Signal Processing



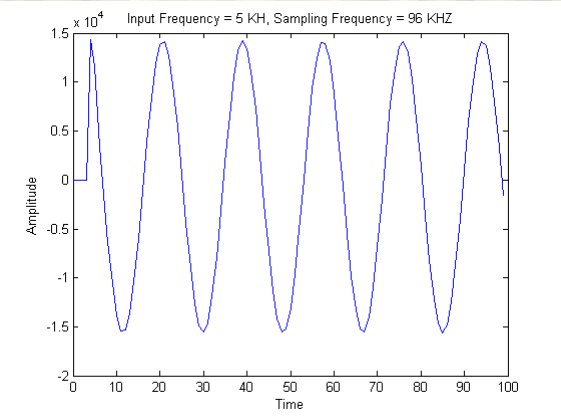
# Implementing Filter for a Noisy Input Signal

1

Signal Processing



## Sampling Frequency Importance

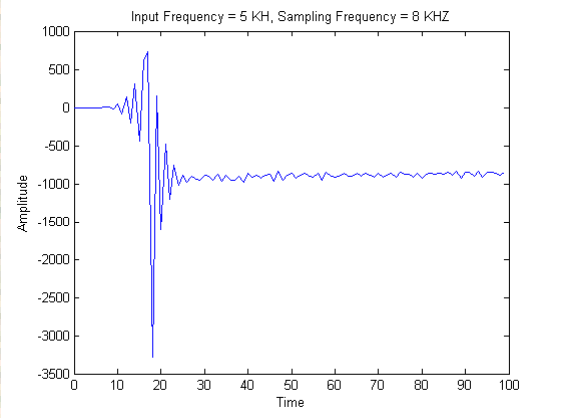



2

## Signal Processing



### Sampling Frequency Importance

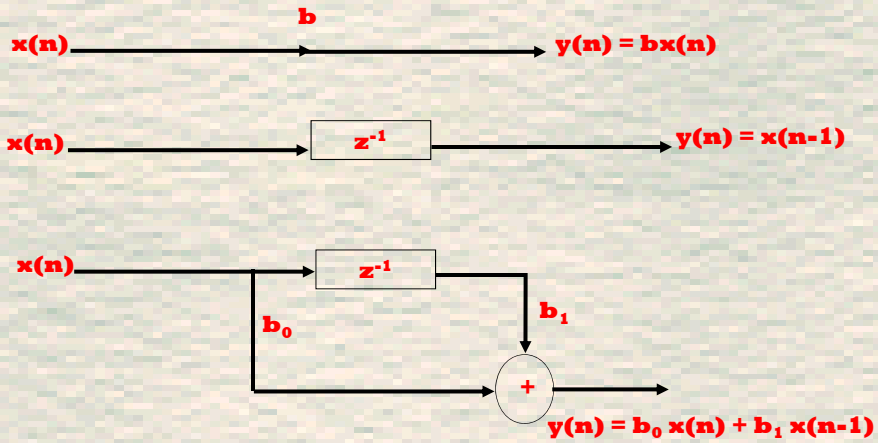



3

## Signal Processing



### Block Diagrams of Multiplication, Delay & simple DSP system




4

## Signal Processing



**The I/O difference equation of a DSP system defines the relationship between input and output.**

**The I/O equation consists of mathematical expression with addition, multiplication and delay.**

**$y(n) = b_0 x(n) + b_1 x(n-1)$**

**$b_0$ & $b_1$ are pre-determined coefficients.**

**A digital System can be described by I/O equation.**

5

## Signal Processing



**$y(n) = b_0 x(n) + b_1 x(n-1)$**

**This equation has a length of 2, however it can be generalized to a system with a length of L as,**

$$y(n) = \sum_{i=0}^{i=L-1} h(i)x(n-i) = \sum_{i=0}^{i=L-1} b_i x(n-i)$$

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + \ldots + b_{L-1} x(n-L+1)$$

6

## Signal Processing



**If we take the z-transform of this equation**

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{L-1} z^{-(L-1)} = \sum_{i=0}^{L-1} b_i z^{-i}$$

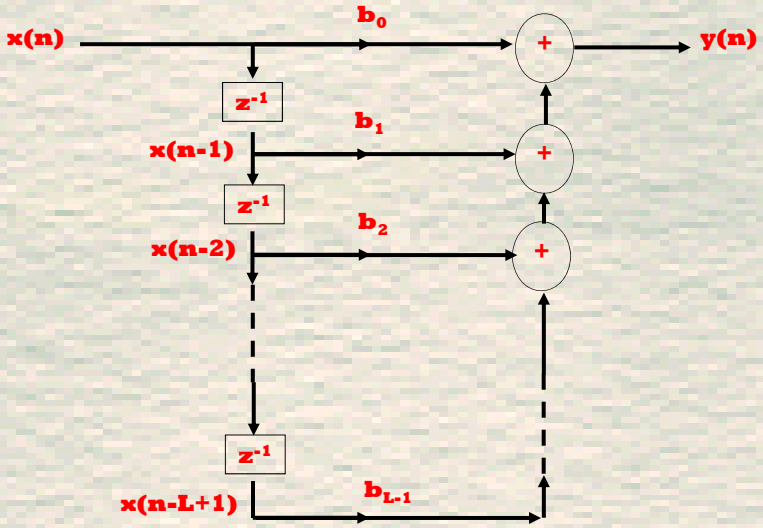
**This filter actually requires 2L memory space, for storing L samples and L filter coefficients. The signal buffer {x(n), x(n-1), x(n-2).....x(n-L+1)} is also called delay buffer or a tapped delay line which is implemented as a FIFO in memory.**

7

## Signal Processing



**FIR FILTER**




8

## Signal Processing



**Moving Average Filter**

$$y(n) = \frac{1}{L}\sum_{i=0}^{L-1} x(n-i) = \sum_{i=0}^{L-1} \frac{1}{L} x(n-i)$$

9

## Signal Processing



**% Generating Sine wave with noise**

```
N = 100;                                  % file length
n = [0:N-1];                              % time index
f = 500;                                  % frequency = 500 Hz
fs = 8000;                                % sampling rate = 8000 Hz
omega = 2*pi*f/fs;                        % w
x1n = 1200*sin(omega*n);                  % sinewave amplitude A = 1200
x2n = (rand(1,N)-0.5)*400;                % noise amplitude: -200 – 200
xn = round(x1n+x2n);                      % rounding to near integer
plot(n,xn,'--b');
fid = fopen ('noisysignal.dat','w');      % save signal to file noisysignal.dat
fprintf (fid,'%4.0f\n',xn);               % in integer format
fclose (fid);
```

10

## Signal Processing



```
% Moving-average filter

L = 4;                          % moving-average filter of order 4
load –ascii noisysignal.dat;    % load noisy sine wave
N = length(noisysignal);        % length of xn vector
n = [0:N-1];                    % time index
b = ones(L,1)/L;                % initialize coefficient vector
%b = [0.. 0.. 0.. 0.. 0.. 0..];  %ones(L,1)/L;
yn = filter(b,1,noisysignal);   % FIR filtering
plot(n,noisysignal,'--b',n,yn,'-r')        % plot both input & output
title('Input & output of moving-average filter');
xlabel('Time index, n');
ylabel('Amplitude');
```



## Signal Processing



# How we can design filter for a random signal (Noisy Signal)

## Signal Processing



- **Compute the spectrum (using FFT) through a MATLAB program to see the response.**
- **Watch the frequency component.**
- **Design a filter which should pass this frequency component (in our example a low pass filter).**

13

## Signal Processing



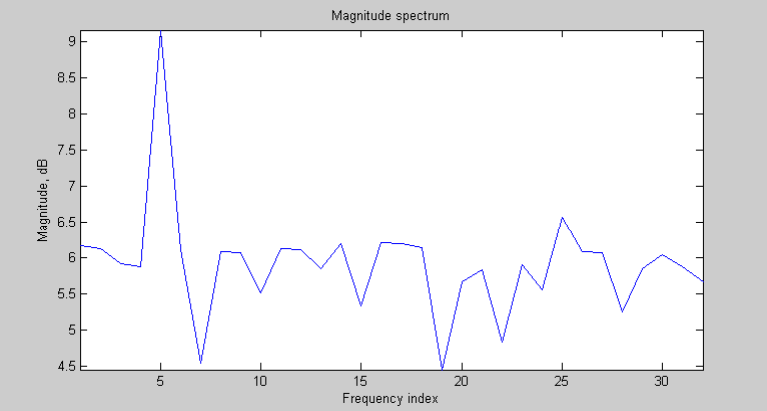
```
% Compute magnitude spectrum of input random signal

load –ascii noisysignal.dat          % load data
N = length(nosiysignal);             % size of data
Xk=fft(noisysignal N);               % perform N-point DFT of signal
magX=20*log10(abs(Xk));              % magnitude spectrum in dB
plot(magX);                          % plot the magnitude spectrum
axis([1 N/2 -inf inf]);              % from DC to Nyquist
title('Magnitude spectrum');
xlabel('Frequency index'),
ylabel('Magnitude, dB');
```

14

## Signal Processing



### Magnitude Spectrum through FFT




15

## Signal Processing



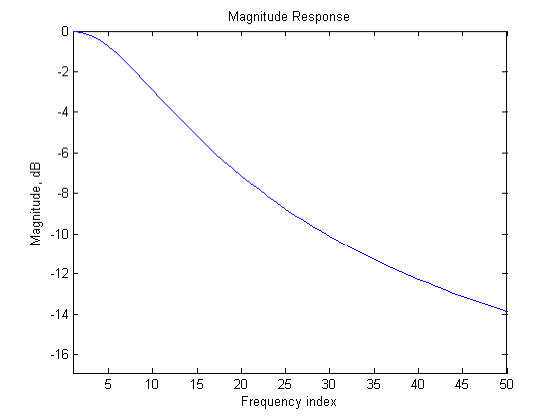
```
% Magnitude response of First-order Filter

alpha = 0.25;
N = 100;
b = [alpha];                    % coefficient vector of numerator
a = [1 -(1-alpha)];             % coefficient vector of denominator
H = freqz(b, a, N);             % frequency response of filter
magX=20*log10(abs(H));          % magnitude response in dB
plot(magX);
axis([1 N/2 -inf inf]);
title('Magnitude Response');
xlabel('Frequency index');
ylabel('Magnitude, dB');
```

16

Signal Processing



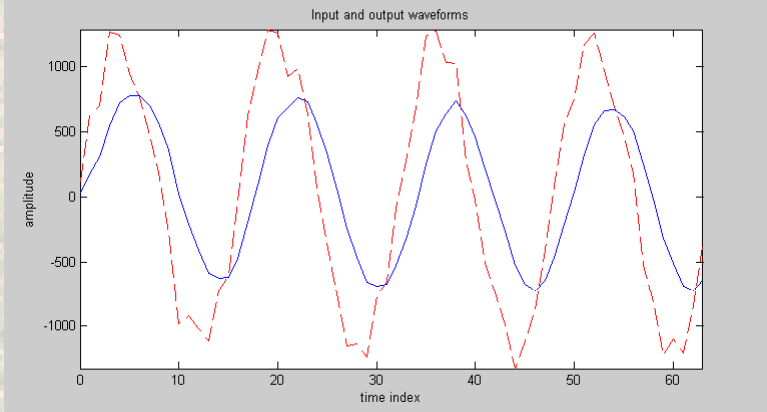
## Magnitude Response of a First Order Filter




17

Signal Processing






18

## Signal Processing



```
Int input;
FILE *xn_in;                                    // file pointer of x(n)
FILE *yn_out;                                   // file pointer of y(n)
xn_in = fopen("noisysignal.dat","r");           // open file for input x(n)
yn_out = fopen("filtersignal.dat","w");         // open file for output y(n)
for (i=0; i<order; i++)
  xnBuf[i] = 0.0;                               // clear signal buffer

//Start of main program

while( (fscanf(xn_in,"%d",&input)) != EOF)
 {                                              // read x(n) from data file
  xn = (float)input;                    // convert to floating-point
```



## Signal Processing



```
while( (fscanf(xn_in,"%d",&input)) != EOF)  // read x(n) from data file
 {
  xn = (float)input;                        // convert to floating-point
  for (i=order-1; i>0; --i)                 // refresh signal buffer
          {
          xnBuf[i] = xnBuf[i-1];            // shift old data x(n-i), i = 1, 2, ...N-1
          }
          xnBuf[0] = xn;                    // insert new data to x(n)
          yn = 0.0;                         // y(n) = 0.
          for (i=0; i<order; ++i)
          {
          yn += bnCoef[i]*xnBuf[i];          // Filtering of x(n) to get y(n)
          }
  fprintf(yn_out,"%d\n",(int)(yn));         // round to integer and write y(n) to file
 }
 fclose(xn_in);
 fclose(yn_out);                            // close all opened files
 printf ("Program Complete \n");
```