

**كلية الهندسة - جامعة الزقازيق**  
**قسم هندسة الحاسبات والمنظومات**  
**الفصل الدراسي الثاني**  
**2022/2023**

**الفرقة: الثالثة هندسة الحاسبات والمنظومات**

**المقرر: القياسات والإختبارات**

**اسم التجربة: Digital Signal Processing**

**رقم التجربة: 7**

**اسم المعاون المسئول: أسامة سمير محمد عبداللطيف**

## Objectives:

- To deal with audio signals in MATLAB/Octave.
- To calculate frequency spectrum of digital signal in MATLAB/Octave.
- To cover some basic concepts of digital filters.
- To introduce the concepts of FIR filter design.
- To introduce the concepts of IIR filter design.

## 1. Obtain signal to work with in MATLAB/Octave:

### 1. Generate sinusoidal signal:

Since we're going to be looking at the functions as time signals, let's start by creating a time base, a vector containing the value of  $t$  at each sample interval in our *three* seconds signal.

```
Fs = 8000;           % sampling frequency
time = 3;            % seconds
t = 0:(1/Fs):time;   % discretize time
N = length(t);       % #samples
f0 = 500;            % signal frequency (Hz)
x = 0.1*cos(2*pi*f0*t); % scaling by 0.1
```

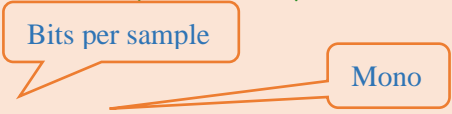
### 2. Generate random signal:

Random signal is just a white noise containing all frequencies equally.

```
Fs = 8000;           % sampling frequency
time = 3;            % seconds
N = time*Fs;         % #samples
x = rand(1,N)*2-1;    % Uniformly
distributed..         % .. random signal
t = linspace(0,time,N); % discretize time
```

### 3. Record voice from Microphone:

```
% Preferred to use Fs = 8000, 11025, 22050, 44100
Hz
Fs = 8000;
time = 3;
recObj = audiorecorder(Fs, 16, 1);
recordblocking(recObj, time); %stop prog. &
record
x = getaudiodata(recObj);
N = length(x);           % #samples
t = linspace(0,time,N);  % discretize time
```



#### 4. Read audio file:

If you have an audio file named *file0.wav* and you want to read it into MATLAB to work with.

```
[x,Fs] = audioread('file0.wav');  
N = length(x);  
time = N/Fs;  
t = linspace(0,time,N);           % discretize time
```

#### Note that:

To write an audio data to a file you shall use this command.

```
audiowrite('file1.wav',x,Fs);
```

## 2. Play the signal

```
nBits = 16;  
sound(x,Fs,nBits)           % Convert signal data to  
sound
```

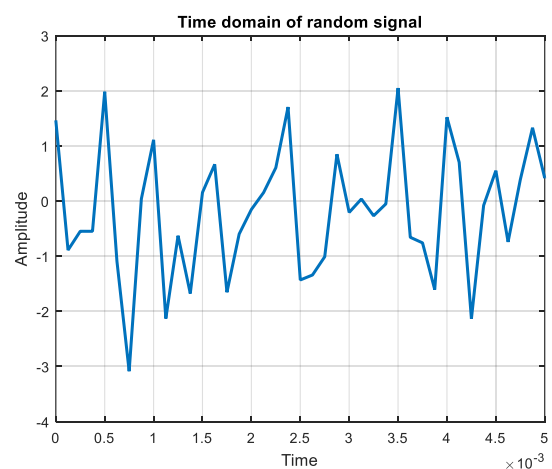
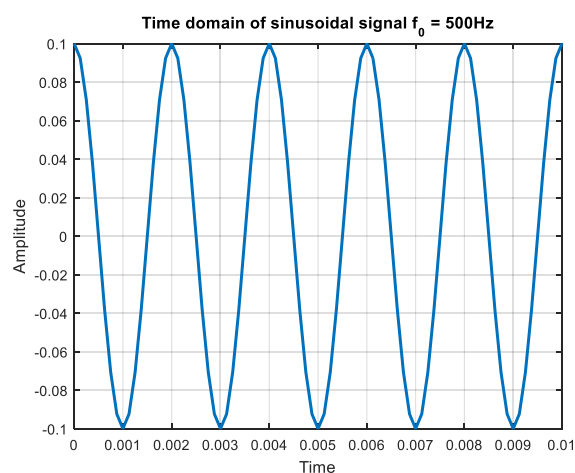
#### Note that:

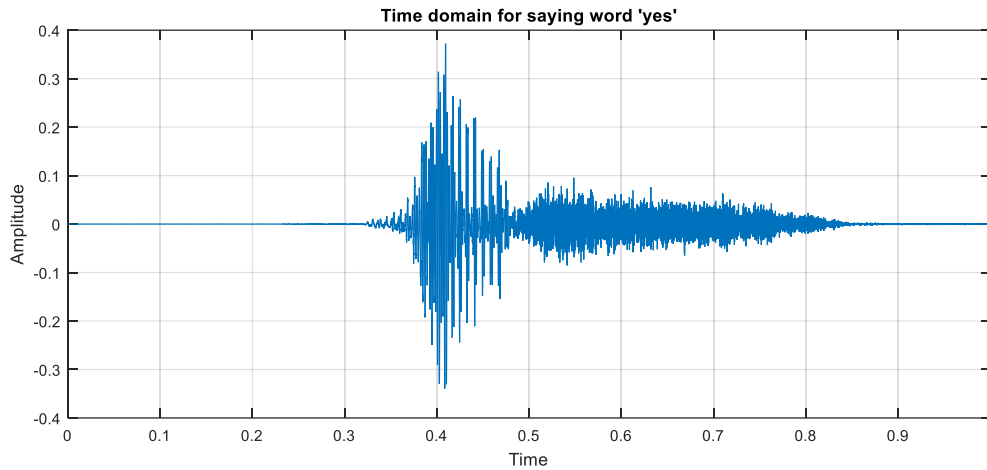
If you have an *audiorecorder* object you can play it directly.

```
play(recObj);
```

## 3. Plot it in time domain:

```
figure(1);plot(t,x),grid;  
xlabel('Time')  
ylabel('Amplitude')  
title('Time domain representation of x(t)')
```





#### 4. Get FFT and plot absolute spectrum:

##### What is FFT?

The FFT (Fast Fourier Transform) is an algorithm for efficient computation of the DFT (Discrete Fourier Transform). It is not a new transform.

Convert discrete periodic signal in time-domain  $x_d[n]$  to discrete periodic signal in frequency-domain  $X(k)$

##### Recall that:

DFT transform the signal from discrete time  $n$  to discrete frequency  $k$ :

$$X_N(k) = \sum_{n=0}^{N-1} x_d[n] e^{\frac{-j2\pi kn}{N}}$$

$N$ : No. samples.

$n$ : Discrete time.

$k$ : Discrete frequency.

$x_d[n]$ : Discrete time signal.

$X_N(k)$ : Discrete frequency signal.

Relation between discrete frequency  $k_0$  and continues frequency  $f_0$ :

$$\frac{k_0}{N} = \frac{f_0}{f_s}$$

##### Note that:

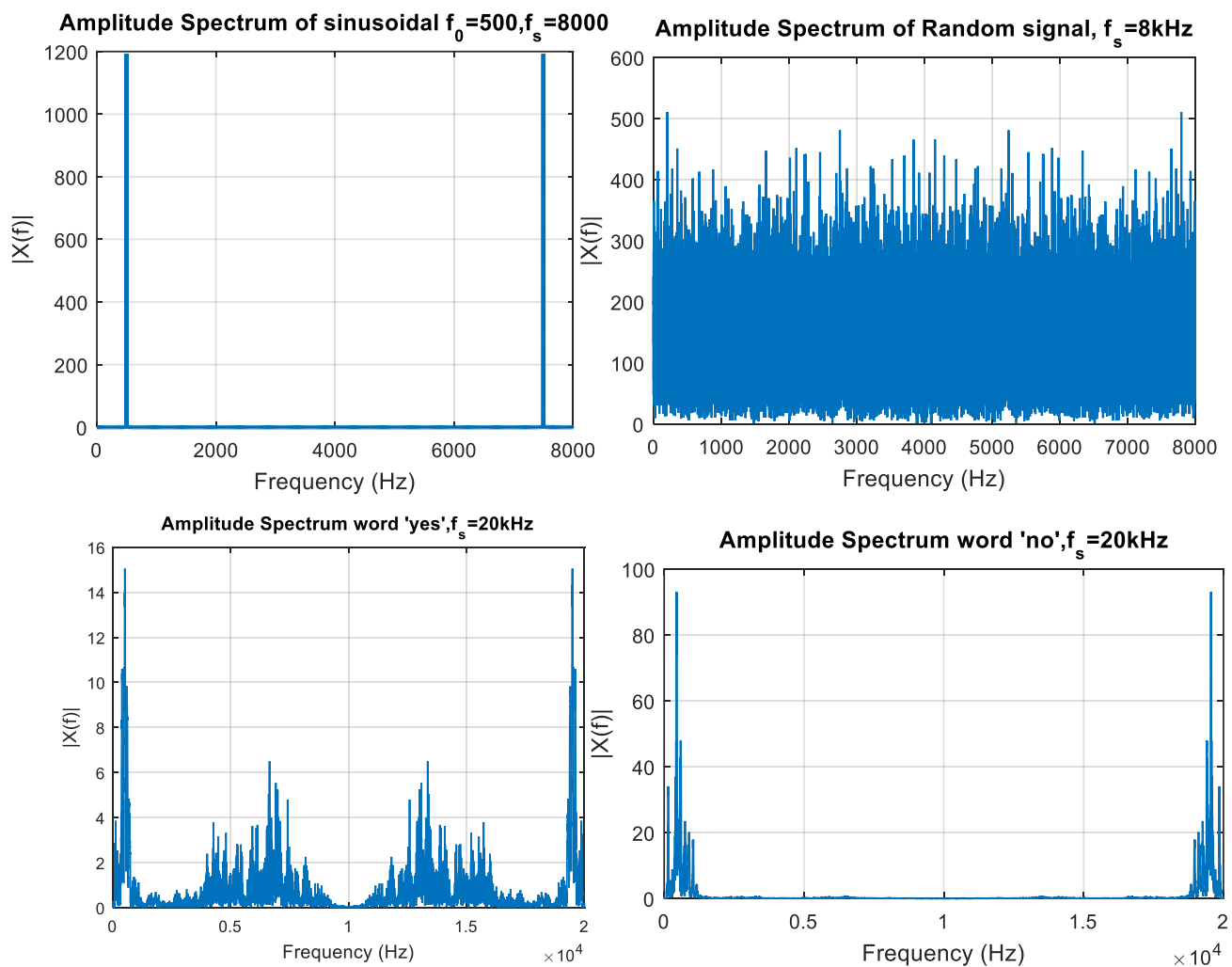
In CTFT  $X_s(f)$  is periodic over  $f_s$

In DTFT  $X_d(\omega)$  is periodic over  $2\pi$

In DFT  $X_N(k)$  is periodic over  $N$

In this code we want to plot  $|X_N(k)|$  vs. continuous frequency  $f$  instead of plotting it vs. discrete frequency  $k$  so it can be easy to read frequencies from the plot.

```
X_k = abs(fft(x)); % calculate absolute of fft
f = linspace(0,Fs,N); % discretize frequency
figure(2);plot(f,X_k),grid;
title('Amplitude Spectrum of x(t)')
xlabel('Frequency (Hz)')
ylabel('|X(f)|')
```



**Note that:**

For any real-valued signal  $|X_N(k)|$  is even so all frequency information can be extracted from the first half of the plot between  $f = 0$  to  $\frac{f_s}{2}$  and the second half is just a flipped repeat for the first half.

## 5. Simple filtering:

In digital signal processing applications, it is often necessary to change the relative amplitudes of frequency components or remove undesired frequencies of a signal. This process is called filtering. Digital filters are used in a variety of applications. We saw that digital filters may be used in systems that perform interpolation and decimation on discrete-time signals. Digital filters are also used in audio systems that allow the listener to adjust the bass (low-frequency energy) and the treble (high frequency energy) of audio signals.

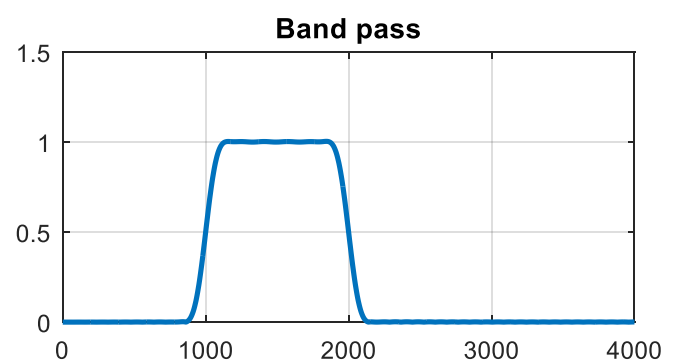
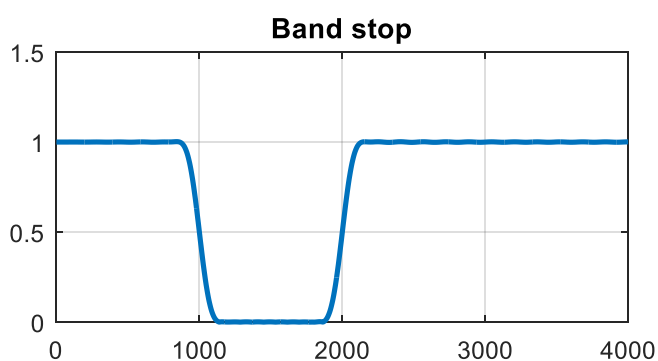
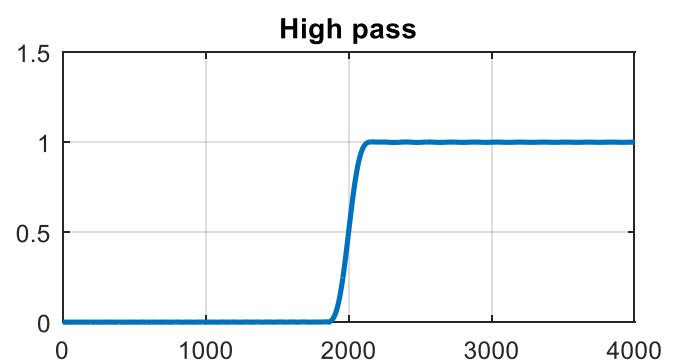
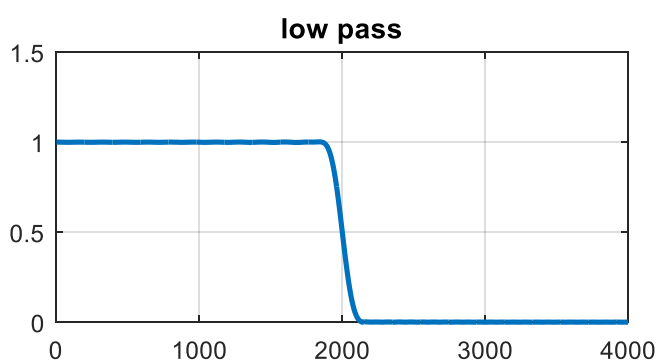
Digital filter design requires the use of both frequency domain and time domain techniques. This is because filter design specifications are often given in the frequency domain, but filters are usually implemented in the time domain with a difference equation.

Filtering a digital signal involves forming a weighted sum of the past input and output samples:

$$a_0 y(n) = \sum_{k=0}^N b_k x(n-k) - \sum_{k=1}^M a_k y(n-k)$$

Choosing the values of the  $a_k$  s and  $b_k$  s is the art of *filter design*.

## 6. Filter design: (causal LTI systems)



## 1. Finite impulse response:

A *finite impulse response* (FIR) or non-recursive filter has all the  $a_k$  (except  $a_0$ ) equal to zero, i.e. it is a weighted sum of input samples.

**E.g.1** LPF with order  $n = 3$  moving average filter:

$$y[n] = \frac{1}{4} (x[n] + x[n-1] + x[n-2] + x[n-3])$$
$$a = 1, \quad b = 0.25[1, 1, 1, 1]$$

**E.g.2** HPF with order  $n = 3$  moving average filter:

$$y[n] = \frac{1}{4} (x[n] - x[n-1] + x[n-2] - x[n-3])$$
$$a = 1, \quad b = 0.25[1, -1, 1, -1]$$

The code below calculate best  $b$  coefficient for FIR low pass filter:

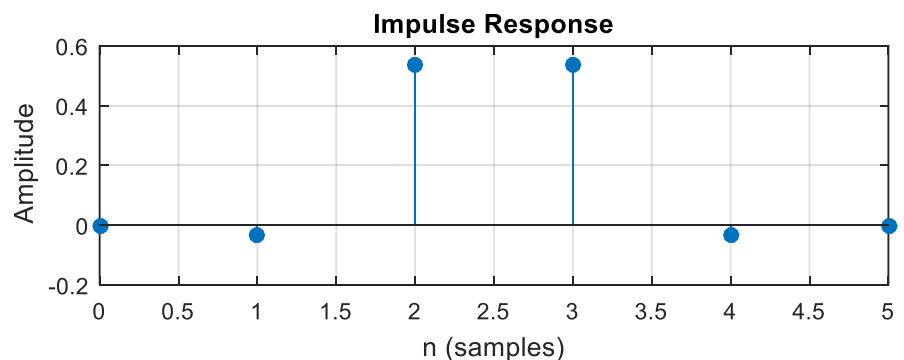
```
n = 5; % filter order
fc = 3000; % filter cutoff frequency
a = 1;
% Choose filter type
'low', 'high', 'bandpass', 'stop'
b = fir1(n, fc/(Fs/2), 'low');
%b = fir1(n, fc/(Fs/2), 'high');
%b = fir1(n, [500 1500]/(Fs/2), 'bandpass');
%b = fir1(n, [500 1500]/(Fs/2), 'stop');
```

After running MATLAB script we get the following equation:

$$y[n] = -0.0039 * x[n] - 0.0323 * x[n-1] + 0.5362 * x[n-2] + 0.5362 * x[n-3] - 0.0323 * x[n-4] - 0.0039 * x[n-5]$$

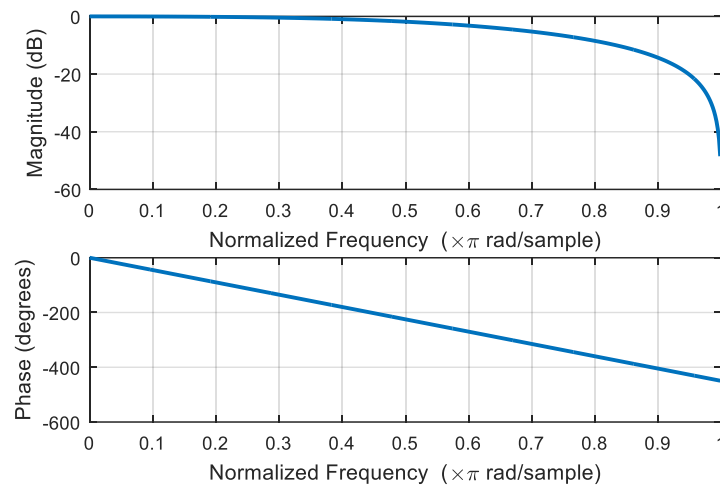
To get the impulse response and display it (note that it is finite )

```
figure(3);
impz(b, a);
grid;
```



To compute and plot the logarithmic scale frequency response of the filter:

```
figure(4);freqz(b, a),grid;
```

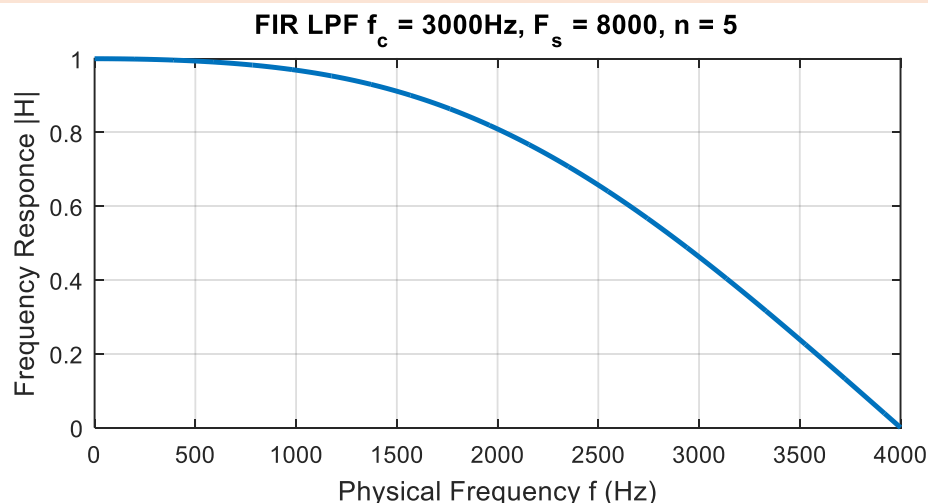


**Note that:**

- The frequency axis is labeled in terms of Normalized frequency  $\pi$ .

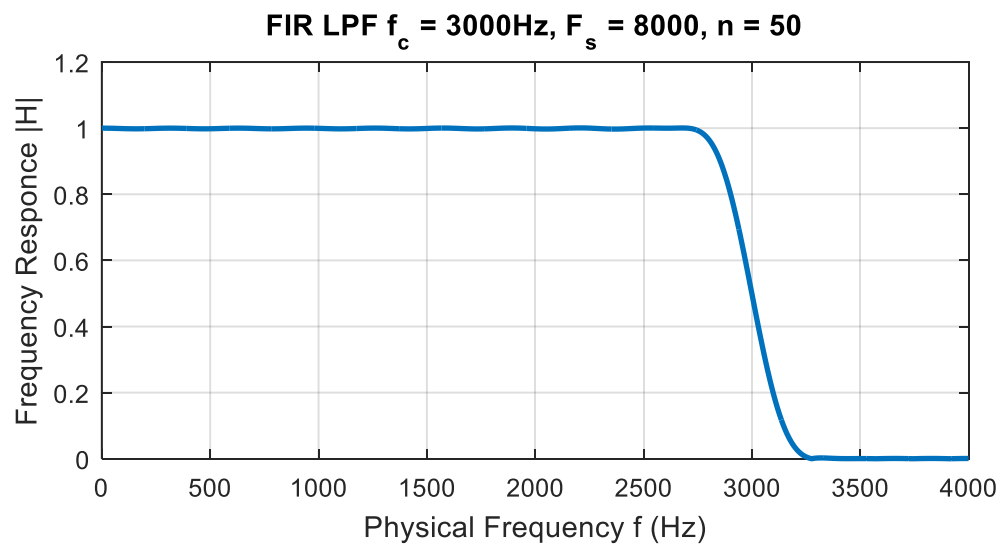
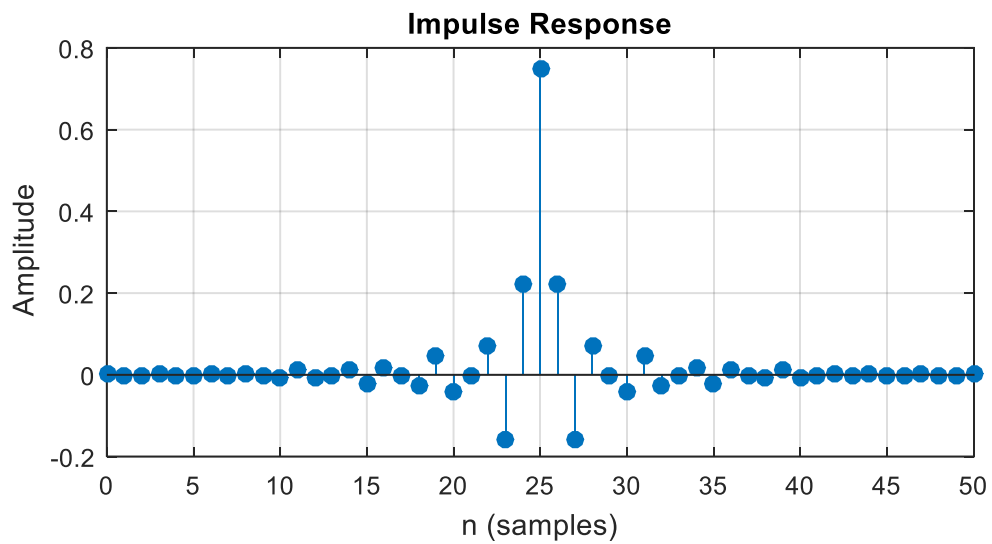
To compute and display the frequency response of a digital filter vs. continuous frequency  $f$  use this code below:

```
f = (0:.001:1)*Fs/2;  
H = freqz(b,a,f,Fs);  
figure(4);plot(f,abs(H)),grid;  
xlabel('Physical Frequency f (Hz)')  
ylabel('Frequency Response |H|')
```



FIR filters really start to get interesting when we build very long ones. Try the same cutoff frequency with  $n = 50$ .





But increasing order of FIR filter will increase the delay

## 2. Infinite impulse response:

An *infinite impulse response* (IIR) filter has a unit-sample response that doesn't go to zero after a finite number of samples. IIR filters are usually implemented *recursively*, i.e. with non-zero  $a_k$  coefficients

**E.g.1** LPF with 1<sup>st</sup> order

$$y[n] = x[n] + 0.5 * y[n - 1]$$

**E.g.2** HPF with 1<sup>st</sup> order

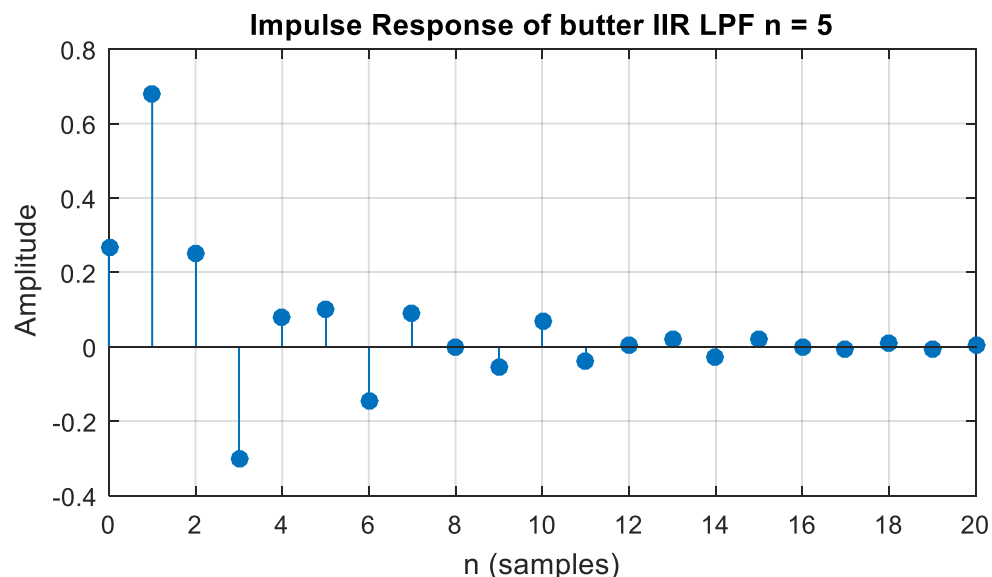
$$y[n] = x[n] - 0.9 * y[n - 1]$$

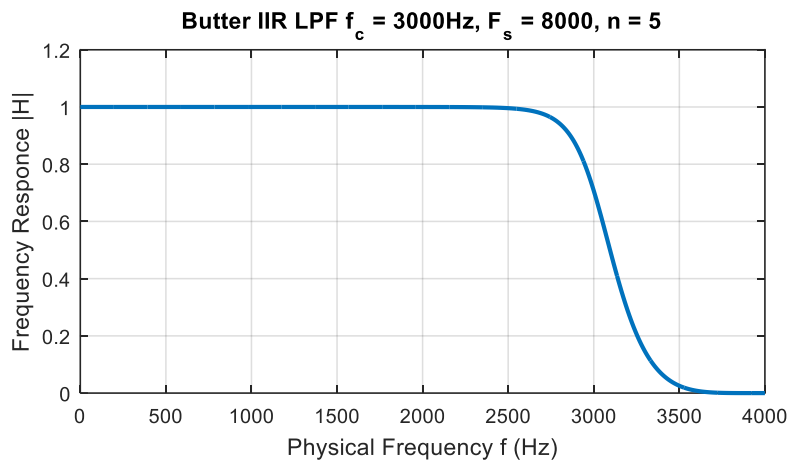
The **Butterworth filter** is one of the classic filter designs. It is characterized by having the "flattest" possible pass-band. Unfortunately, its performance in the stop band is decidedly mediocre.

```
n = 5; % filter order
fc = 1000; % filter cutoff frequency
[b,a] = butter(n, fc/(Fs/2), 'low');
```

After running MATLAB script we get the following equation:

$$\begin{aligned} y[n] = & 0.2689 * x[n] + 1.3447 * x[n - 1] + 2.6894 * x[n - 2] + 2.6894 \\ & * x[n - 3] + 1.3447 * x[n - 4] + 0.2689 * x[n - 5] \\ & - 2.4744 * y[n - 1] - 2.811 * y[n - 2] - 1.7038 \\ & * y[n - 3] - 0.5444 * y[n - 4] - 0.0723 * y[n - 5] \end{aligned}$$





**Note that:**

- Higher order FIR filters can be approximated to lower order IIR filters.
- Any FIR filter is stable but in IIR it may become unstable especially at higher orders so, IIR must be studied carefully.

**7. Filtering any signal using causal LTI system:**

MATLAB has a function for performing this operation:

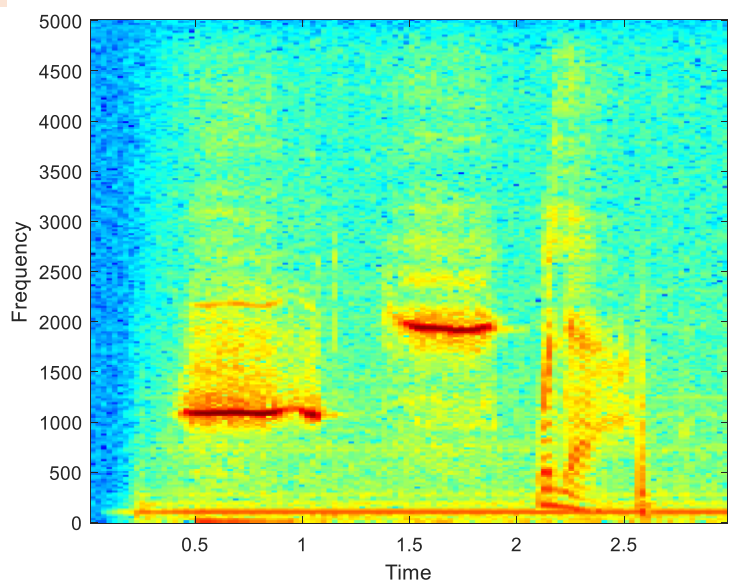
```
y = filter(b,a,x);
```

Then you can play the sound and plot its frequency spectrum.

**Plotting frequency domain vs time domain: (spectrogram)**

if you want to decide when a specific frequency appeared in the signal

```
specgram(x, 512, Fs)
```



## 8. Assignment:

1. For each of the following filters plot frequency responses, impulse responses (DON'T FORGET TO LABEL THE AXES AND TITLE YOUR PLOTS),  $F_s = 10$  kHz, **Is the system stable?**
  - Butterworth filter with  $n = 4,21$  (band pass  $f_{c1}=200$ ,  $f_{c2}=2000$ )
2. You have a file named “**whistle.wav**”:
  - Read the file into MATLAB, specify #samples and time of recording in sec.
  - Plot the frequency spectrum of signal **x**, do you notice the peaks?
  - Design a filter to reject the sinusoidal signals from signal **x**.
  - Plot frequency response, impulse response of the designed filter. Is the filter stable?
  - Plot the frequency spectrum of signal **y** (the output of the filter).
  - Play the output signal is the whistle still there?
  - Calculate the **energy** for the original signal and the filtered signal.
3. For each of the following filters plot frequency responses, impulse responses:
  - a.  $y[n] = \frac{1}{8} (x[n] + x[n-1] + x[n-2] + x[n-3] + x[n-4] + x[n-5] + x[n-6] + x[n-7])$
  - b.  $y[n] = \frac{1}{8} (x[n] - x[n-8]) + y[n-1]$Comment on the plots.