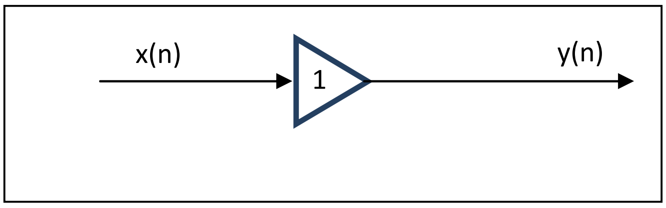
Difference equations, The digital filter

A difference equation specifies the output *y(n*) for a given input *x(n).* Essentially specifies the behaviour of the digital filter.

Examples of simple digital filters

Unity Gain Filter:

$$y(n) = x(n)$$



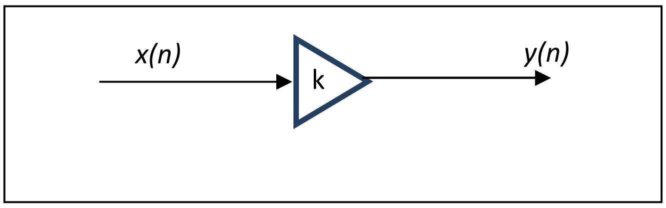
Each output value *y(n)* is exactly the same as the corresponding input value *x(n)*:

$$y(0) = x(0)$$
$$y(1) = x(1)$$
$$y(2) = x(2)$$

This is a trivial case in which the filter has no effect on the signal. Note also that *x(n)* is equivalent to *x(nT).* This is the value of *x* at time *t=nT*. For brevity, the *T* is generally dropped. It is not necessary to keep on writing *T's* all the time.

Simple Gain Filter:

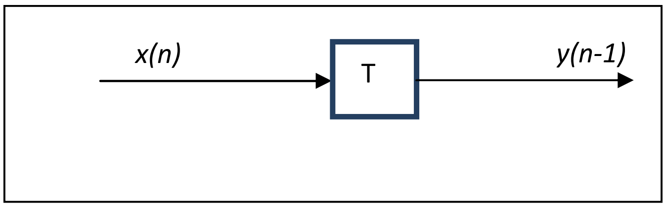
$$y(n) = kx(n)$$



This simply applies a gain factor k to each input value. If *k* > 1, the filter an amplifier, while if 0 < *k* < 1 it is an attenuator. If *k* < 0, this corresponds to an inverting amplifier. The unity gain filter above is simply the special case where *K* = 1.

Pure Delay Filter:

$$y(n) = x(n-1)$$



The output value at time *t* = *nT* is simply the input at time *t* = *(n-1)T*, i.e. the signal is delayed by time T.

$$y(0) = x(-1)$$
$$y(1) = x(0)$$
$$y(2) = x(1)$$

Note that as sampling is assumed to commence at *t* = 0, the input value *x-1* at *t* = *-T* is undefined. It is usual to take this (and any other values of *x* prior to *t* = 0) as zero (Initialisation).

Two-Term Difference Filter:

$$y(n) = x(n) - x(n-1)$$

Problem: Draw a block diagram that represents the above filter.

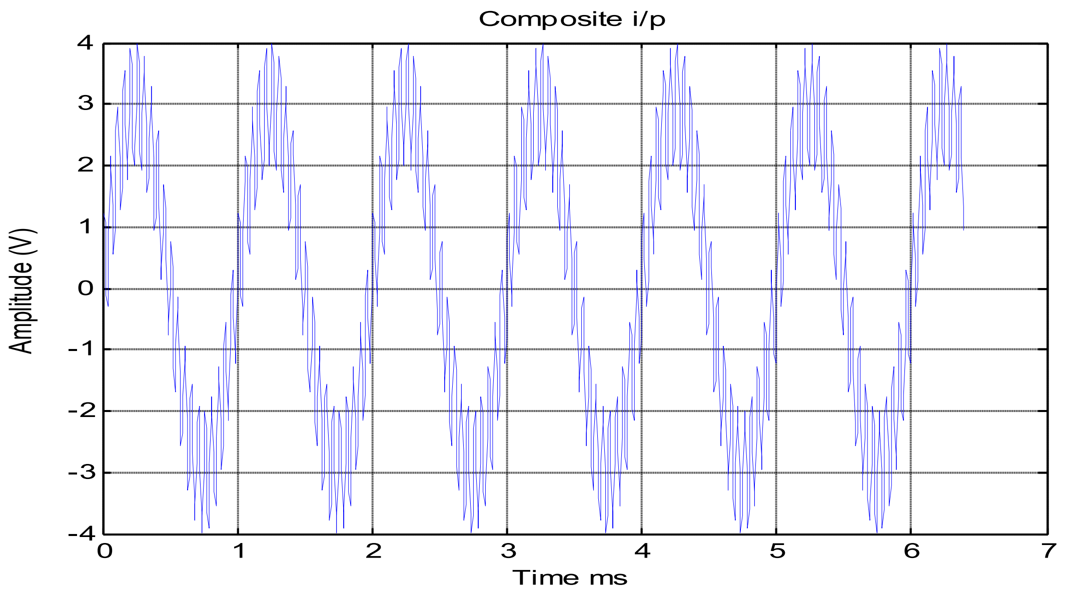
The output value at *t* = *nT* is equal to the difference between the current input value *x(n)* and the previous input *x(n-1)*:

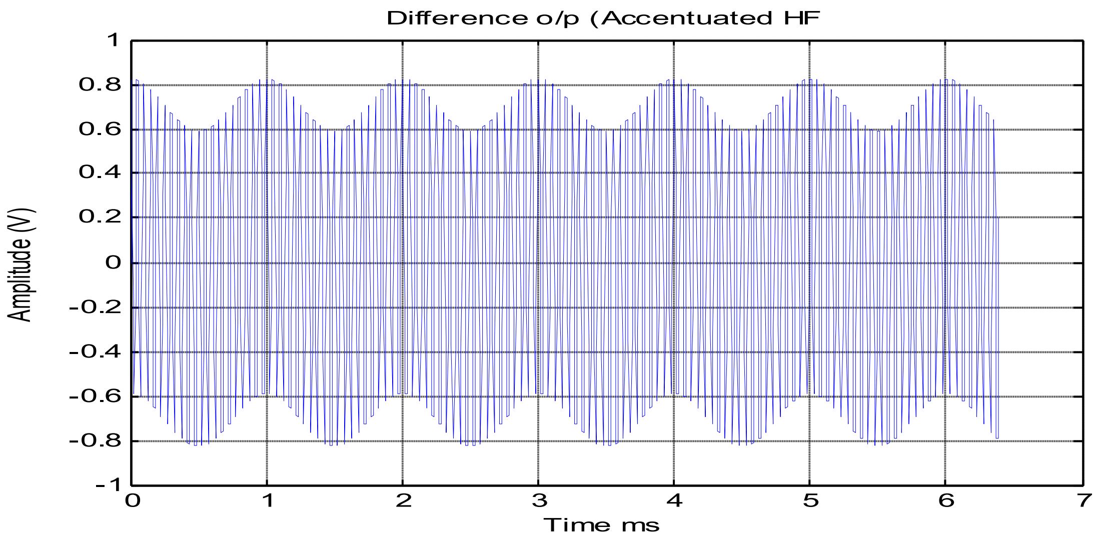
$$\begin{aligned} y(0) &= x(0) - x(-1) \\ y(1) &= x(1) - x(0) \\ y(2) &= x(2) - x(1) \end{aligned}$$

i.e. the output is the *change* in the input over the most recent sampling interval *T*. The effect of this filter is similar to that of an analogue differentiator circuit. The output level is sensitive to the difference between adjacent inputs. Consider the case where the composite signal:

$$s(t) = 3\sin(n \times \theta_a) + \sin(n \times 20 \times \theta_a)\,, \quad \theta_a = \frac{2\pi f_a}{f_s},\; f_s = 1\,kHz$$

is applied to the input of the difference filter.





You can see that the output has the high frequencies accentuated.

Two-Term Average Filter:

$$y(n) = \frac{x(n) + x(n-1)}{2}$$

Problem: Draw a block diagram that represents the above filter.

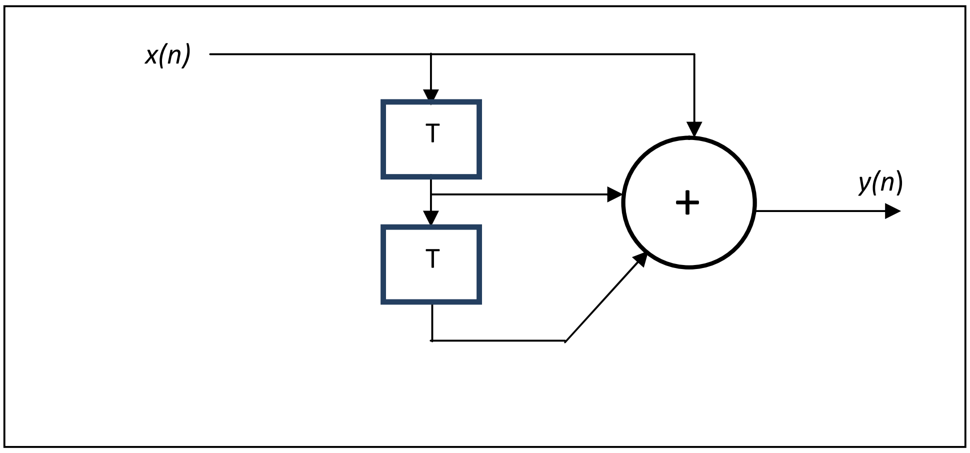
The output is the average (arithmetic mean) of the current and previous input. This is a simple type of low pass filter as it tends to smooth out high-frequency variations in a signal. We will look at more effective low pass filter designs later.

Three-term average filter:

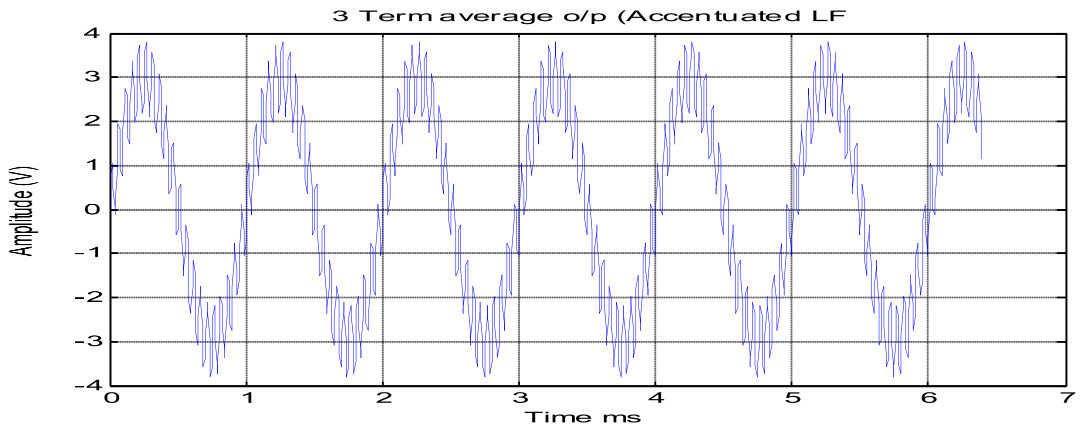
This is similar to the previous example, with the average being taken of the current and two previous inputs.

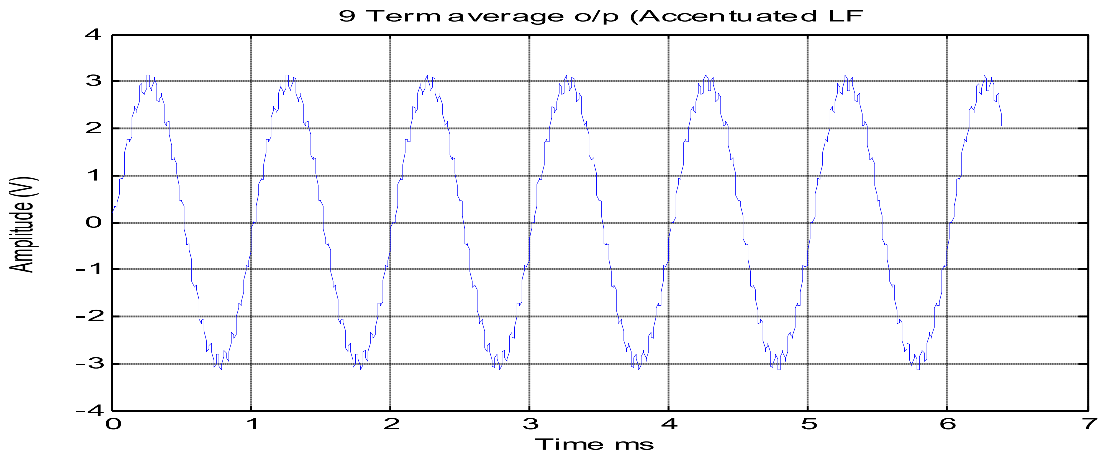
$$y(n) = \frac{x(n) + x(n-1) + x(n-2)}{3}$$

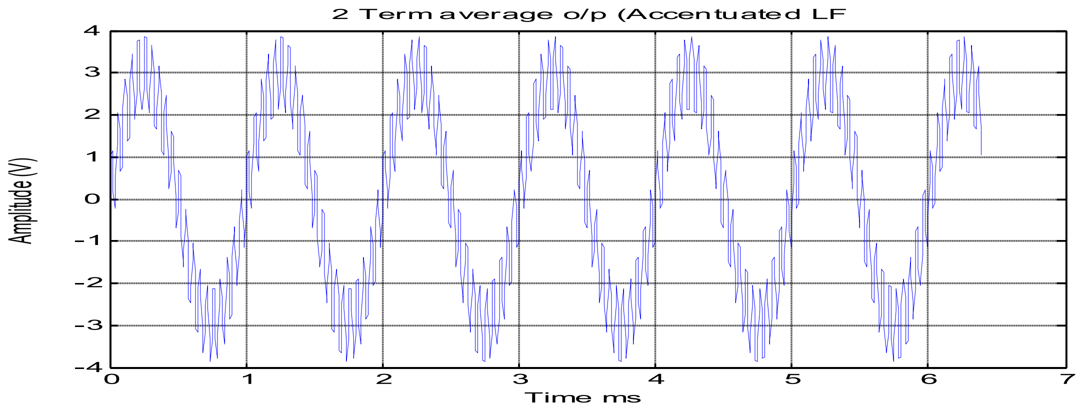
As before, *x(-1)* and *x(-2)* are taken to be zero. Or expressed in block diagram form:

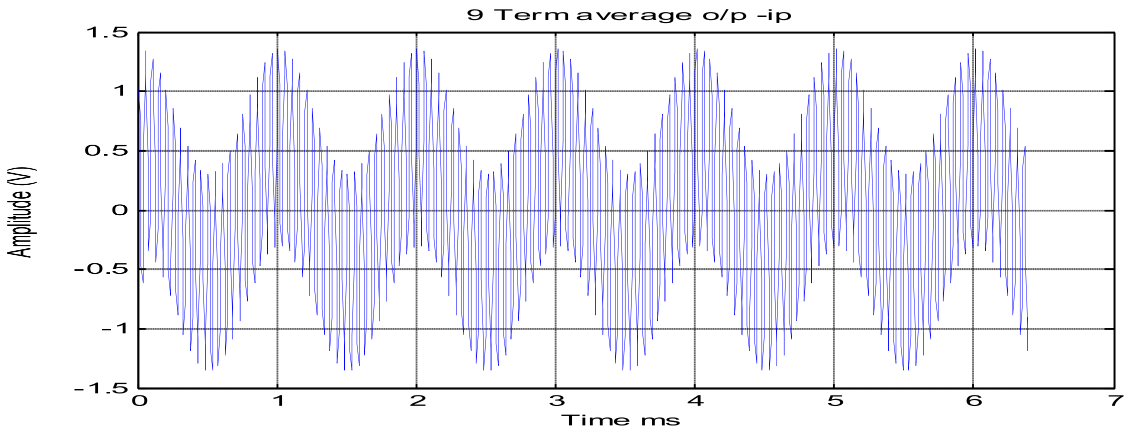


This is a 3 term moving average filter. The more terms that you use in the moving average filter, the more effective it is. The same composite input is applied to the averaging filter and the following outputs are obtained:


3 Term average o/p (Accentuated LF
Amplitude (V)
4
3
2
1
0
-1
-2
-3
-4
0
1
2
3
4
5
6
7
Time ms


9 Term average o/p (Accentuated LF
Amplitude (V)
4
3
2
1
0
-1
-2
-3
-4
0
1
2
3
4
5
6
7
Time ms


2 Term average o/p (Accentuated LF
Amplitude (V)
4
3
2
1
0
-1
-2
-3
-4
0
1
2
3
4
5
6
7
Time ms



You can see from the graphs above that the difference filter accentuates the high frequencies (useful for edge detection) and the average accentuates the low frequencies (useful for noise reduction).

Problem: Draw a block diagram that represents the above filter.

Order of a digital filter

The *order* of a digital filter is the number of *previous* inputs or outputs (stored in the processor's memory) used to calculate the current output. Thus the unity gain filter and the simple gain filter above are zero-order filters, as the current output *y(n)* depends only on the current input *x(n)* and not on any previous inputs. The next 3 filters are all of first order, as one previous input *x(n-1*) is required to calculate *y(n)*. In the 3-Term average filter and the central difference filter two previous inputs *x(n-1)* and *x(n-2*) are needed, so these are second-order filters. Filters may be of any order from zero upwards.

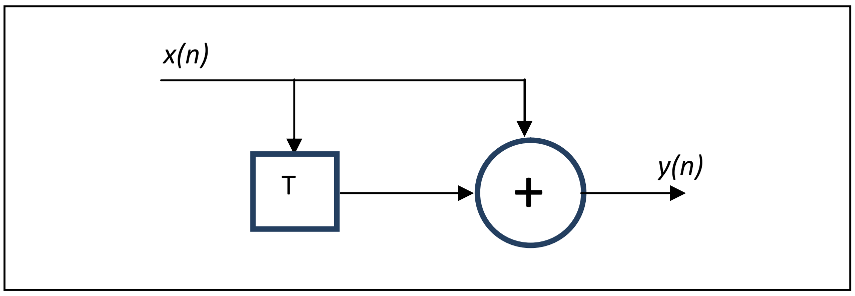
Digital filter coefficients

All of the digital filter examples given above can be written in the following general forms:

$$\begin{aligned} &Zero\ order: && y(n)=a_0x(n) \\ &1st\ order: && y(n)=a_0x(n)+a_1x(n-1) \\ &2nd\ order: && y(n)=a_0x(n)+a_1x(n-1)+a_2x(n-2) \end{aligned}$$

Similar expressions can be developed for filters of any order. The constants *a0, a1, a2*, ... appearing in these expressions are called the *filter coefficients*. It is the values of these coefficients that determine the characteristics of a particular filter.

1st order filter (FIR)



$$y(n) = x(n) + 0.5x(n-1)$$
$$Y(z) = X(z) + 0.5X(z)z^{-1} = X(z)\left\{1 + 0.5z^{-1}\right\}$$
$$H(z) = \frac{Y(z)}{X(x)} = 1 + 0.5z^{-1}$$

Note that the delay of 1 in *x(n-1)* results in a z^{-1} in the equation for the z-transform.

FIR/IIR Classification

A filter is finite impulse response (*FIR*) if the output has a finite response time to an input impulse and is *IIR* infinite impulse response (*IIR*) if the response time is infinite. An *IIR* filter is also called a *recursive* filter and an FIR is called a *non-recursive* filter. An *FIR* filter will contain only *x(n)* terms and an *IIR* filter will contain *y(n)* terms and also possibly *x(n)* terms.

Tutorial question

A digital filter is described by the expression:

$$y(n) = 2x(n) - x(n-1) + 0.8y(n-1) + y(n-2)$$

(a) State whether the filter is recursive or non-recursive.
(b) State the order of the filter.
(c) Derive the filter transfer function.
(d) The following sequence of input values is applied to the filter.

x = 5, 16, 8, -3

Determine the output sequence for the filter, from *y(0)* to *y(4)*. The order of a recursive filter is the largest number of previous input *or* output values required to compute the current output. This definition can be regarded as being quite general: it applies both to FIR and IIR filters.