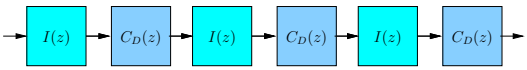
and Fig. 9(b) shows, the amount of attenuation (i.e. the steepness of the sinc shape) depends on the number of stages used.

The main application of CIC filters is in implementing antialiasing/imaging for decimation and interpolation. We use $D = RM$, where R is the rate change

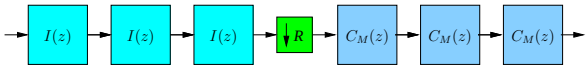


(a) Integrator. (b) Comb.

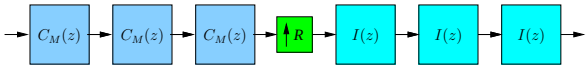
Figure 7: Basic components of the CIC filters.



(a) CIC filter.



(b) CIC decimator.



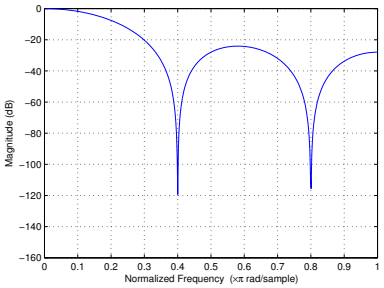
(c) CIC interpolator.

Figure 8: CIC systems ($N = 3$).

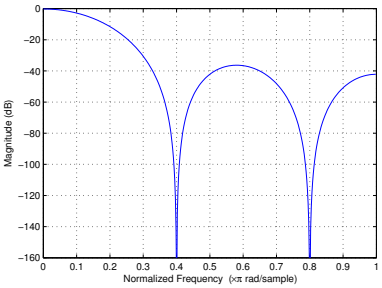
factor and M is a filter design parameter, whose value is typically either 1 or 2. The integrator and comb are linear systems and we can change their order in the CIC filter. In the case of decimation, we organize the components of the filter so that a cascade of N integrators is followed by a N-stage cascade of combs. The downsampler, which follows the filter, can then be moved between these cascades using the noble identities. The result is exemplified by Fig. 8(b).

In the case of interpolation, we change the order of the integrator/comb cascades, that is, the combs are followed by the integrators. The upsampler, which preceeds the filter, can then be moved between the cascades (Fig. 8(c)). Note that in the systems, the comb stages $C_D(z)$ have $D = M$.

Finally, note that CIC filters are far from ideal for antialiasing/imaging. To tolerate this, the input signal should be narrowband (say below 0.1-0.2 in the case of Fig. 9). In addition, a CIC rate changer is typically accompanied by a FIR filter, which makes the passband response flat (see (Lyons 2020)).



(a) Frequency response ($N = 2$).



(b) Frequency response ($N = 3$).

Figure 9: Examples of CIC filter frequency responses ($D = 5$).

5 Decimation example: image rescaling

We illustrate the concepts now by a concrete application in image processing, reduction of image resolution.

5.1 Aliasing

Let us consider reduction of the image size to 1/4 of the original size. If the image is just downsampled by factor 2 in both horizontal and vertical directions, aliasing effects may be observed in the final result as illustrated in Fig. 10, bottom-left image. We see variations in the areas of fine texture as the high-frequency content becomes aliased. To avoid aliasing, the image must be low-pass filtered before downsampling. The result is illustrated by Fig. 10, bottom-right, where the original image has been filtered by the mask $\mathbf{h}\mathbf{h}^T$, where $\mathbf{h} = [1/16, 1/4, 3/8, 1/4, 1/16]^T$, that is,

$$\mathbf{h}\mathbf{h}^T = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}.$$

5.2 Computation

Let us now consider different structures for implementing the rescaling operation for one color channel (R, G, or B). A straightforward solution is just to filter the image first with the 5×5 mask $\mathbf{h}\mathbf{h}^T$, and then to pick every second row and

Figure 10: An example of aliasing in image scaling. Annoying variation in the high-textured areas is suppressed by low-pass filtering.

column. So, 75% of computed pixel values are just discarded. As there are about 25 MAC operations per *input* pixel (5×5 filter mask contains 25 coefficients), there are 100 MAC operations per *output* pixel.

Of course, an improved solution would center the filter mask according to the coordinates of the output pixels. Then, only 25 MAC operations would be needed per *output* pixel.

Images are 2-D signals, whereas our theory for efficient decimation has considered only 1-D signal processing. However, filtering by the mask $\mathbf{h}\mathbf{h}^T$ can be done so that we filter the image first by the mask $\mathbf{h}^T$ and then by the mask $\mathbf{h}$: it is said that $\mathbf{h}\mathbf{h}^T$ is a *separable* filter. Filtering by $\mathbf{h}^T$, which is a row vector, means that we consider each row of the image as a 1-D signal, and filter them separately. Similarly, filtering by $\mathbf{h}$ means that we consider each column of the image as a 1-D signal, and filter them separately. As the length of $\mathbf{h}$ is 5 elements, we need $5+5 = 10$ MAC operations per *input* pixel and $4 \times 10 = 40$ MAC operations per *output* pixel, if downsampling is done after filtering.

We can again do computations needed just for output pixels. Then, $\mathbf{h}^T$ is applied only at every second pixel on the row. Every row must be processed so there are total of $H \times (W/2) \times 5$ MAC operations for the first step. In the

second step, $\mathbf{h}$ is applied only at every second pixel on the column. There are $W/2$ columns to process, so the total number of MAC operations is $(W/2)\times(H/2)\times 5$. As there are $(W/2)\times(H/2)$ output pixels, the operation count per output pixel is

$$\frac{H\times(W/2)\times 5+(W/2)\times(H/2)\times 5}{(W/2)\times(H/2)}=15.$$

In fact, this implementation corresponds to the one that we get by application of polyphase decomposition and noble identities. Note that the filter vector $\mathbf{h}$ corresponds to the system

$$H(z)=(1/16)+(1/4)z^{-1}+(3/8)z^{-2}+(1/4)z^{-3}+(1/16)z^{-4},$$

whose polyphase decomposition for $M=2$ is

$$H(z)=\underbrace{(1/16)+(3/8)z^{-2}+(1/16)z^{-4}}_{H_0(z^2)}+z^{-1}(\underbrace{(1/4)+(1/4)z^{-2}}_{H_1(z^2)}).$$

In Fig. 11, the starting point is the structure where row filtering is followed by downsampling. Polyphase decomposition and noble identities give the efficient structure. There are $W/2$ computations of $H_0(z)$ and $H_1(z)$ outputs for one row, that is, $(W/2)\times(3+2)$ MAC operations. Similarly, for each column, we have $H/2$ computations of $H_0(z)$ and $H_1(z)$: $(H/2)\times(3+2)$ MACs. The net result is that the operation count per output pixel is

$$\frac{H\times(W/2)\times(3+2)+(W/2)\times(H/2)\times(3+2)}{(W/2)\times(H/2)}=15.$$

Even more optimized version can be obtained by exploiting the fact that $\mathbf{h}$ is symmetric (1/16 and 1/4 appear twice in $\mathbf{h}$).[2]

[2]We can write $H_0(z)=(1/16)(1+z^{-2})+(3/8)z^{-1}$ and $H_1(z)=(1/4)(1+z^{-1})$, which provides hints for the structure of the signal flow. How many MACs per output pixel are needed finally? Moreover, do you need multiplications?

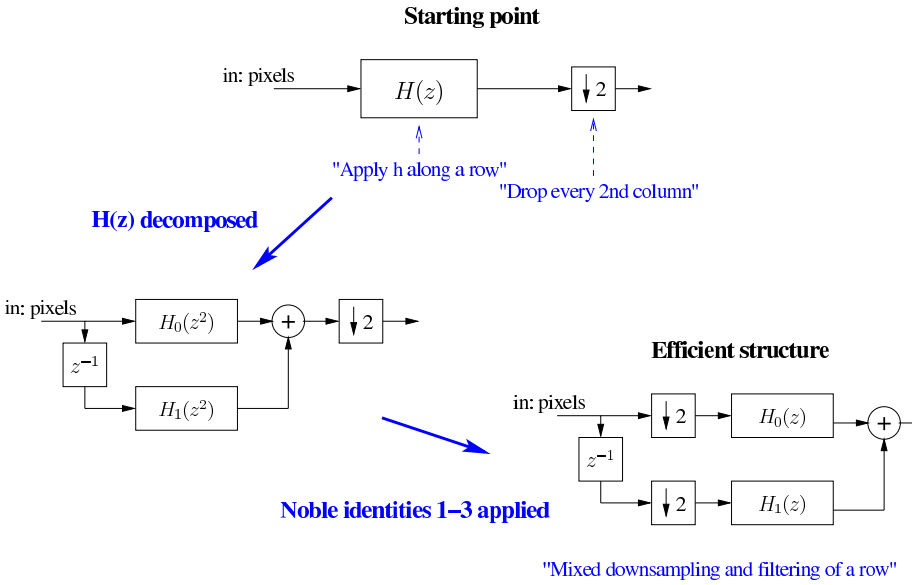


Figure 11: Derivation of the efficient decimator for image rows (and columns).

References

Donadio, M. P. (2000) CIC filter introduction. https://dspguru.com/files/cic.pdf (accessed 18.11.2025).

Hogenauer, E. B. (1981) An economical class of digital filters for decimation and interpolation. IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-29(2):155–162.

Lyons, R. (2020) A beginner's guide to cascaded integrator-comb (CIC) filters. https://www.dsprelated.com/showarticle/1337.php (accessed 18.11.2025).