DSP Lab Manual -4

Table of Contents

Moving Average filter

The input-output relation of the moving average filter is given by

$$y[n] = \frac{1}{M}\sum_{l=0}^{M-1} x[n-l]$$

Impulse response of the filter is given as

$$h[n] = \begin{cases} \frac{1}{M} & 0 \le n \le M-1 \\ 0 & \text{Otherwise} \end{cases}$$

The transfer function of the moving-average filter is thus,

$$H(z) = \frac{1}{M}\sum_{n=0}^{M-1} z^{-n}$$
$$= \frac{z^M - 1}{M(z^{M-1})(z-1)}$$

Function zplane can be used to plot the pole-zero plot of the function as follows:

About the function zplane -

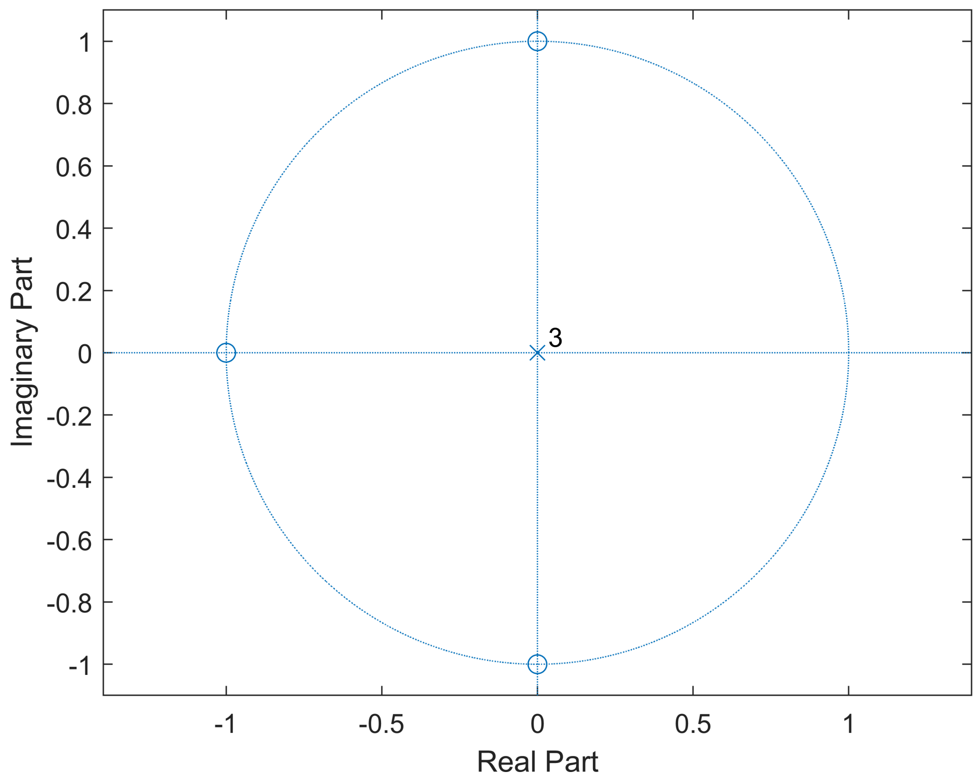
zplane(Z,P) plots the zeros Z and poles P (in column vectors) with the unit circle for reference. Each zero is represented with a 'o' and each pole with a 'x' on the plot. Multiple zeros and poles are indicated by the multiplicity number shown to the upper right of the zero or pole. zplane(Z,P) where Z and/or P is a matrix, plots the zeros or poles in different columns using the colors specified by the axes ColorOrder property.

zplane(B,A) where B and A are row vectors containing transfer function polynomial coefficients plots the poles and zeros of B(z)/A(z). Note that if B and A are both scalars they will be interpreted as Z and P.

```
% Pole zero plot of Moving average filter of length M
clc;
M= 4
```

```
M = 4
```

```
numM = ones(1,M);
denM =[M, zeros(1, M-1)];
zplane(numM, denM)
```



From the plot it can be seen that the transfer function has M zeros on the unit circle at $z = e^{\frac{j2\pi k}{M}}, k = 0, 1, 2, \ldots, M-1$.There is a (M-1)th order pole at the origin Z=0, and a single pole at Z=1. But the ploe at z=1 exactly cancles a zero at the same place, resulting with transfer function with all poles at the origin.

You can use another buit-in function "tf2zp" for fetching the locations of poles and zeros of the transfer function.

```
[z,p,k] = tf2zp(numM, denM)
```

```
z = 3×1 complex
  -1.0000 + 0.0000i
  -0.0000 + 1.0000i
  -0.0000 - 1.0000i
p = 3×1
     0
     0
     0
k = 0.2500
```

Similarly, the function "zp2tf" is available to get the transfer function from location of zeros, poles and constant values.

```
[Num_1, Den_1] = zp2tf(z,p,k)
```

```
Num_1 = 1×4
    0.2500    0.2500    0.2500    0.2500
Den_1 = 1×4
     1     0     0     0
```

Frequency response of the Moving average filter

By letting $z = e^{j\omega}$ with $0 \le \omega \le 2\pi$, we can determine the frequency response of the system/filter.

Hence,

$$
\begin{aligned}
H(e^{j\omega}) &= \frac{1}{M}\sum_{n=0}^{M-1} e^{-j\omega n} \\
&= \frac{1}{M}\left[\frac{1-e^{-\mathrm{jM}\omega}}{1-e^{-j\omega}}\right] \\
&= \frac{1}{M}\frac{\sin\left(\frac{M\omega}{2}\right)}{\sin\left(\frac{\omega}{2}\right)} e^{-j(M-1)\omega/2}
\end{aligned}
$$

From the above, magnitude and phase response of the moving-average filter can be obtained as:

$$
|H(e^{j\omega})| = \left|\frac{1}{M}\frac{\sin\left(M\frac{\omega}{2}\right)}{\sin\left(\frac{\omega}{2}\right)}\right|
$$

and

$$
\theta(\omega) = -\frac{(M-1)\omega}{2} + \pi \sum_{k=1}^{\left\lfloor \frac{M}{2} \right\rfloor} \mu\left(\omega - \frac{2\pi k}{M}\right),
$$

where, $\mu(\omega)$ is the step funciton of ω.

The M-file function freqz(h,w) can be used to determine the values of the frequency response of the system/filter at a set of given frequency points. We can then calculate the real and imaginary parts of the function (using real and imag) and magnitude and phase (using functions abs and angle or argtan2).

About freqz -

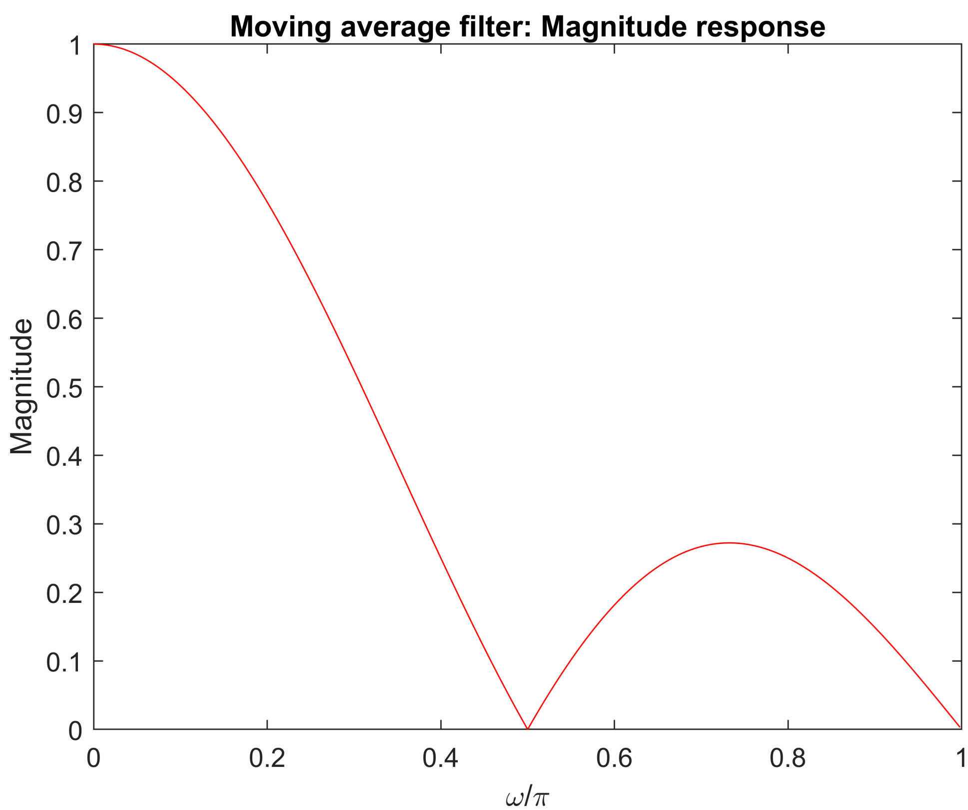
freqz - Frequency response of digital filter

[H,W] = freqz(B,A,N) returns the N-point complex frequency response vector H and the N-point frequency vector W in radians/sample of the filter:

```
                 jw                  -jw              -jmw
        jw   B(e)      b(1) + b(2)e + .... + b(m+1)e
     H(e)  = ---- =    ------------------------------------
                 jw                     -jw              -jnw
              A(e)       a(1) + a(2)e + .... + a(n+1)e
```

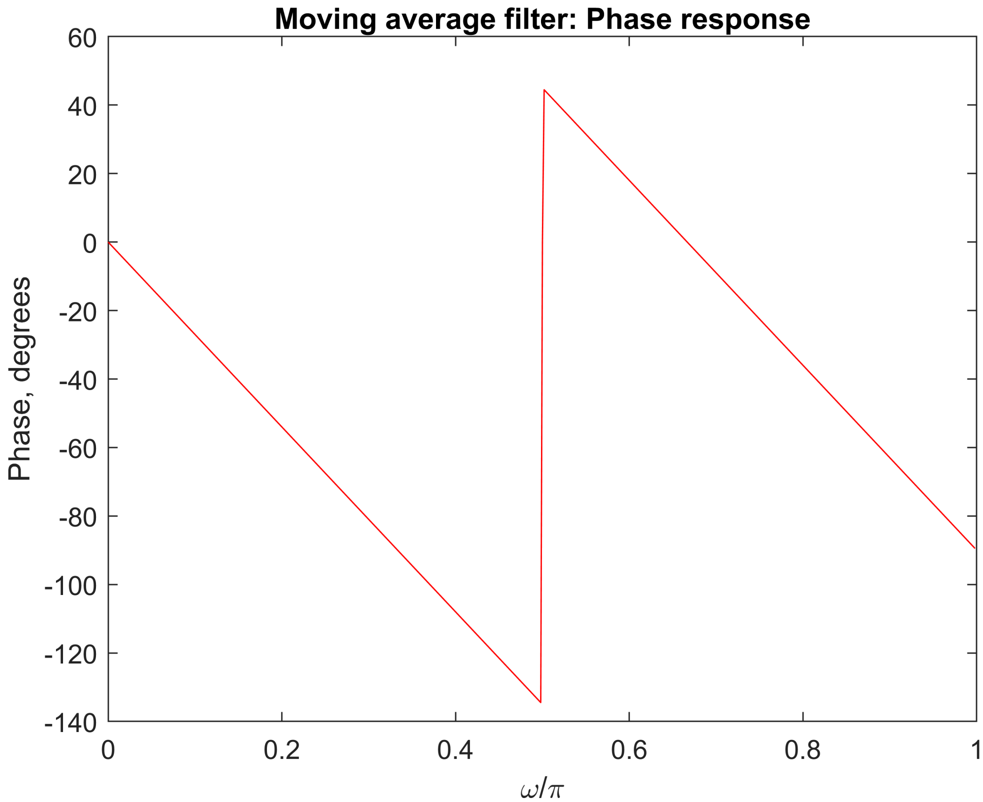
given numerator and denominator coefficients in vectors B and A.

```
% For plotting magnitude and phase plot of moving average filter
[H,w] = freqz(numM, denM, 512);
m = abs(H);
plot(w/pi,m,'r-');
ylabel('Magnitude'); xlabel('\omega/\pi');
title('Moving average filter: Magnitude response')
```




```
figure(2)
%  Compute and plot the phase responses
ph = angle(H)*180/pi;
plot(w/pi,ph,'r-');
ylabel('Phase, degrees');xlabel('\omega/\pi');
```

```
title('Moving average filter: Phase response')
```



Several observations can be done based on the magnitude and phase plots as follows:

- In the range of $0 \le \omega \le \pi$.the magnitude has a maximum value of unity at $\omega = 0$, and magnitude is zero aat $\omega = 2\pi\frac{k}{M}$ with $k = 1, 2, , \ldots, \left\lfloor\frac{M}{2}\right\rfloor$
- The phase function exhibits discontinuities of πat each zero of frequency response, and is linear elsewhere with a slope of -(M-1)/2.
- Both the magnitude and phase functions are periodic in ω with a period of 2π.

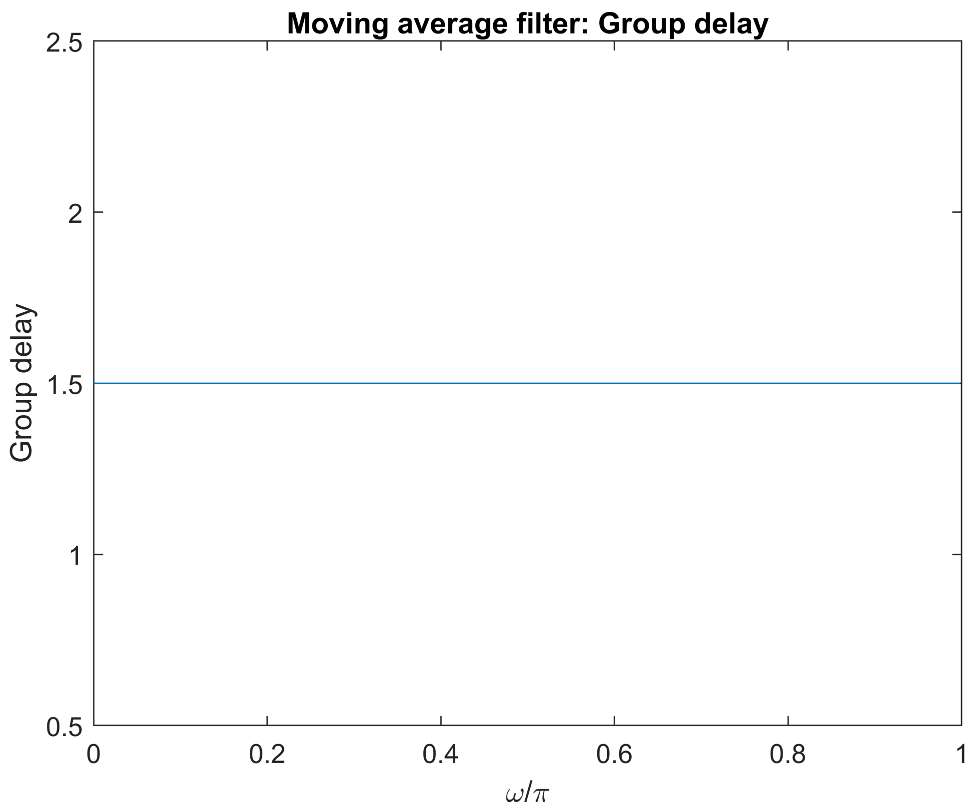
grpdelay Group delay of digital filter

[Gd,W] = grpdelay(B,A,N) returns length N vectors Gd and W containing the group delay, and the frequencies (in radians) at which it is evaluated.

Group delay is defined as -d{angle(w)}/dw. The frequency response is evaluated at N points equally spaced around the upper half of the unit circle. If you don't specify N, it defaults to 512.

[Gd,W] = grpdelay(...,N,'whole') uses N points around the whole unit circle.

```
[Gd,W] = grpdelay(numM,denM,512);
plot(W/pi,Gd);
ylabel('Group delay'); xlabel('\omega/\pi');
title('Moving average filter: Group delay')
```



Using the moving-average filter to remove noise

Suppose a signal $x(t) = 5\cos(2\pi 5t)$, sampled with Fs = 1000 samples/sec is corrupted by a small amount of noise. We can minimize the noise effect by averaging M successive samples of (noise-corrupted) signal as follows:

$$y[n] = \frac{x[n] + x[n-1] + x[n-2] + \ldots + x[n-M-1]}{M}$$

i.e. by making use of moving-average filter.

From MATLAB, we can make use of built-in function filter and pass on the impulse response vector of moving-average filter to it.

filter One-dimensional digital filter.

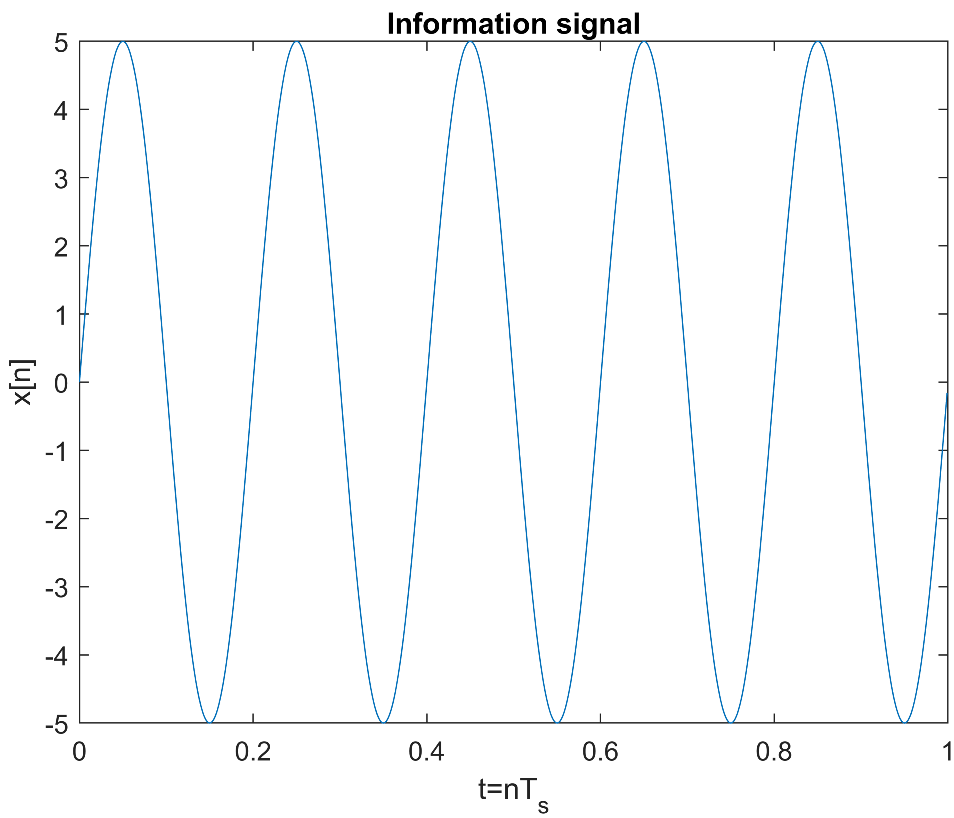
Y = filter(B,A,X) filters the data in vector X with the filter described by vectors A and B to create the filtered data Y.

a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb) - a(2)*y(n-1) - ... - a(na+1)*y(n-na)

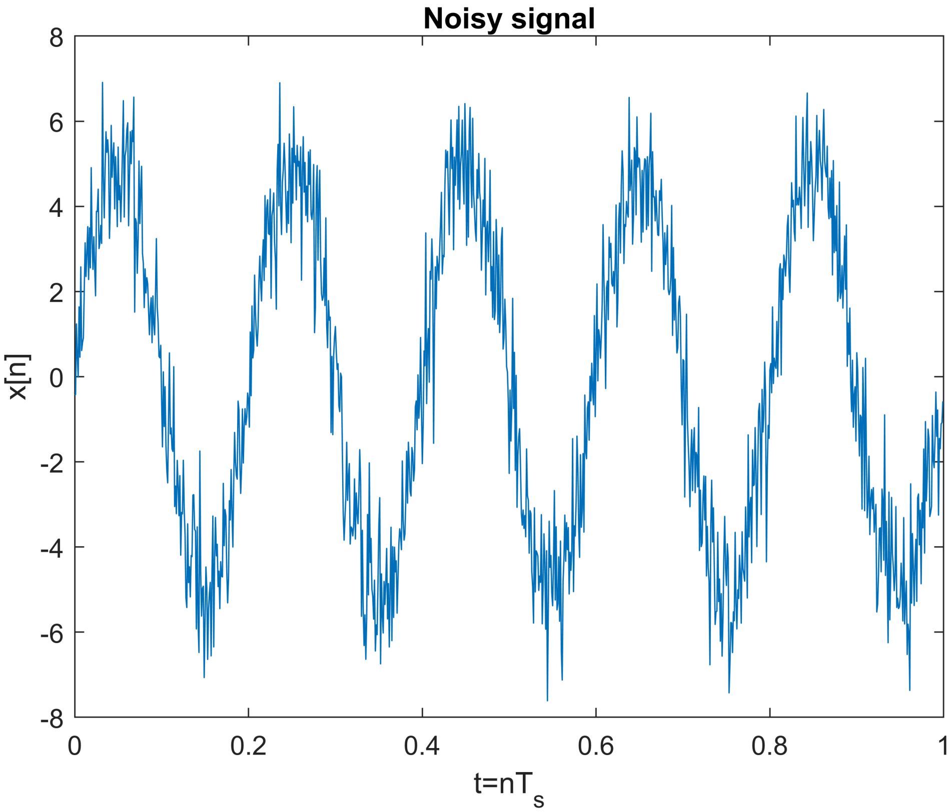
If a(1) is not equal to 1, filter normalizes the filter coefficients by a(1).

In this case, vector b would be b = ones(M,1)/M

```
amplitude_1 = 5; % Amplitude of signal
freq_1 = 5;  % Frequency of signal
Fs = 1000;  % Sampling frequency
time = 0:1/Fs:(1-1/Fs); % 1 second duration
signal_1 = amplitude_1*sin(2*pi*freq_1.*time); % time-domain (information) signal
noise = randn(1,length(time)); % Noise
signal_noisy = signal_1 + noise; % Noise corrupted signal
figure(1)
plot(time,signal_1);
title('Information signal'); xlabel('t=nT_s'); ylabel('x[n]');
```




```
figure(2)
plot(time,signal_noisy);
title('Noisy signal'); xlabel('t=nT_s'); ylabel('x[n]');
```



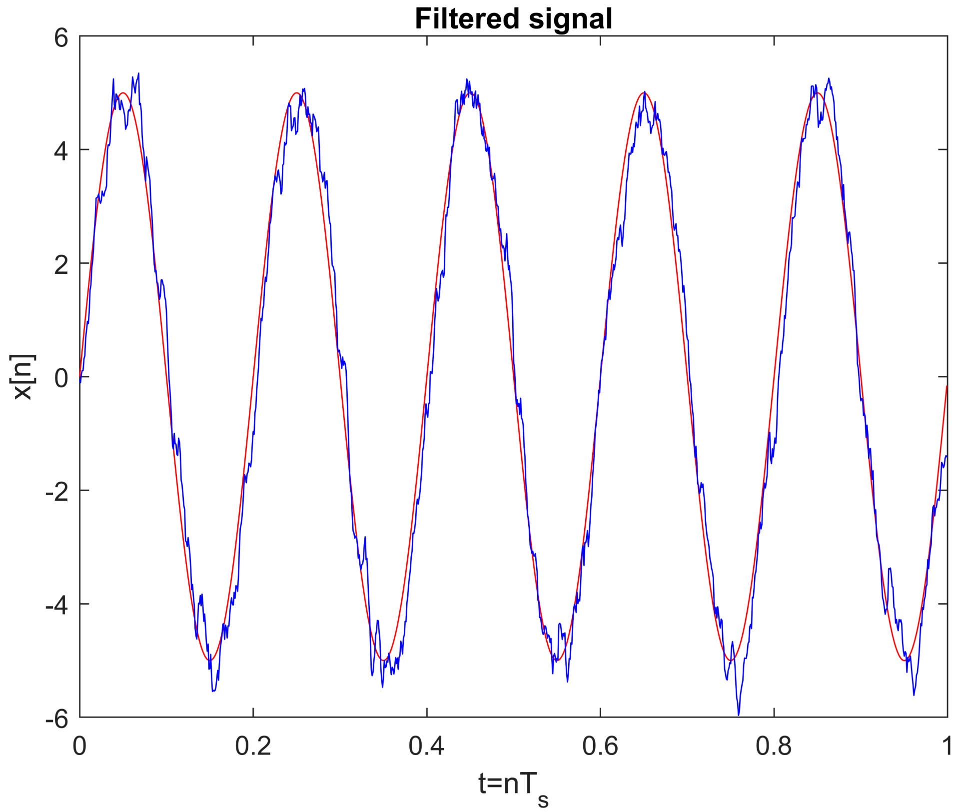

```
% using a moving-average filter
M1 =8
```

```
M1 = 8
```

```
b = ones(M1,1)/M1
```

```
b = 8×1
    0.1250
    0.1250
    0.1250
    0.1250
    0.1250
    0.1250
    0.1250
    0.1250
```

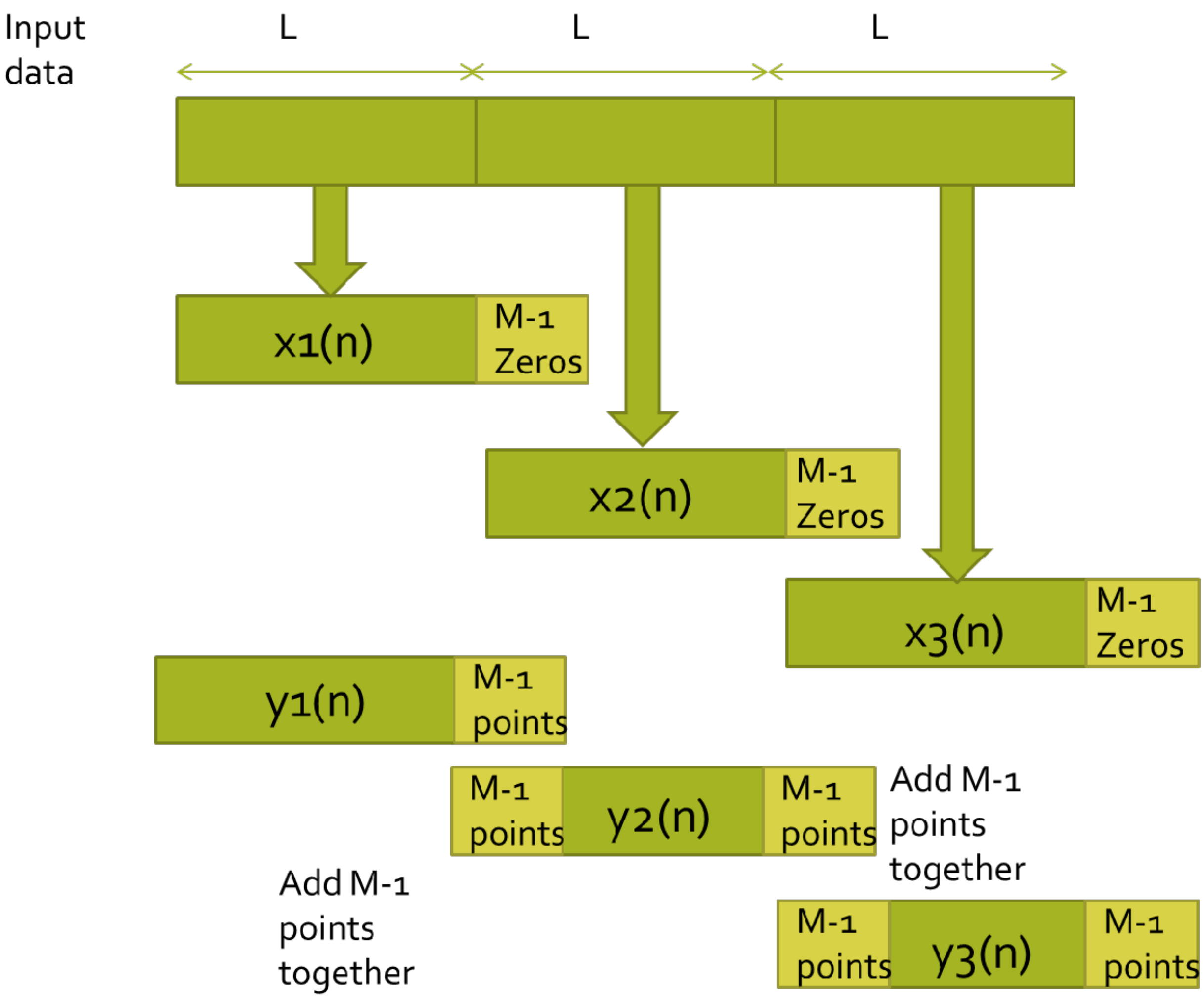
```
y = filter(b,1,signal_noisy);
plot(time,signal_1,'r-',time,y,'b-');
title('Filtered signal'); xlabel('t=nT_s'); ylabel('x[n]');
```



Block convolution using segmentation methods

- When the length of sequence becomes longer, signal segmentation methods can be used to perform “fast-convolution”.
- It is done by sectioning or grouping the long input sequence into blocks of samples.
- Final convolution is obtained by combining the partial convolution results generated by each block.
- Each block is processed via DFT and IDFT (FFT algo) to produce block of output data.
- Two popular methods - Overlap-add method and overlap-save method.

Overlap-add method




```
% Theory:
%
% Overlap Add Method:
% The overlap-add method is an efficient way to evaluate the discrete convolution of a
% very long signal with a finite impulse response (FIR) filter where h[m] = 0 for m
% outside the region [1, M].The concept here is to divide the problem into multiple
% convolutions of h[n] with short segments of x[n], where L is an arbitrary segment
% length. Because of this y[n] can be written as a sum of short convolutions.
%
% Algorithm:
%
% The signal is first partitioned into non-overlapping sequences, then the discrete
% Fourier transforms of the sequences are evaluated by multiplying the FFT xk[n] of
% with the FFT of h[n]. After recovering of yk[n] by inverse FFT, the resulting
% output signal is reconstructed by overlapping and adding the yk[n]. The overlap
% arises from the fact that a linear convolution is always longer than the original
% sequences. In the early days of development of the fast Fourier transform, L was
% often chosen to be a power of 2 for efficiency, but further development has
% revealed efficient transforms for larger prime factorizations of L, reducing
% computational sensitivity to this parameter.
% A pseudo-code of the algorithm is the following:
%
```

```
% Algorithm 1 (OA for linear convolution)
%  Evaluate the best value of N and L
%    H = FFT(h,N)       (zero-padded FFT)
%    i = 1
%    while i <= Nx
%        il = min(i+L-1,Nx)
%        yt = IFFT( FFT(x(i:il),N) * H, N)
%        k  = min(i+N-1,Nx)
%        y(i:k) = y(i:k) + yt    (add the overlapped output blocks)
%        i = i+L
%    end
%
% Note: The following method uses the block convolution algorithm to
% compute the convolution
%

%x = input('Enter the sequence X(n) = ');
%fprintf('This sequence should be a integral multiple of 2*n \n');
%h = input('Enter the sequence H(n) = ');

x =[3, 0, -2, 0, 2, 1, 0, -2, -1,0, 2, 3]
```

```
x = 1×12
     3     0    -2     0     2     1     0    -2    -1     0     2     3
```

```
h = [2 2 1]
```

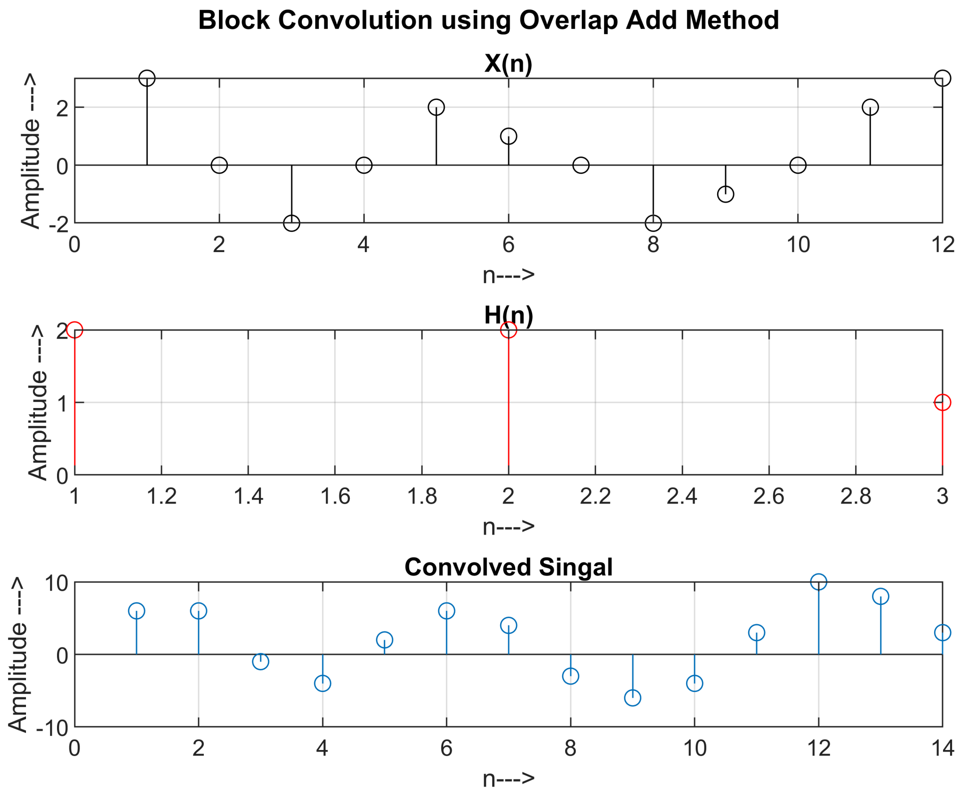
```
h = 1×3
     2     2     1
```

```

% Code to perform Convolution using Overlap Add Method
n1 = length(x);
n2 = length(h);
N = n1+n2-1;
y = zeros(1,N);
h1 = [h zeros(1,n2-1)];
n3 = length(h1);
y = zeros(1,N+n3-n2);
H = fft(h1);
for i = 1:n2:n1
    if i<=(n1+n2-1)
        x1 = [x(i:i+n3-n2) zeros(1,n3-n2)];
    else
        x1 = [x(i:n1) zeros(1,n3-n2)];
    end
    x2 = fft(x1);
    x3 = x2.*H;
    x4 = round(ifft(x3));
    if (i==1)
        y(1:n3) = x4(1:n3);
    else
        y(i:i+n3-1) = y(i:i+n3-1)+x4(1:n3);
    end
```

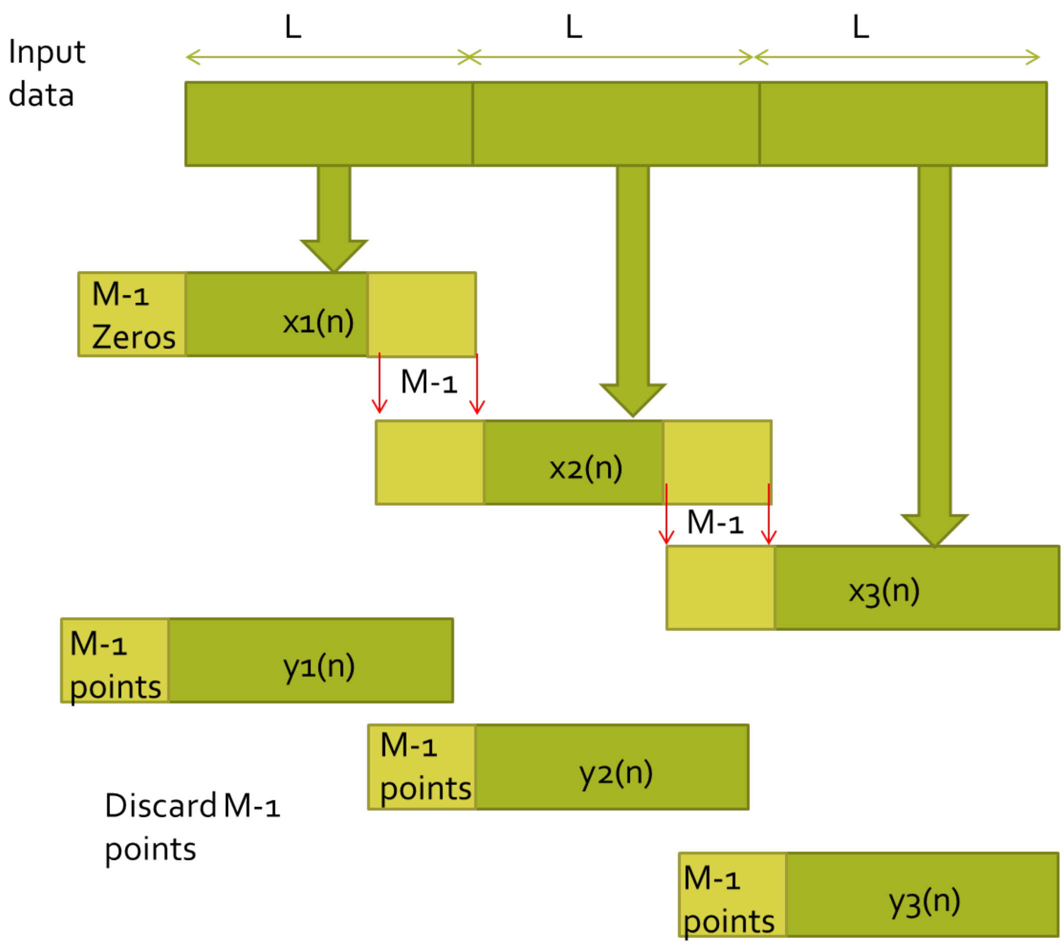
```
end

% Code to plot X(n)
subplot(3,1,1);
stem(x(1:n1),'black');
grid on;
title('X(n)');
xlabel('n--->');
ylabel('Amplitude --->');

%Code to plot H(n)
subplot(3,1,2);
stem(h(1:n2),'red');
grid on;
title('H(n)');
xlabel('n--->');
ylabel('Amplitude --->');

%Code to plot the Convolved Signal
subplot(3,1,3);
disp(y(1:N));
```

```
     6     6    -1    -4     2     6     4    -3    -6    -4     3    10     8     3
```

```
stem(y(1:N));
grid on;
title('Convolved Singal');
xlabel('n--->');
ylabel('Amplitude --->');

% Add title to the Overall Plot
ha = axes ('Position',[0 0 1 1],'Xlim',[0 1],'Ylim',[0 1],'Box','off',...
    'Visible','off','Units','normalized', 'clipping' , 'off');
text (0.5, 1,'\bf Block Convolution using Overlap Add Method ',...
    'HorizontalAlignment','center','VerticalAlignment', 'top')
```



Overlap-Save method




```
% Theory
%
% Overlap Save Method
%
% In this method, the size of the input data blocks is N=L+M-1 and the DFTs and
% the IDFTs are of length L. Each Data Block consists of the last M-1 data points
% of the previous block followed by L new data points to form a data sequence
% of length N=L+M-1.An N point DFT  is computed for each data block. The impulse
% response of the FIR filter is increased in length by appending L-1 zeros and
% an N-point DFT of the sequence is computed once and stored. The multiplication
% of the N-point DFTs for the mth block of data yields
%                              Ym(k)=h(k)Xm(k).
% Since the data record is of length N, the first M-1 points of Ym(n)are corrupted
% by aliasing and must be discarded. The last L points of Ym(n) are exactly the same
% as the result  from linear convolution. To avoid loss of data due to aliasing,
% the last M-1  points of each data record are saved and these points become the first
% M-1 data points of the subsequent record. To begin the processing, the first M-1
% point of the first record is set to zero. The resulting data sequence from the IDFT
% are given where the first M-1 points are discarded due to aliasing and the remaining L
% points constitute the desired result from the linear convolution. This segmentation of
% the input data and the fitting of the output data blocks together
% form the output sequence.
%
% Note: The following method uses the block convolution algorithm
```

```
% to compute the convolution.
%

%x = input('Enter the sequence X(n) = ');
%h = input('Enter the sequence H(n) = ');

x =[3, 0, -2, 0, 2, 1, 0, -2, -1,0, 2, 3]
```

```
x = 1×12
     3     0    -2     0     2     1     0    -2    -1     0     2     3
```

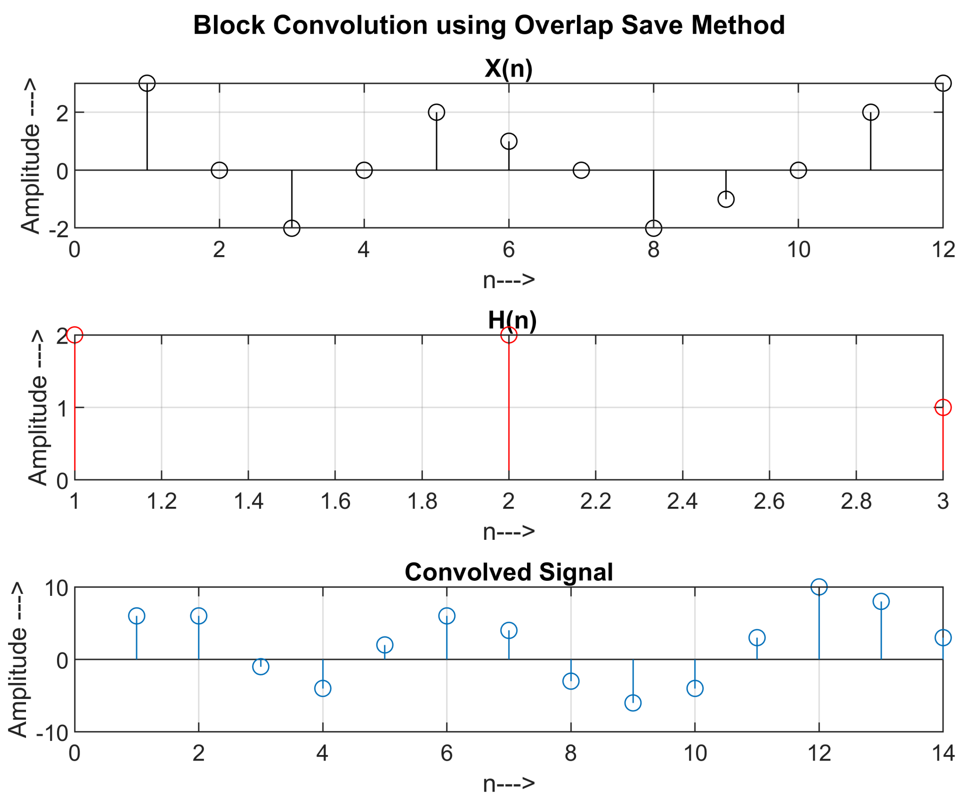
```
h = [2 2 1]
```

```
h = 1×3
     2     2     1
```

```
% Code to perform Convolution using Overlap Save Method
n1 = length(x);
n2 = length(h);
N = n1+n2-1;
h1 = [h zeros(1,N-n1)];
n3 = length(h1);
y = zeros(1,N);
x1 = [zeros(1,n3-n2) x zeros(1,n3)];
H = fft(h1);
for i = 1:n2:N
    y1 = x1(i:i+(2*(n3-n2)));
    y2 = fft(y1);
    y3 = y2.*H;
    y4 = round(ifft(y3));
    y(i:(i+n3-n2)) = y4(n2:n3);
end

% Code to plot X(n)
subplot(3,1,1);
stem(x(1:n1),'black');
grid on;
title('X(n)');
xlabel('n--->');
ylabel('Amplitude --->');

%Code to plot H(n)
subplot(3,1,2);
stem(h(1:n2),'red');
grid on;
title(' H(n)');
xlabel('n--->');
ylabel('Amplitude --->');

% Representation of the Convoled Signal
subplot(3,1,3);
disp(y(1:N));
```

```
     6     6    -1    -4     2     6     4    -3    -6    -4     3    10     8     3
```

```
stem(y(1:N));
grid on;
title('Convolved Signal');
xlabel('n--->');
ylabel('Amplitude --->');

% Add title to the Overall Plot
ha = axes ('Position',[0 0 1 1],'Xlim',[0 1],'Ylim',[0 1],'Box','off',...
    'Visible','off','Units','normalized', 'clipping' , 'off');
text (0.5, 1,'\bf Block Convolution using Overlap Save Method ',...
    'HorizontalAlignment','center','VerticalAlignment', 'top')
```



Speech Processing

```
% MATLAB Program for calculating and displaying speech spectrum

load we.dat
x=we;
fs=8000;                    %sampling rate
% apply the FFT algorithm
N=length(x);                % number of samples
index_t=[0:1:N-1];          % time index
```

```
f=[0:1:N-1]*fs/N;          % map frequency index to frequency in Hz
xf=abs(fft(x))/N;          % compute amplitude spectrum
figure(1)

N;
length(x);
xf_b=abs(fft(x))/N;    % compute amplitude spectrum

subplot(2,1,1);plot(index_t,x);grid
ylabel('x(n)');
xlabel('Time index n');

subplot(2,1,2);plot(f,xf);grid;axis([0 fs 0 400]);
xlabel('Frequency (Hz)');
ylabel('Amplitude spectrum Ak');
```