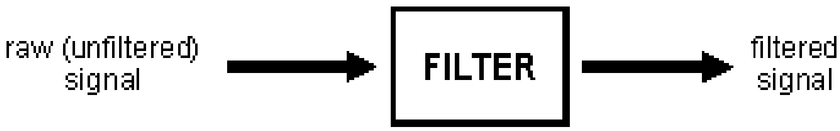
## Filter Analysis

### 1 Digital filters




$$x_0, x_1, x_2, x_3, \ldots x_n$$

=>

$$y_0, y_1, y_2, y_3, \ldots y_n$$

#### (a) Unity gain filter

$$y_n = x_n$$

Each output value $y_n$ is exactly the same as the corresponding input value $x_n$:

$$y_0 = x_0$$
$$y_1 = x_1$$
$$y_2 = x_2$$
$$\ldots etc$$

#### (b) Simple gain filter

$$y_n = Kx_n$$

where $K$ = constant.

#### (c) Pure delay filter

$$y_n = x_{n-1}$$

The output value at time $t = nh$ is simply the input at time $t = (n-1)h$, i.e. the signal is delayed by time $h$:

$$y_0 = x_{-1}$$
$$y_1 = x_0$$
$$y_2 = x_1$$
$$y_3 = x_2$$
… etc

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

## (d) Two-term difference filter:

$$y_n = x_n - x_{n-1}$$

The output value at $t = nh$ is equal to the difference between the current input $x_n$ and the previous input $x_{n-1}$:

$$y_0 = x_0 - x_{-1}$$
$$y_1 = x_1 - x_0$$
$$y_2 = x_2 - x_1$$
$$y_3 = x_3 - x_2$$
… etc

## (e) Two-term average filter

$$y_n = \frac{x_n + x_{n-1}}{2}$$

The output is the average (arithmetic mean) of the current and previous input:

$$y_0 = \frac{x_0 + x_{-1}}{2}$$

$$y_1 = \frac{x_1 + x_0}{2}$$

$$y_2 = \frac{x_2 + x_1}{2}$$

$$y_3 = \frac{x_3 + x_2}{2}$$

… etc

This is a simple type of low pass filter as it tends to smooth out high-frequency variations in a signal.

## (f) Central difference filter

$$y_n = \frac{x_n - x_{n-2}}{2}$$

This is similar in its effect to example (4). The output is equal to half the change in the input signal over the previous two sampling intervals:

$$y_0 = \frac{x_0 - x_{-2}}{2}$$

$$y_1 = \frac{x_1 - x_{-1}}{2}$$

$$y_2 = \frac{x_2 - x_0}{2}$$

$$y_3 = \frac{x_3 - x_1}{2}$$

… etc

## (2) Order of a digital filter

The *order* of a digital filter is the number of *previous* inputs (stored in the processor's memory) used to calculate the current output.

Thus

Examples (a) and (b) above are zero-order filters;
Examples (c) and (d) above are first-order filters;
Examples (d) and (e) above are second-order filters;

## (3) Digital filter coefficients

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

*Zero order:* $y_n = a_0 x_n$

*First order:* $y_n = a_0 x_n + a_1 x_{n-1}$

*Second order:* $y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$

## (4) Recursive and non-recursive filters

For all the examples of digital filters discussed so far, the current output ($y_n$) is calculated solely from the current and previous input values ($x_n$, $x_{n-1}$, $x_{n-2}$, ...). This type of filter is said to be *non-recursive*.

A *recursive* filter is one which in addition to input values also uses previous *output* values. These, like the previous input values, are stored in the processor's memory.

## (5) Example of a recursive filter

A simple example of a recursive digital filter is given by

$$y_n = x_n + y_{n-1}$$

In other words, this filter determines the current output ($y_n$) by adding the current input ($x_n$) to the previous output ($y_{n-1}$):

$$\begin{aligned} y_0 &= x_0 + y_{-1} \\ y_1 &= x_1 + y_0 \\ y_2 &= x_2 + y_1 \\ y_3 &= x_3 + y_2 \\ &\ldots \text{ etc} \end{aligned}$$

Note that $y_{-1}$ (like $x_{-1}$) is undefined, and is usually taken to be zero.

Let us consider the effect of this filter in more detail. If in each of the above expressions we substitute for $y_{n-1}$ the value given by the previous expression, we get the following:

$$\begin{aligned} y_0 &= x_0 + y_{-1} = x_0 \\ y_1 &= x_1 + y_0 = x_1 + x_0 \\ y_2 &= x_2 + y_1 = x_2 + x_1 + x_0 \\ y_3 &= x_3 + y_2 = x_3 + x_2 + x_1 + x_0 \\ &\ldots \text{ etc} \end{aligned}$$

This example demonstrates an important and useful feature of recursive filters: the economy with which the output values are calculated, as compared with the equivalent non-recursive filter. In this example, each output is determined simply by adding two numbers together. For instance, to calculate the output at time $t = 10h$, the recursive filter uses the expression

$$y_{10} = x_{10} + y_9$$

To achieve the same effect with a non-recursive filter (i.e. without using previous output values stored in memory) would entail using the expression

$$y_{10} = x_{10} + x_9 + x_8 + x_7 + x_6 + x_5 + x_4 + x_3 + x_2 + x_1 + x_0$$

This would necessitate many more addition operations as well as the storage of many more values in memory.

## Order of a recursive (IIR) digital filter

**The order of a recursive filter is the largest number of previous input *or* output values required to compute the current output.**

## Coefficients of recursive (IIR) digital filters

From the above discussion, we can see that a recursive filter is basically like a non-recursive filter, with the addition of extra terms involving previous inputs ($y_{n-1}$, $y_{n-2}$ etc.).

A first-order recursive filter can be written in the general form

$$y_n = \frac{(a_0 x_n + a_1 x_{n-1} - b_1 y_{n-1})}{b_0}$$

Note the minus sign in front of the "recursive" term $b_1 y_{n-1}$, and the factor $(1/b_0)$ applied to all the coefficients. The reason for expressing the filter in this way is that it allows us to rewrite the expression in the following symmetrical form:

$$b_0 y_n + b_1 y_{n-1} = a_0 x_n + a_1 x_{n-1}$$

In the case of a second-order filter, the general form is

$$y_n = \frac{a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} - b_1 y_{n-1} - b_2 y_{n-2}}{b_0}$$

The alternative "symmetrical" form of this expression is

$$b_0 y_n + b_1 y_{n-1} + b_2 y_{n-2} = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$$

Note the convention that the coefficients of the inputs (the *x*'s) are denoted by *a*'s, while the coefficients of the outputs (the *y*'s) are denoted by *b*'s.

## The transfer function of a digital filter

In this section, we introduce what is called the *transfer function* of a digital filter. This is obtained from the symmetrical form of the filter expression, and it allows us to describe a filter by means of a convenient, compact expression. We can also use the transfer function of a filter to work out its frequency response.

First of all, we must introduce the *delay operator*, denoted by the symbol $z^{-1}$.

Applying the operator $z^{-1}$ to an input value (say $x_n$) gives the previous input ($x_{n-1}$):

$$z^{-1}\, x_n = x_{n-1}$$

Suppose we have an input sequence

$$\begin{aligned} x_0 &= 5 \\ x_1 &= -2 \\ x_2 &= 0 \\ x_3 &= 7 \\ x_4 &= 10 \end{aligned}$$

Then

$$\begin{aligned} z^{-1}\, x_1 &= x_0 = 5 \\ z^{-1}\, x_2 &= x_1 = -2 \\ z^{-1}\, x_3 &= x_2 = 0 \end{aligned}$$

and so on. Note that $z^{-1}\, x_0$ would be $x_{-1}$, which is unknown (and usually taken to be zero, as we have already seen).

Similarly, applying the $z^{-1}$ operator to an output gives the previous output:

$$z^{-1}\, y_n = y_{n-1}$$

Applying the delay operator $z^{-1}$ twice produces a delay of two sampling intervals:

$$z^{-1}\,(z^{-1}\, x_n) = z^{-1}\, x_{n-1} = x_{n-2}$$

We adopt the (fairly logical) convention

$$z^{-1}\, z^{-1} = z^{-2}$$

i.e. the operator $z^{-2}$ represents a delay of two sampling intervals:

$$z^{-2}\,x_n = x_{n-2}$$

This notation can be extended to delays of three or more sampling intervals, the appropriate power of $z^{-1}$ being used.

Let us now use this notation in the description of a recursive digital filter. Consider, for example, a general second-order filter, given in its symmetrical form by the expression

$$b_0 y_n + b_1 y_{n-1} + b_2 y_{n-2} = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$$

We will make use of the following identities:

$$y_{n-1} = z^{-1}\,y_n$$

$$y_{n-2} = z^{-2}\,y_n$$

$$x_{n-1} = z^{-1}\,x_n$$

$$x_{n-2} = z^{-2}\,x_n$$

Substituting these expressions into the digital filter gives

$$(b_0 + b_1 z^{-1} + b_2 z^{-2})\,y_n = (a_0 + a_1 z^{-1} + a_2 z^{-2})\,x_n$$

Rearranging this to give a direct relationship between the output and input for the filter, we get

$$\frac{y_n}{x_n} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{b_0 + b_1 z^{-1} + b_2 z^{-2}}$$

The general form of the transfer function of a nth-order recursive filter

$$\frac{y_n}{x_n}=\frac{a_0+a_1 z^{-1}+\ldots+a_n z^{-n}}{b_0+b_1 z^{-1}+\ldots+b_n z^{-n}}$$

Any a filter can be expressed in form of transfer function. For example,

$$y_n = x_n + 2x_{n-1} + x_{n-2} - 2y_{n-1} + y_{n-2}$$

Expressing this in terms of the $z^{-1}$ operator gives

$$(1 + 2z^{-1} - z^{-2})\, y_n = (1 + 2z^{-1} + z^{-2})\, x_n$$

and so the transfer function is

$$\frac{y_n}{x_n} = \frac{1 + 2z^{-1} + z^{-2}}{1 + 2z^{-1} - z^{-2}}$$

## Matlab Analysis

**Function name for filter analysis is "filter"**

Y = FILTER(B,A,X) filters the data in vector X with the filter described by vectors A and B to create the filtered data Y.

B, and A are coefficients discussed above, which determine properties of the designed filter. For different purposes (such as low-pass, high-pass and band-pass), different A and B should be chosen. There are many methods to design a specific filter. The typical one is called the Butterworth digital and analog filter design. In Matlab, the function name is **"butter"**

[B,A] = BUTTER(N,Wn) designs an Nth order lowpass digital Butterworth filter and returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z. The cutoff frequency Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. If Wn is a two-element vector, Wn = [W1 W2], BUTTER returns an order 2N bandpass filter with passband W1 < W < W2.

[B,A] = BUTTER(N,Wn,'high') designs a highpass filter.
[B,A] = BUTTER(N,Wn,'low') designs a lowpass filter.
[B,A] = BUTTER(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

(Signal Processing Toolbox)