# Digital Filters

## *1.0 Aim*

Understand the principles of operation and characterization of digital filters

## *2.0 Learning Outcomes*

You will be able to:

- Implement a digital filter in MATLAB.
- Investigate how commercial filters work
- Describe the transfer function of a filter

## *A. Introduction*

Suppose the "raw" signal which is to be digitally filtered is in the form of a voltage waveform described by the function V = x (t) where t is time. This signal is sampled at time intervals h (the sampling interval). The sampled value at time t = ih is $x_i = x(ih)$

Thus the digital values transferred from the ADC to the processor can be represented by the sequence

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

corresponding to the values of the signal waveform at times t = 0, h, 2h, 3h, ... (where t = 0 is the instant at which sampling begins).

At time t = nh (where n is some positive integer), the values available to the processor, stored in memory, are

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

Note that the sampled values xn+1, xn+2 etc. are not available as they haven't happened yet!

The digital output from the processor to the DAC consists of the sequence of values

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

In general, the value of yn is calculated from the values $x_0, x_1, x_2, x_3, \ldots , x_n$. The way in which the y's are calculated from the x's determines the filtering action of the digital filter.

*Examples of simple digital filters*

The following examples illustrate the essential features of digital filters.

1. UNITY GAIN FILTER: $y_n = x_n$

Each output value yn is exactly the same as the corresponding input value xn:

$y_0 = x_0$
$y_1 = x_1$
$y_2 = x_2$
... etc

This is a trivial case in which the filter has no effect on the signal.

2. SIMPLE GAIN FILTER: $y_n = Kx_n$ (K = constant)

This simply applies a gain factor K to each input value:

$y_0 = Kx_0$
$y_1 = Kx_1$
$y_2 = Kx_2$
... etc

$K > 1$ makes the filter an amplifier, while $0 < K < 1$ makes it an attenuator. $K < 0$ corresponds to an inverting amplifier. Example (1) above is the special case where $K = 1$.

3. 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.

4. 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

i.e. the output is the change in the input over the most recent sampling interval h. The effect of this filter is similar to that of an analog differentiator circuit.

5. TWO-TERM AVERAGE FILTER: $y_n = (x_n + x_{n-1}) / 2$

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

$y_0 = (x_0 + x_{-1}) / 2$
$y_1 = (x_1 + x_0) / 2$
$y_2 = (x_2 + x_1) / 2$
$y_3 = (x_3 + x_2) / 2$
... etc

This is a simple type of low pass filter as it tends to smooth out high-frequency variations in a signal. (We will look at more effective low pass filter designs later).

6. THREE-TERM AVERAGE FILTER: $y_n = (x_n + _{xn-1} + x_{n-2}) / 3$

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

$y_0 = (x_0 + x_{-1} + x_{-2}) / 3$
$y_1 = (x_1 + x_0 + x_{-1}) / 3$
$y_2 = (x_2 + x_1 + x_0) / 3$
$y_3 = (x_3 + x_2 + x_1) / 3$
... etc

As before, x-1 and x-2 are taken to be zero.

7. CENTRAL DIFFERENCE FILTER: $y_n = (x_n - x_{n-2}) / 2$

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:

$y_0 = (x_0 - x_{-2}) / 2$
$y_1 = (x_1 - x_{-1}) / 2$
$y_2 = (x_2 - x_0) / 2$
$y_3 = (x_3 - x_1) / 2$
... etc

*Order of a digital filter*

The order of a digital filter can be defined as the number of previous inputs (stored in the processor's memory) used to calculate the current output. This is illustrated by the filters given as examples in the previous section.

Examples (1-2): These are zero order filters, since the current output depends only on the current input and not on any previous inputs.

Example (3-5): These are first order filters, as one previous input (n-1) is required to calculate current sample. (Note that in (3) the filter is classed as first-order because it uses one previous input, even though the current input is not used).

Example (6-7): To compute the current output, two previous inputs are needed; this is therefore a second-order filter.

The order of a digital filter may be any positive integer. A zero-order filter is possible, but somewhat trivial, since it does not really filter the input signal in the accepted sense.

*Digital filter coefficients*

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

$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2}$

Similar expressions can be developed for filters of any order.

The constants $a_0$, $a_1$, $a_2$, ... appearing in these expressions are called the filter coefficients. The values of these coefficients determine the characteristics of a particular filter.

*Recursive and non-recursive filters*

For all the examples of digital filters discussed so far, the current output is calculated solely from the current and previous input values. This type of filter is said to be non-recursive.

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

The word recursive literally means "running back", and refers to the fact that previously-calculated output values go back into the calculation of the latest output. The expression

for a recursive filter therefore contains not only terms involving the input values (xn, xn-1, xn-2, ...) but also terms in yn-1, yn-2, ...

*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 by adding the current input (xn) to the previous output.

Thus:

y0 = x0 + y-1
y1 = x1 + y0
y2 = x2 + y1
y3 = x3 + y2
... etc

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 yn-1 the value given by the previous expression, we get the following:

$y_0 = x_0 + y_{-1} = x_0$
$y_1 = x_1 + y_0 = x_1 + x_0$
$y_2 = x_2 + y_1 = x_2 + x_1 + x_0$
$y_3 = x_3 + y_2 = x_3 + x_2 + x_1 + x_0$
... etc

Thus we can see that yn, the output at t = nh, is equal to the sum of the current input $x_n$ and all the previous inputs. This filter therefore sums or integrates the input values, and so has a similar effect to an analog integrator circuit.

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.

*FIR and IIR filters*

Some people prefer an alternative terminology in which a non-recursive filter is known as an FIR (or Finite Impulse Response) filter, and a recursive filter as an IIR (or Infinite

Impulse Response) filter. These terms refer to the differing "impulse responses" of the two types of filter

The impulse response of a digital filter is the output sequence from the filter when a unit impulse is applied at its input. (A unit impulse is a very simple input sequence consisting of a single value of 1 at time t = 0, followed by zeros at all subsequent sampling instants). An FIR filter is one whose impulse response is of finite duration. An IIR filter is one whose impulse response (theoretically) continues for ever, because the recursive (previous output) terms feed back energy into the filter input and keep it going. The term IIR is not very accurate, because the actual impulse responses of nearly all IIR filters reduce virtually to zero in a finite time. Nevertheless, these two terms are widely used.

*Order of a recursive (IIR) digital filter*

The order of a digital filter was defined earlier as the number of previous inputs that have to be stored in order to generate a given output. This definition is appropriate for non-recursive (FIR) filters, which use only the current and previous inputs to compute the current output. In the case of recursive filters, the definition can be extended as follows:

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

For example, the recursive filter discussed above, given by the expression

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

is classed as being of first order.

*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 outputs ($y_{n-1}$, $y_{n-2}$ etc.).

The general form of recursive filter with M recursive (output) and N direct (input) elements is

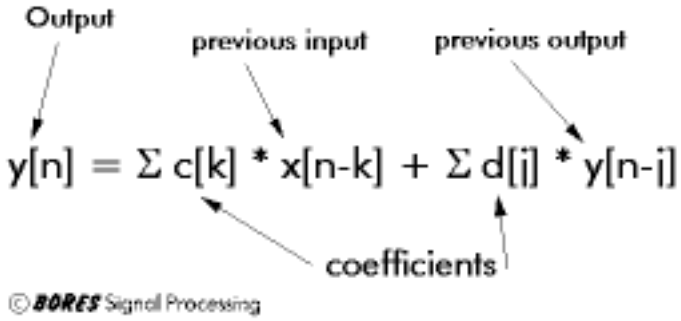
$$b_0y_n + b_1y_{n-1} + \ldots + b_My_{n-M} = a_0x_n + a_1x_{n-1} + a_Nx_{n-N}$$

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.

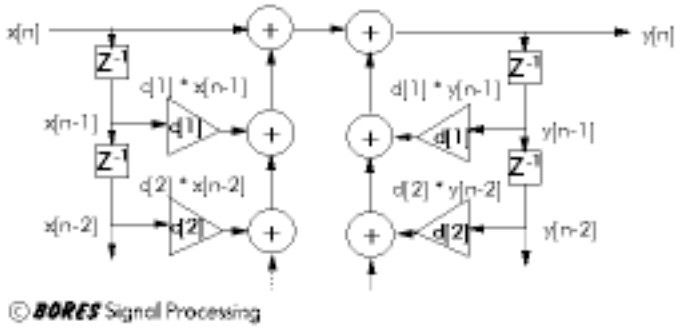
The order of IIR filter is max(N,M)

*Implementation of Digital Filters*

Output from a digital filter is made up from previous inputs and previous outputs. This operation can be written as weighted moving average between filter coefficients and a signal flipped back to front. This summation is called the operation of *convolution*




The filter can also be drawn as a following block diagram




The filter diagram can show what hardware elements will be required when implementing the filter:

Commercial DSP chips are specifically designed to perform fast buffering, additions and multiplications of multiple data in one clock cycle. The precision of the output depends on the precision of the numerical representation of the coefficients, the samples and the operation results. Finite precision of these elements leads to quantization errors. So apart from errors when measuring signals, the following errors arise within the hardware used for processing:

- errors in arithmetic within the hardware (for example 16 bit fixed point roundoff)
- trncation when results are stored (most DSP processors have extended registers internally, so truncation usually occurs when results are stored to memory)
- quantisation of filter coefficients which have to be stored in memory

## *B. The transfer function of a digital filter*

In the last section, we used two different ways of expressing the action of a digital filter: a form giving the output yn directly, and a "symmetrical" form with all the output terms (y's) on one side and all the input terms (x's) on the other.

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. The transfer function of a filter can be used to determine many of the characteristics of the filter, such as its frequency response.

*The unit delay operator*

First of all, we must introduce the unit delay operator, denoted by the symbol

$z^{-1}$

When applied to a sequence of digital values, this operator gives the previous value in the sequence. It therefore in effect introduces a delay of one sampling interval.

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

$x_0 = 5$
$x_1 = -2$
$x_2 = 0$
$x_3 = 7$
$x_4 = 10$

Then

$z^{-1} x_1 = 5$
$z^{-1} x_2 = -2$
$z^{-1} x_3 = 0$

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. We adopt the convention (soon to be explained why)

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

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 expressed using the delay operator given in its symmetrical form

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

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

$y_n / x_n = (a_0 + a_1 z^{-1} + a_2 z^{-2}) / (b0 + b_1 z^{-1} + b_2 z^{-2})$

This is the general form of the transfer function for a second-order recursive (IIR) filter.

*Transfer function examples*

1. The three-term average filter, defined by the expression

   $y_n = 1/3 (x_n + x_{n-1} + x_{n-2})$

   can be written using the z-1 operator notation as

$y_n = 1/3\ (x_n + z^{-1}x_n + z^{-2}x_n)$

$= 1/3\ (1 + z^{-1} + z^{-2})\ x_n$

The transfer function for the filter is therefore

$y_n / x_n = 1/3\ (1 + z^{-1} + z^{-2})$

2. The general form of the transfer function for a first-order recursive filter can be written

$y_n / x_n = (a_0 + a_1z^{-1}) / (b_0 + b_1z^{-1})$

Exercise: Write done transfer function of a simple first-order recursive filter

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

3. In MATLAB it is common to express the coefficients as vectors A and B. As a further example, consider the second-order IIR filter with coefficients B = [1 2 -1], A=[1 2 1].

Expressing this filter in terms of the delay operator gives

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

and so the transfer function is

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

*Delaying a phasor*

Filters operate by combining delayed version of input and output signals. The operation of a delay line is best understood using a complex periodic signal called phasor, expressed in terms of continuous time parameter *t* (instead of index n)

$e^{j\omega t}$

This is a rotating vector on the unit circle (complex number of magnitue 1) that completes one full cycle in $T = 2\pi/\omega$ seconds. Note that we express here the imaginary unit number by $j = \sqrt{-1}$. (It is commonly expressed also using the letter *i,* so we might use *i* or *j* interchangeably)

If we delay the phase by $\tau$ seconds, we get

$$e^{j\omega(t-\tau)} = e^{-j\omega\tau} e^{j\omega t}$$

Note that $\tau$ in the exponent represents a fixed delay time. It should not be confused with the time parameter *t* in the exponent of the phasor that keeps increasing (time goes on). A delay then becomes a multiplication by a constant complex number $e^{j\omega\tau}$ .

In case of a single sample delay, the multiplicative delay factor is written using the symbol z

$$z = e^{-j\omega}$$

Example:

For a FIR order one filter

$$y_t = a_o x_t + a_1 x_{t-1}$$

the effect of applying a delay to input that is a phasor

$$x_t = e^{-j\omega t}$$

becomes

$$y_t = [a_0 + a_1 e^{-j\omega}] x_t$$

The multiplicative expression

$$H(z) = a_0 + a_1 e^{-j\omega}$$

is called the filter *Transfer Function.*

*Describing filters in terms of Transfer Function.*

Filters are made of weighted sums of delayed input and output samples. When filter is applied to a general signal, we used the unit delay notation to represent the filter in an elegant way as weighted sums of repeated unit delay operations.

So far the unit delay appeared as a new mathematical operation that alters the time index *n* of input or output samples (x(n) and y(n)). As we just saw, in the special case of a phasor signal the delay actually becomes a multiplication by a complex number that has a unit magnitude and whose angle is equal to the amount of time delay that is being applied to that phasor.

What is interesting and important in understanding the effect of a filter on a phasor is that the now the operation of any filter on any signal can be described in terms of transfer function. How?

Using transfer function, the effect of the filter on a phasor is written mathematically in terms of multiplication of the phasor by complex polynomial or by a ratio of complex polynomials. The coefficients of this polynomial are equal to coefficients of the filter, with forward filter coefficients appearing at the numerator and backward (recursive) filter coefficients at the denominator, and the delays are the negative powers of z.

When considering some other signal, it still can be described in terms of a combination of complex phasors. This is done by writing any signal in terms of a weighted sum of phasors and different frequency. The coefficients (amplitudes and phases) of these phasors are found by transforming the signal into a complex domain using the Fourier Transform. In the frequency domain, the delay operation becomes multiplication of each phasor component by a transfer function. This is also called the *system Z-transform*.

In summary: Using Fourier Transform allows any signal in terms of combination of phasor. This allows expressing the output of a filter in terms of a multiplication between signal input frequency elements and the filter transfer function. The filtering operation can be described in the frequency domain as weighted combination of delays of the phasor components of the input x(n) signals.

We will learn about Fourier Transforms in the next section.

*The z-plane*
Once the Z-transform of a system has been determined, one can use the information contained in function's polynomials to graphically represent the function and easily observe many defining characteristics.

$$H(z) = \frac{P(z)}{Q(z)}$$

*Zeros*

1. The value(s) for z where P(z)=0
2. The complex frequencies that make the overall gain of the filter transfer function zero.

*Poles*

1. The value(s) for z where Q(z)=0
2. The complex frequencies that make the overall gain of the filter transfer function infinite.

Example 1

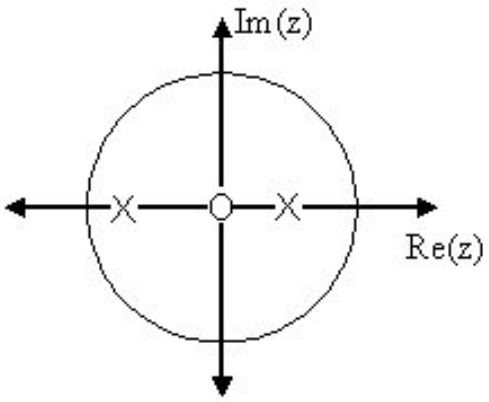
This is a simple transfer function with the poles and zeros shown below it.

$$Hz=(z+1)/[(z-1/2)(z+3/4)]$$

The zeros are: {-1}, The poles are: {1/2, -3/4}

Once the poles and zeros have been found for a given Z-Transform, they can be plotted onto the Z-Plane. The Z-plane is a complex plane with an imaginary and real axis referring to the complex-valued variable z. The position on the complex plane is given the radio and the angle from the positive, real axis around the plane. When mapping poles and zeros onto the plane, poles are denoted by an "x" and zeros by an "o".

The below figure shows the Z-Plane, and examples of plotting zeros and poles onto the plane can be found in the following section.




Examples of Pole/Zero Plots

Plotting z-plan in MATLAB

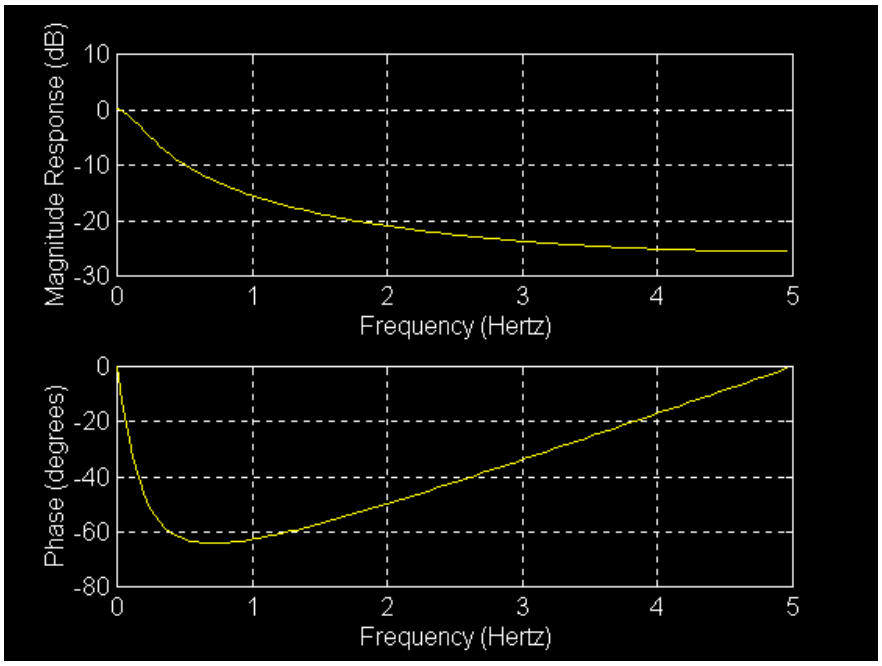
```
% Set up vector for zeros
z = [j ; -j];

% Set up vector for poles
p = [-1 ; .5+.5j ; .5-.5j];

figure(1);
zplane(z,p);
title('Pole/Zero Plot for Complex Pole/Zero Plot Example');
```

Frequency Response and the Z-Plane

The reason it is helpful to understand and create these pole/zero plots is due to their ability to help us easily design a filter. Based on the location of the poles and zeros, the magnitude response of the filter can be quickly understood. Also, by starting with the pole/zero plot, one can design a filter and obtain its transfer function very easily. Refer to this module for information on the relationship between the pole/zero plot and the frequency response.

*Estimating the Frequency Response of an IIR Filter in MATLAB*

```
%***Low Pass Averaging Filter***
%
% y(n) = y(n-1) + 0.1[x(n) - y(n-1)]
%
%H(Z) = 0.1Z/(Z - 0.9)
num = [0.1]
den = [1 -0.9]
Fs = 10
freqz(num, den, 128, Fs)
```




Additional reading: Steiglitz book 4.7, 4.8, 5.3, 5.4, 5.5, 5.7, 6.1