Imperial College London

Lecture 15

Impulse Response & FIR Filter

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

URL: www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/
E-mail: p.cheung@imperial.ac.uk

In this lecture, we will learn about:

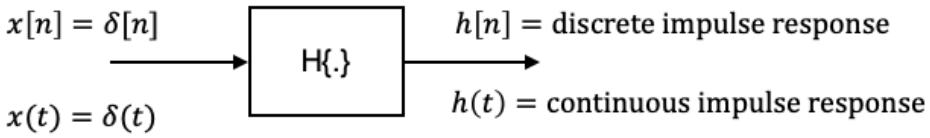
1. Impulse response of a discrete system and what it means.
2. How impulse response can be used to determine the output of the system given its input.
3. The idea behind convolution.
4. How convolution can be applied to moving average filter and why it is called a Finite Impulse Response (FIR) filter.
5. Frequency spectrum of the moving average filter
6. The idea of recursive or Infinite Impulse Response (IIR) filter.

I will also introduce two new packages for the Segway project:

1. mic.py – A Python package to capture data from the microphone
2. motor.py – A Python package to drive the motors

Impulse Response of a Discrete System

- The response of a discrete time system to a discrete impulse at the input is known as the system's **impulse response**



- The impulse response of a **linear** system completely define and characterise the system – both its transient behaviour and its frequency response.
- This applies to both continuous time and discrete time linear systems.
- An impulse signal contains ALL frequency components (L3, S7). Therefore, applying an impulse to input stimulates the systems at ALL frequencies.
- Since integrating a unit impulse = a unit step function, we can obtain the step response of the system by integrating the impulse response.

If we apply an impulse at the input of a system, the output response is known as the system's **impulse response**: h(t) for continuous time, and h[n] of discrete time.

We have demonstrated in Lecture 3 that the Fourier transform of a unit impulse is a constant (of 1), meaning that an impulse contains ALL frequency components. Therefore apply an impulse to the input of a system is like applying ALL frequency signals to the system simultaneously.

Furthermore, since integrating a unit impulse gives us a unit step, we can integrate the impulse response of a system, and we can obtain it's step response.

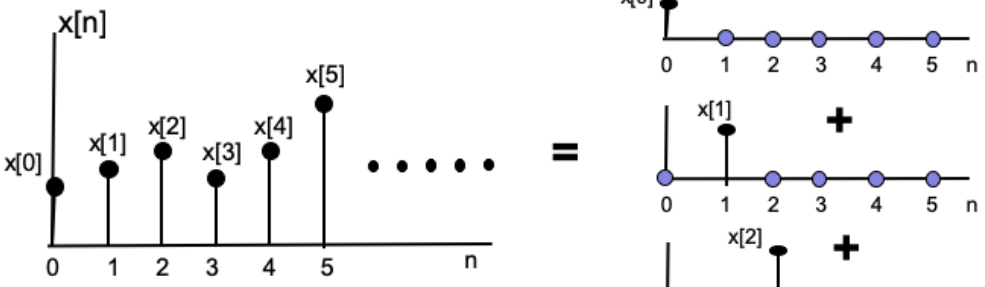
In other words, the impulse response of a system completely specify and characterise the response of the system.

So here are two important lessons:

1. Impulse response h(t) or h[n] characterizes a system in the time-domain.
2. Transfer function H(s) or H[z] characterizes a system in the frequency-domain.

Discrete signal as sum of weighted impulses

- Remember from L10, S7, we can represent a causal discrete signal *x[n]* in terms of sum of weighted delayed impulses:

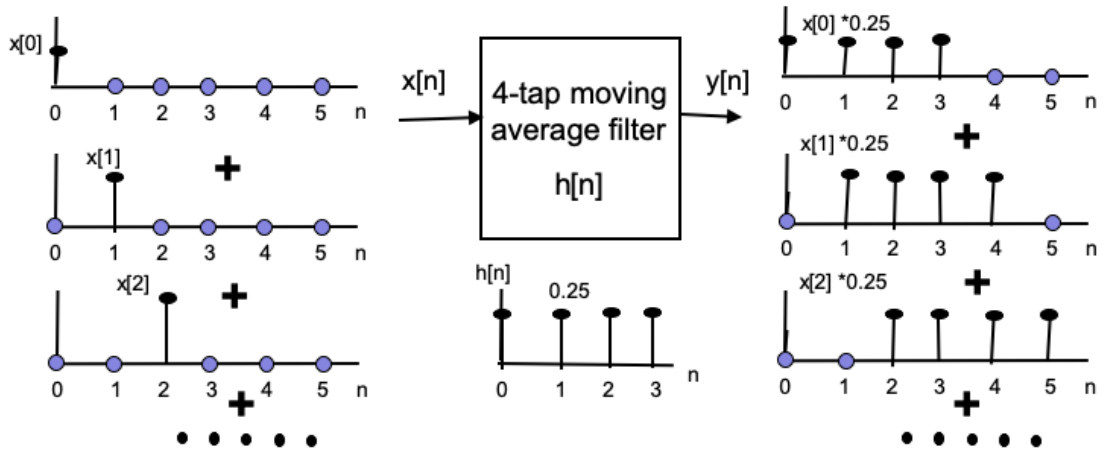


$$x[n] = \sum_{k=0}^{\infty} x[k]\,\delta[n-k]$$

Mathematically, it is really useful to model a discrete time signal x[n] consisting of sample sequence: {x[0], x[1], x[2] …} in terms of the unit impulse with different delays. We have already seen this in Lecture 11 before.

The plots on the right is the sequence x[n] decomposed into a sum of delay impulses at each sampling point, each being weighted by the signal x[n].

Impulse Response and Convolution



- We can therefore derive the output of a discrete linear system, by adding together the system's response to EACH input sample separately.
- This operation is known as **convolution**:

$$y[n] = x[n] * h[n] = \sum_{m=0}^{\infty} x[m]h[n-m]$$

PYKC 20 Feb 2020 DE2.3 – Electronics 2 Lecture 15 Slide 4

Therefore when considering the response of a discrete system to any arbitrary input, it is useful to think of the inputs as a sum of lots of weighted impulses (delayed).

Let us consider the simple 4-tap moving average filter, whose impulse response h[n] is as shown.

x[0] sample will produce a response on the right side as shown. Similarly, we apply x[1], x[2] etc.

Now the output y[n] is just the sum of all the responses to each impulses.

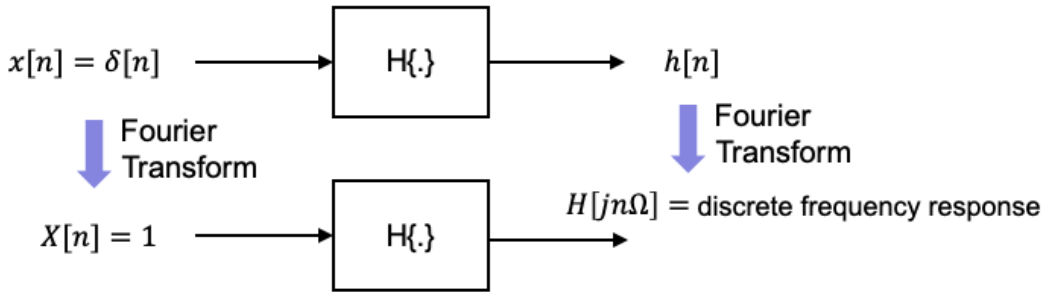
This operate of sum of product between x[.] and h[.], is represented mathmetically as (assume causality):

$$y[n] = x[n] * h[n] = \sum_{m=0}^{\infty} x[m]h[n-m]$$

This operation indicated by the '*' operator is known as convolution.

Impulse Response & Frequency Response

- Since a unit impulse contains all frequency, and its Fourier transform is a constant at 1 (see L3, S7), the Fourier transform of the impulse response h[n] or h(t) give us the systems' frequency response:

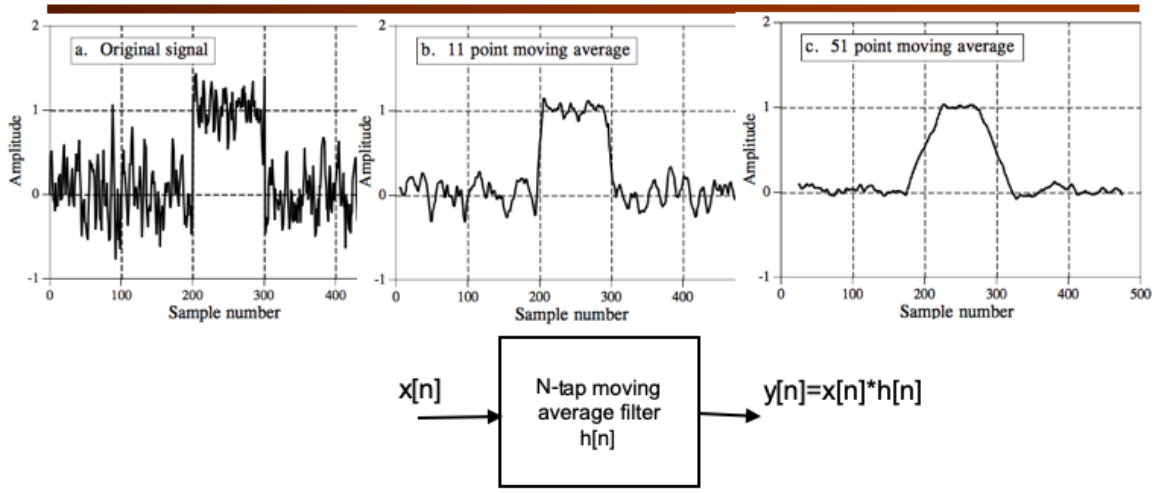


- This applies to both continuous time and discrete time linear systems.
- An impulse signal contains ALL frequency components (L3, S7). Therefore, applying an impulse to input stimulates the systems at ALL frequencies.

We have already seen that a unit impulse contains all frequency components. Therefore it is obvious that the impulse response is the system's response to ALL possible frequencies stimuli at the input.

This leads to the interesting result that the Fourier transform of the impulse response of a system give us the system's frequency response. This fact follows the argument that the Fourier transform is a linear transformation.

Moving Average Filter = FIR lowpass filter



- N-tap (or N point) moving after filter – high N, lower the cut off frequency
- For a N-tap moving average filter, it impulse response has N impulses.
- If input x[n] has M non-zero samples (i.e. finite length), output y[n] is also finite in length, and has M+N non-zero samples. Hence the name Finite Impulse Response (FIR) filter.

Now let us consider some real discrete time system - the moving average filter.

We have seen in the last lecture and in Lab 5 that by averaging current input and N-1 previous input samples to produce current output has a low pass filtering effect (which is an averaging function). Here we show that a noisy signal being filtered by a 11-tap and 51-tap moving average filter.

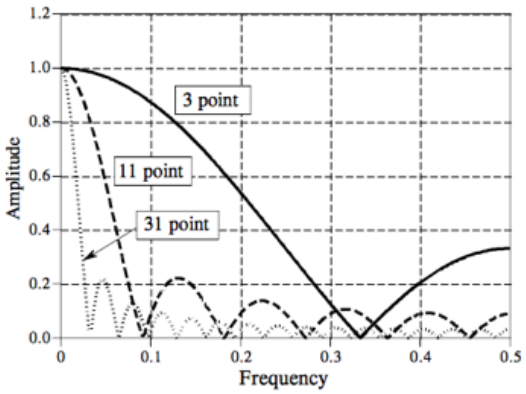
The procedure you followed in Lab 5 Exercise 4 is effectively performing the convolution operation: $y[n] = x[n]*h[n]$.

Frequency Response of N-tap moving average filter

- The impulse response of a moving average filter is a rectangular pulse.
- The Fourier transform of a rectangular pulse is of the form (sin x)/x or sinc(x) function (see Lecture 3 slide 6) in the case of continuous time.
- For discrete time case, the frequency response of a moving average filter with N taps (or points) is:

$$H[f] = \frac{\sin(fN)}{N \sin f}$$

- Here *f* is normalised to 0 -> 0.5 x sampling frequency *fs*.



Remember that the frequency response of a system can be found by taking the Fourier transform of the impulse response.

The moving average filter has an impulse response = rectangular function rect(.).

From Lecture 3, slide 6, we have learned that the Fourier transform of a rectangular function is of the form of sin(x)/x, (or sinc(x)). Shown here is the frequency response of the moving average filter for different number of taps.

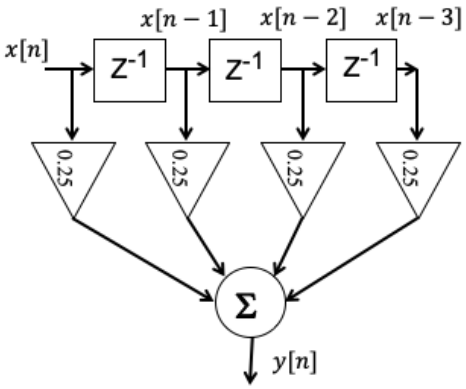
Note that the frequency axis is normalised (i.e. divided by) the sampling frequency fs. This is often the case in discrete time domain, we consider frequency in this way, as a ratio to the sampling frequency.

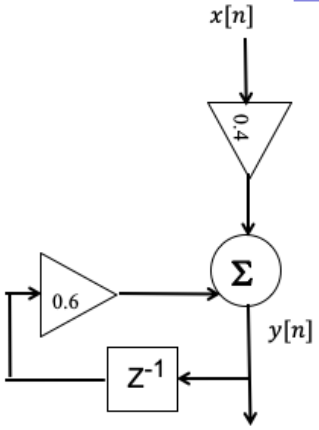
Recursive or Infinite Impulse Response Filter

FIR

x[n] =
{1.0, 1.0, 1.0, 1.0, 1.1, 0.8, 1.2, 0.9, 1.0, 1.2, 0.9, …}

IIR





y[n] = (0.25, 0.5, 0.75, 1.0, 1.025,
0.975, 1.025, 1.0, 1.0, 1.075, 1.0, …..}

y[n] = (0.4, 0.64, 0.784, 0.87, 0.962,
0.897, 1.018, 0.971, 0.983, 1.07,
1.002…..}

FIR filters are also known as **non-recursive filter** because output sample y[n] are ONLY depended on current input sample x[n], and previous input samples x[n-1], x[n-2] ….

FIR filters require lots of multiplications and additions. Based on the convolution formula, a N-tap FIR filter in general would need N multiples and N adds.

A more computationally efficient approach would be to derive output samples y[n] from both current and past input samples, x[n], x[n-1], .. As well as previous outputs y[n-1], y[n-2] etc.

These type of filters are known as **recursive filters** or **Infinite Impulse Response** (IIR) filter.

It is interesting to consider the response of the FIR and the IIR filter to the input shown. In spite of a very simple structure (only 1 delay element, one multiply, and one add) of the recursive filter, it has an excellent low pass function as seen in the output sequence y[n].

Package mic.py to use the microphone

- The package mic.py (new) is available to help you to set up interrupt to sample signals from the microphone, and obtain the instantaneous energy.
- You must first import the package, and then create the mic object:

```
from mic import MICROPHONE
# Create microphone object
SAMP_FREQ = 8000
N = 160
mic = MICROPHONE(Timer(7,freq=SAMP_FREQ),ADC('Y11'), N)
```

- This defines Timer 7 is used to define sampling frequency, which is set at 8kHz. Microphone is connected to the ADC input at pin 'Y11', and instantaneous energy is to be calculated over N=160 samples.
- Thereafter, you can use the following methods:
- This package will handle or interrupt service and computer the energy value over N samples.
- The buffer_full flag is used to indicate when 20msec has elapsed and the inst_energy() method can be used.

Method	Description
mic.buffer_full( )	Return True if N samples taken
mic.set_buffer_empty( )	Reset buffer_full flag
mic.inst_energy( )	Return instantaneous energy for N samples

PYKC 20 Feb 2020 DE2.3 – Electronics 2 Lecture 15 Slide 9

If you have completed Lab 5, you would have written Python code to capture 160 samples of microphone data at a sampling rate of 8kHz. You would also have learned how to use interrupt to make this audio signal acquisition "transparent" to your main program. That is, the interrupt, once set up, will do the sampling and storing automatically in the background. Therefore, your main program can do other things, such as displaying the captured data on the OLED display. The data is passed to the main program via two variables: a) s_buf, which is the signal buffer (an array) with 160 values; b) buffer_full, which is set True once the full 160 values are captured.

Throughout the Lab sessions, you have used many other packages written by me or by others. The mic.py package here is to show you HOW you can write your own Python package, so that you can define your own objects and the corresponding methods.

It is therefore advantageous for you to study the mic.py package provided, and relate this to the code you used in Lab 5. You should understand how objects are created (via constructor) and methods are defined.

Package to drive motors

- The package **motor.py** is available to help you drive the two motors with ease. It will make developing your milestone code much easier.
- You must first import the package, and then create the motor object:

```
from motor import MOTOR
# Create motor object for the two motors
motor = MOTOR()
```

- Thereafter, you can use the following methods:
- The first five methods are useful to control speed of the motors using the CONTROL PAD via Bluetooth
- The last six methods are directly controlling the movements of the two motors (in an open-loop manner)
- v is not really the speed, but the PWM drive value (i.e. duty cycle) to the motors.

Method	Description
motor.up_Aspeed(v)	increase motor A speed by v
motor.up_Bspeed(v)	increase motor B speed by v
motor.dn_Aspeed(v)	Reduce motor A speed by v
motor.dn_Bspeed(v)	Reduce motor B speed by v
motor.drive( )	Drive motors at their set speeds
motor.A_forward(v)	Drive motor A forward at v
motor.B_forward(v)	Drive motor B forward at v
motor.A_back(v)	Drive motor A backward at v
motor.B_back(v)	Drive motor B backward at v
motor.A_stop( )	Stop motor A
motor.B_stop( )	Stop motor B

I have also provided you with another package motor.py, which control the speed of the motors.

Both packages can be downloaded from the course webpage.