Practical DSP

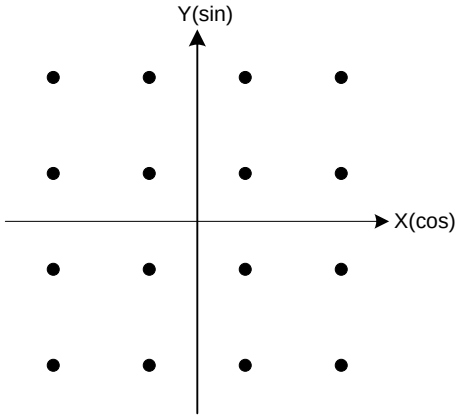
Duy-Ky Nguyen
24 Feb 2000

1. QAM (Quadrature Amplitude Modulation)

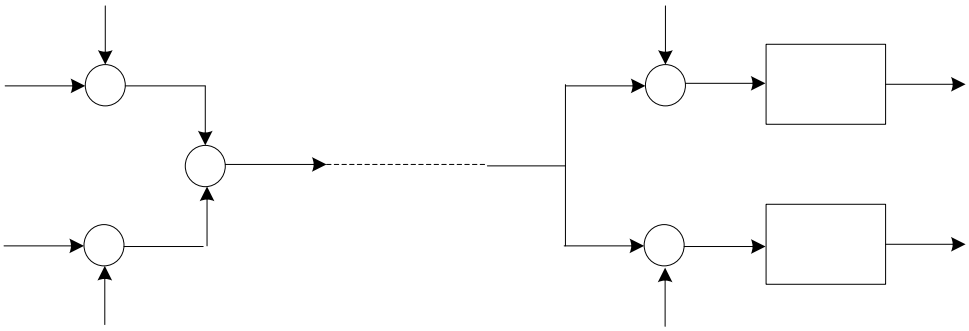
QAM is based on 2 properties below

$$\begin{cases} \int_0^\tau \sin\omega t \cos\omega t dt = 0 \\ \int_0^\tau \sin^2 \omega t = \int_0^\tau \cos^2 \omega t = \dfrac{\tau}{2} \end{cases} \quad (1)$$

A 4-bit information could be sent per QAM symbol as in the figure below



We have QAM modulation and demodulation in the figure below



We have

$$Vm = X\cos\omega t + Y\sin\omega t \quad (2)$$

$$Vi = Vm \times \cos\omega t = X\cos^2\omega t + Y\sin\omega t\cos\omega t \quad (3)$$

$$Vq = Vm \times \sin\omega t = X\sin\omega t\cos\omega t + Y\sin^2\omega t \quad (4)$$

So

$$I = \int_0^\tau Vi\, dt = \int_0^\tau \left(X\cos^2\omega t + Y\sin\omega t\cos\omega t\right) dt = \frac{X\tau}{2} \quad (5)$$

$$Q = \int_0^\tau Vq\, dt = \int_0^\tau \left(X\sin\omega t\cos\omega t + Y\sin^2\omega t\right) dt = \frac{Y\tau}{2} \quad (6)$$

2. Fourrier Analysis

Any periodic "time" function can be uniquely decomposed into sum of sine and cosine functions of frequency, called Fourrier series. So any periodic function can be uniquely represented either in time domain or frequency domain (spectrum) via Fourrier series.

Once the period become infinite, the periodic function becomes aperiodic one, and Fourrier series becomes Fourrier transform.

In the real world, all signals are continuous (analog signals). To be used in digital processing, These analog signal must be time-discretized at some rate called sampling rate and the values are round off to integers to become digitized to become digital signals, vs analog ones.

Once the signal is time-discretized, the Fourrier transform becomes Discrete-Time Fourrier Transform (DTFT) in forms infinite sequence.

If a DTFT is a finite seuquence, it comes Discrete Fourrier Transform (DFT). So DTFT is infinite, while DFT is finite.

2.1. Fourrier Series

A continuous-time function $x(t)$ of period P has a Fourrier series as

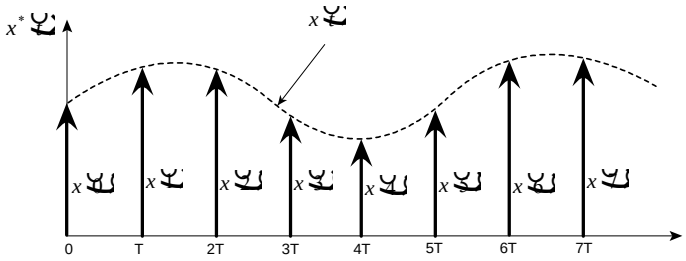
$$\begin{cases} x(t) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi nt/P} \\ c_n = \frac{1}{P}\int_0^P x(t)e^{-j2\pi nt/P}dt = \frac{1}{P}\int_{-P/2}^{P/2} x(t)e^{-j2\pi nt/P}dt \end{cases} \tag{7}$$

2.2. Fourrier Transform

An aperiodic continuous-time function $x(t)$ has a Fourrier transform pair as

$$\begin{cases} X(w) = \int_{-\infty}^{\infty} x(t)e^{-jwt}dt \\ x(t) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(w)e^{jtw}dw \end{cases} \tag{8}$$

2.3. Discrete-Time Fourrier Transform



Discretization of $x(t)$ is

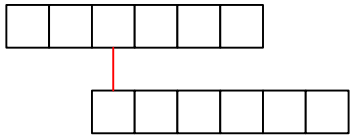
$$x^*(t) = \sum_{k=-\infty}^{\infty} x(kT)\delta(t-kT) \tag{9}$$

where $t = nT$ and $\delta(t)$ is a Dirac delta function.

Taking Fourrier transform of the equation above gives Discrete-Time Fourrier Transform (DTFT)

$$\begin{cases} X(e^{j\omega}) = \sum_{k=-\infty}^{\infty} x(k)e^{-j\omega k} \\ x(k) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega k}\,d\omega \end{cases} \tag{10}$$

2.4. Time Shifting (Sample Shifting)



By DTFT equation

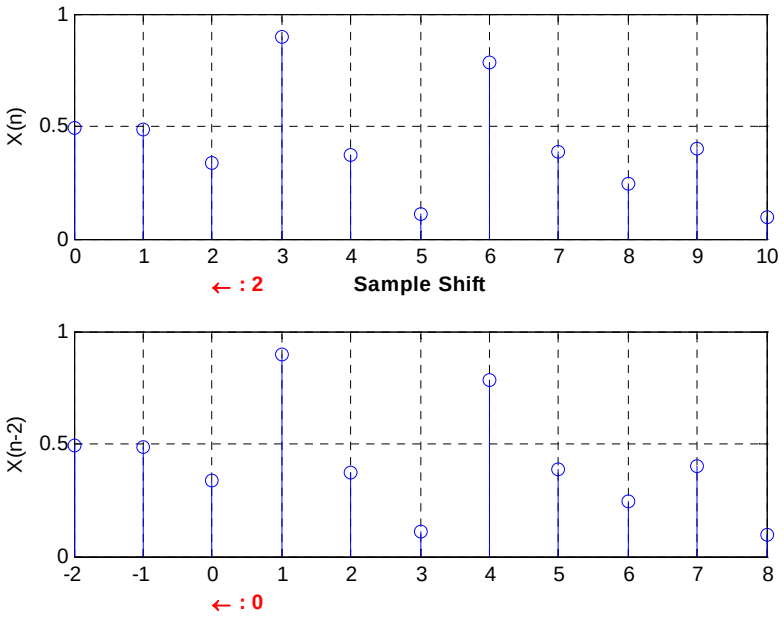
$$Y[n] = X[n-k] = \sum_{n=-\infty}^{+\infty} x(n-k)e^{-j\omega n} \tag{11}$$

Let $m = n-k$, then $n = m+k$, so

$$X[n-k] = \sum_{m=-\infty}^{+\infty} x(m)e^{-j\omega(m+k)} = e^{-j\omega k}\sum_{m=-\infty}^{+\infty} x(m)e^{-j\omega m} = e^{-j\omega k}\sum_{n=-\infty}^{+\infty} x(n)e^{-j\omega n}$$

or

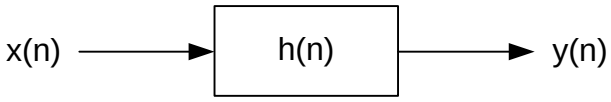
$$X[n-k] = e^{-j\omega k}X[n] \tag{12}$$



x 0 1 2

y 0

2.5. Frequency Domain Representation of LTI Systems



If we have Difference Equation

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{m=0}^{M} b_m x(n-m), \ \forall n, a_0 \neq 0 \tag{13}$$

$$y(n) + \sum_{l=1}^{N} a_l y(n-l) = \sum_{m=1}^{M} b_m x(n-m) \tag{14}$$

Taking Fourrier transform, we have

$$Y[n] + \sum_{l=1}^{N} a_l Y[n-l] = \sum_{m=1}^{M} b_m X[n-m]$$

By sample shifting property, we have

$$Y[n] \times \left(1 + \sum_{l=1}^{N} a_l e^{-j\omega l}\right) = X[n] \times \sum_{m=1}^{M} b_m e^{-j\omega m}$$

Thus

$$H(e^{j\omega}) = \frac{\sum_{m=1}^{M} b_m e^{-j\omega m}}{1 + \sum_{l=1}^{N} a_l e^{-j\omega l}} \tag{15}$$

For impulse response

$$x(n) = \delta(n) \ \Rightarrow \ X[n] = 1 \ \Rightarrow \ Y[n] = H[n] \tag{16}$$

so it's a impulse transfer function

So $H(n)$ also known as Impusle Transfer Function

2.6. Discrete Fourrier Transform

The discrete-time Fourrier transform is applied for an infinite sequence of samples, while a *discrete Fourrier transform* defined in this section should be used for an N-sample sequence.

$$\begin{cases} X(n) = \sum_{k=0}^{N-1} x(k) e^{-j\frac{2\pi}{N}nk}, & n = 0,1,\cdots,N-1 \\ x(k) = \frac{1}{N} \sum_{n=0}^{N-1} X(n) e^{-j\frac{2\pi}{N}nk}, & k = 0,1,\cdots,N-1 \end{cases} \tag{17}$$

over period of 2π

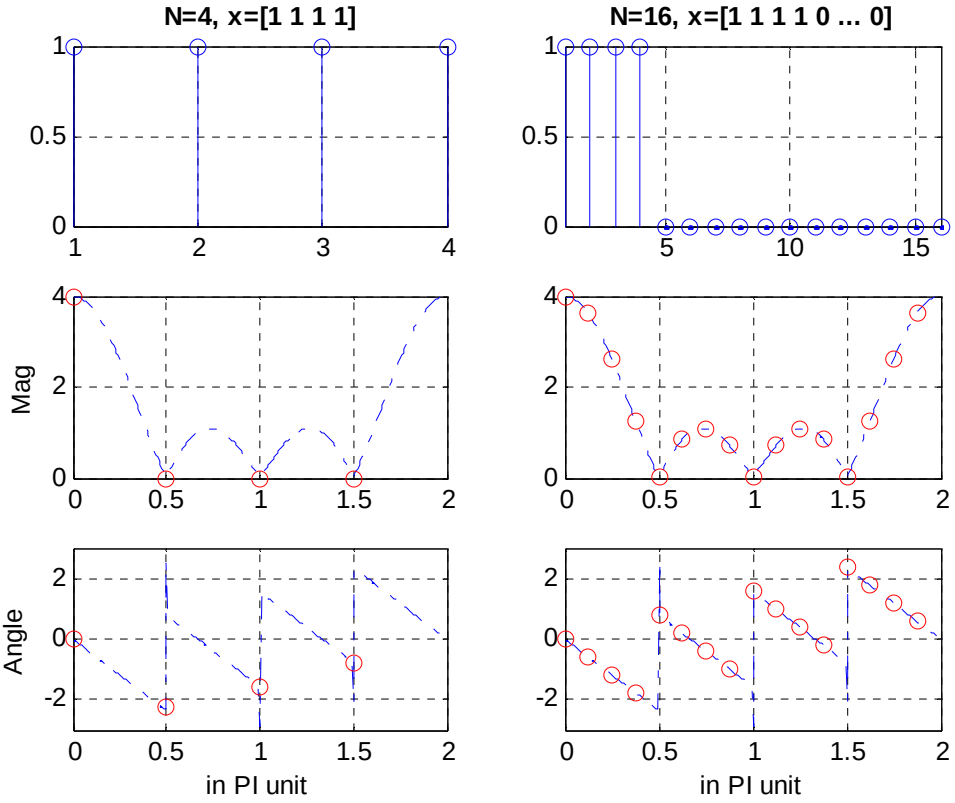
So each term requires N computation, so the transform of N terms involve up to N^2 computations.

FFT (Fast Fourrier Transform) is a fast-computation version of DFT and it requires $N = 2^M$, so zero-padding is a must.

Fourrier transform is used to find spectrum of a signal in frequency domain where a filter could be employed to remove unwanted noise. It's in complex form

$$FFT(x) = R + jI = M\angle\theta \tag{18}$$

where M as frequency response and θ as phase response.



2.7. z-Transform

We have Laplace transform is for analog signal, the Fourrier Series one for analog signal, the DFT for digital signal. For digtal system, we have the *z*-transform define as

$$\begin{cases} X(z) = \sum_{k=-\infty}^{\infty} x[k] z^{-k} \\ x[k] = \oint_C X(z) z^{k-1} dz \end{cases} \tag{19}$$

where z is a complex number

We could see the inverse *z* transform is computed either using advanced calculus or using table, so it's not straight forward like its counter part Fourrier one. To get around this problem, we could compute the *z–inverse* by mapping *z*-transforn into Fourrier transform and using Fouurie inverse to finish the job

2.7.1. Inverse z-Transform

The *z*-Transform gives

$$X(z) = \sum_{n=0}^{N-1} x[n] z^{-n} \tag{20}$$

and the inverse DFT gives

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn} \tag{21}$$

so the previous equation as

$$X(z)=\sum_{n=0}^{N-1}\left(\frac{1}{N}\sum_{k=0}^{N-1}X(k)e^{j\frac{2\pi}{N}kn}\right)z^{-n}=\frac{1}{N}\sum_{k=0}^{N-1}X(k)\sum_{n=0}^{N-1}\left(z^{-1}e^{j\frac{2\pi}{N}k}\right)^{n} \tag{22}$$

By sum of a geometry series

$$\left.\begin{aligned} S &= \sum_{k=0}^{N-1} q^k = 1 + q + q^2 + \cdots + q^{N-1} \\ qS &= q + q^2 + \cdots + q^N \end{aligned}\right\} \Rightarrow (1-q)S = 1 - q^N \Rightarrow S = \sum_{k=0}^{N-1} q^k = \frac{1-q^N}{1-q}$$

so we rewrite Eq(22) as

$$X(z) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) \frac{1 - \left(z^{-1} e^{j\frac{2\pi}{N}k}\right)^N}{1 - z^{-1} e^{j\frac{2\pi}{N}k}} = \frac{1}{N}\sum_{k=0}^{N-1} X(k) \frac{1 - z^{-N} e^{j2\pi k}}{1 - z^{-1} e^{j\frac{2\pi}{N}k}}$$

as $e^{j2\pi k} = 1$, so

$$X(z) = \left(\frac{1 - z^{-N}}{N}\right) \sum_{k=0}^{N-1} \frac{X(k)}{1 - z^{-1} e^{j\frac{2\pi}{N}k}} \tag{23}$$

2.7.2. Difference Equation from z-Transform

Given a transfer function

$$H(z) = \frac{\sum_{m=1}^{M} b_m z^{-m}}{1 + \sum_{l=1}^{N} a_l z^{-l}} \tag{24}$$

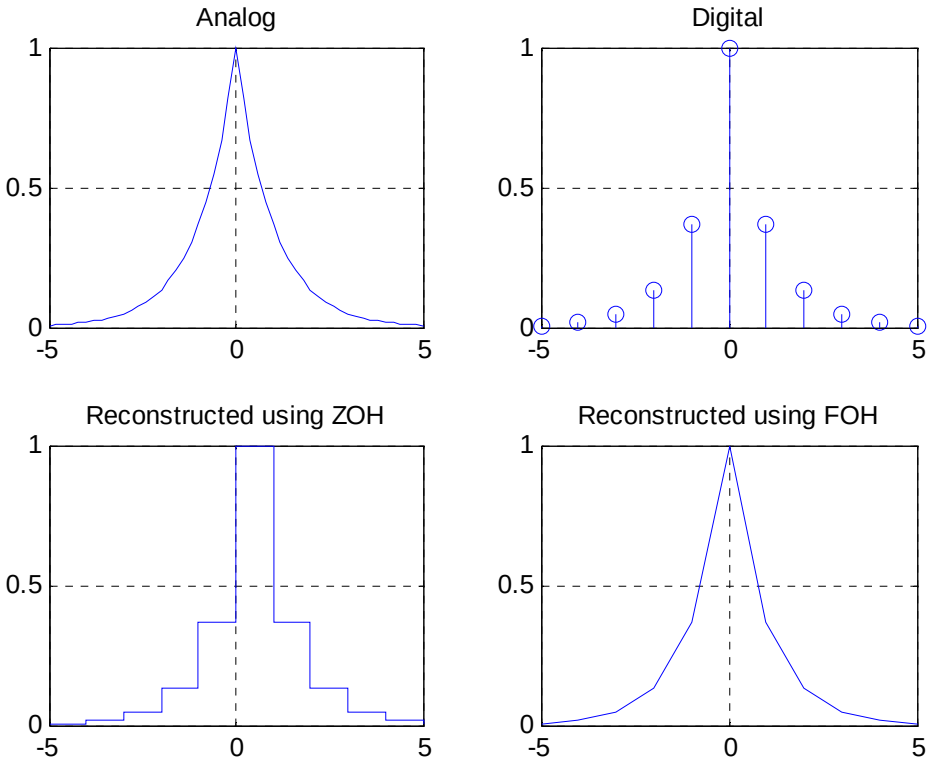
we could convert from $H(z)$ to $h[n]$ and vice versa by using shifting property $X(z)\, z^{-k} \leftrightarrow x[n-k]$

$$\frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} \Rightarrow \left(1 + a_1 z^{-1} + a_2 z^{-2}\right) Y(z) = \left(b_0 + b_1 z^{-1}\right) X(z)$$

so the difference equation

$$y[n] + a_1 y[n-1] + a_2 y[n-2] = b_0 x[n] + b_1 x[n-1] \tag{25}$$

2.8. Reconstruction



A digital signal is reconstructed by some ZOH (Zero-Ordered Hold) where the value is kept the same to the next sample, or FOH (First-Ordered Hold) with a straight line connecting from one sample to the next., then eventually a LPF (Low Pass Filter) is used.

3. Convolution

Linear Convolution is used to check how 2 functions overlap on each other by measuring the overlapping area of their product, *ie* multiplication of 2 functions. This also imply how these 2 function correlate to each other.

$$[f * g](t) \equiv \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau \equiv \int_{-\infty}^{+\infty} f(t-\tau)g(\tau)d\tau \tag{26}$$

For periodic function $g_T()$, it becomes circular convolution

$$[f * g_T](t) \equiv \int_{t_0}^{t_0+T} f(\tau)g_T(t-\tau)d\tau \equiv \int_{t_0}^{t_0+T} f(t-\tau)g_T(\tau)d\tau \tag{27}$$

Linear Discrete Convolution

$$\{f * g_N\}[n] \equiv \sum_{m=-\infty}^{+\infty} f[m]g_N[n-m] \equiv \sum_{m=-\infty}^{+\infty} \left(f[m] \times \sum_{k=-\infty}^{+\infty} g[n-m-kN] \right) \tag{28}$$

Circular Discrete Convolution

$$\{f * g_N\}[n] \equiv \sum_{m=0}^{N-1} f[m]g_N[n-m] \tag{29}$$

It could be proved that transform of convolution of 2 function is equal to product of their transform

$$T\{f * g\} = T[f] \times T[g] \tag{30}$$

So we have Fast Convolution using FFT below

$$\{f * g_N\}[n] \equiv IFFT\{\, FFT\{f\} \times FFT\{g_N\}\,\} \tag{31}$$

where we move back-and-forth between time and frequency (spectrum) domain.

Polynomial multiplication is a convolution of 2 polynomial.

Time-Discretization of analog signal for a discrete-time one, sampling, is the best-known application of convolution

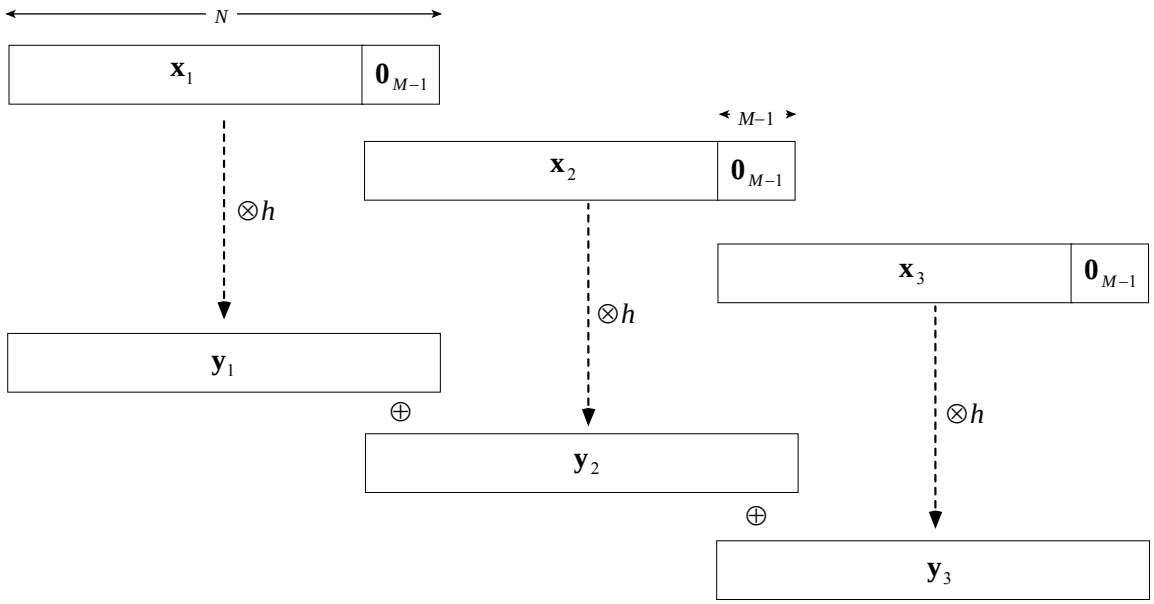
$$x[n] = \sum_{k=-\infty}^{+\infty} x(k)\,\delta[n-k] \tag{32}$$

3.1. Block Convolution

If we want to filter (using convolution) an input sequence received continuously as an infinite sequence, the we experience some practical problems. We will have to compute a large DFT (for fast convolution) which is generally impractical. Furthermore, output samples are not available until all input samples are processed. This introduces an unacceptable large amount of delay. Therefore, we have to segment the infinite input sequence into smaller sections (or blocks), process each sections using DFT, and finally assemble the output sequence from outputs of each section. This procedure is called a *block convolution* operation.

Consider an input sequence *x* of length N_x segmented into *N*-sequence block, and an impulse sequence *h* of length *M*. From Remark 4, we have the first $M-1$ samples in *N*-circular convolution are errornous.

3.1.1. Overlap-Add Method

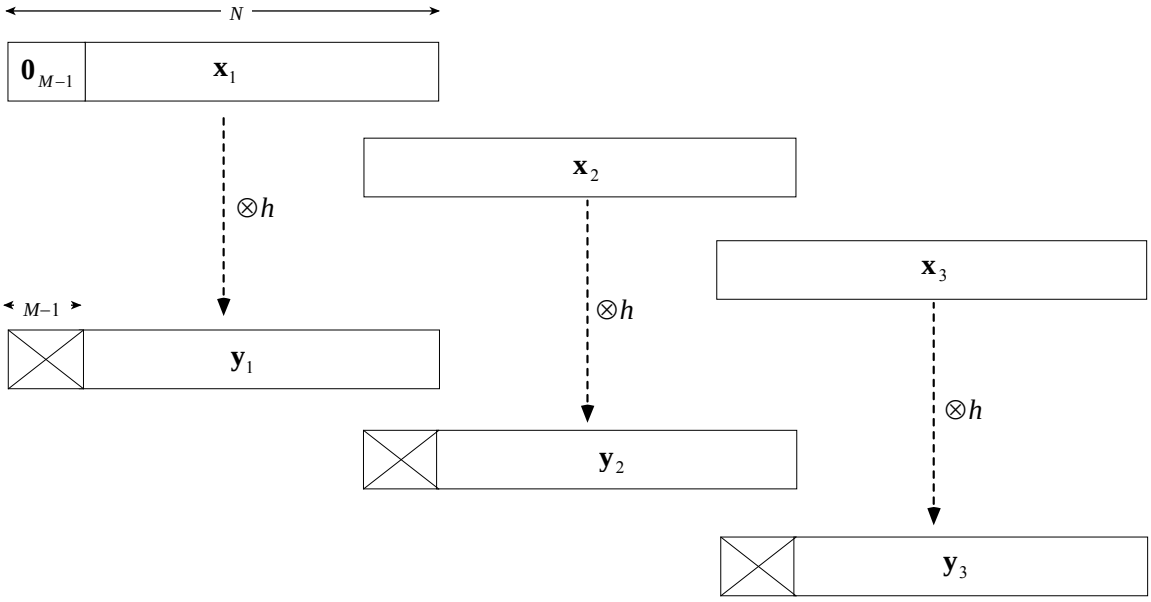



```
clear

x = 1:10;
h = 11:13;

N = 6;
M = length(h);
M1 = M - 1;
```

```
x1 = [ 1  2 3 4 0 0];
x2 = [ 5  6 7 8 0 0];
x3 = [ 9 10 0 0 0 0];

yt1 = circonvt(x1,h,6)
yt2 = circonvt(x2,h,6)
yt3 = circonvt(x3,h,6)

yt2(1:M1) = yt2(1:M1) + yt1(N-M1+1:N);
yt3(1:M1) = yt3(1:M1) + yt2(N-M1+1:N);

yy = [yt1(1:N-M1), yt2(1:N-M1), yt3(1:N-M1)]
y = conv(x,h)
```

```
yt1 =    11    34    70   106    87    52

yt2 =    55   126   214   250   187   104

yt3 =    99   218   237   130     0     0

yy =    11    34    70   106   142   178   214   250   286   322   237   130

y =     11    34    70   106   142   178   214   250   286   322   237   130
```

3.1.2. Overlap-Save Method




```
clear

x = 1:10;
h = 11:13;

N = 6;
M = length(h);
M1 = M - 1;
x1 = [ 0 0 1  2 3 4];
x2 = [ 3 4 5  6 7 8];
x3 = [ 7 8 9 10 0 0];

yt1 = circonvt(x1,h,6)
yt2 = circonvt(x2,h,6)
yt3 = circonvt(x3,h,6)

yy = [yt1(M:N), yt2(M:N), yt3(M:N)]
y = conv(x,h)
```

```
yt1 =    87    52    11    34    70   106
```

```
yt2 =   220   184   142   178   214   250

yt3 =    77   172   286   322   237   130

yy =    11    34    70   106   142   178   214   250   286   322   237   130

y =     11    34    70   106   142   178   214   250   286   322   237   130
```

4. Digital Filters

There're 2 types : IIR (Infinite Impulse Response) and FIR (Finite Impulse Response). The IIR filter is digitized from analog one while FIR is not, *ie* pure digital filter

- IIR is infinite and used for applications where linear characteristics are not of concern.
- FIR filters are Finite IR filters which are required for linear-phase characteristics.
- IIR is better for lower-order tapping, whereas the FIR filter is used for higher-order tapping.
- FIR filters are preferred over IIR because they are more stable, and feedback is not involved.
- IIR filters are recursive and used as an alternate, whereas FIR filters have become too long and cause problems in various applications.

We will present the most popular low-pass filter in both forms of IIR and FIR with MatLab simulations using both discrete equation and transfer function for comparison purpose. We will use input with low and high frequency and expect the high frequency will be filtered out after LPF.

If we have Difference Equation

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{m=0}^{M} b_m x(n-m), \quad \forall n, a_0 \neq 0 \tag{33}$$

From the difference equation above we have extreme case for FIR (Finite Impulse Response)

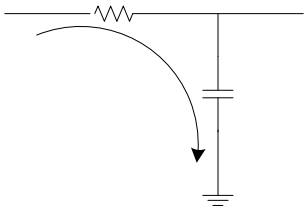
$$y(n) = \sum_{m=0}^{M} b_m x(n-m), \quad \forall n \tag{34}$$

and the other extreme case for IIR (In-finite Impulse Response)

$$\sum_{k=0}^{N} a_k y(n-k) = x(n), \quad \forall n \tag{35}$$

4.1. IIR Low Pass Filter

FIR LPF is discretization of analog RC LPF



By definition of capacitance

$$I = C\frac{dV_o}{dt} \tag{36}$$

so

$$\frac{V_i - V_o}{R} = C\frac{dV_o}{dt} \tag{37}$$

or, in discrete form

$$\frac{V_i[n] - V_o[n]}{R} = C(V_o[n] - V_o[n-1]) \tag{38}$$

where $dV_o = V_o[n] - V_o[n-1]$, and assuming $dt = 1$, *ie* sampling rate of unity.

Let $x = V_i$, $y = V_o$, $\tau = RC$, we have

$$x_n - y_n = \tau(y_n - y_{n-1}) \tag{39}$$

then the discrete equation

$$y_n = \frac{\tau}{1+\tau} y_{n-1} + \frac{1}{1+\tau} x_n$$

as $y_{n-1} = z^{-1} y_n$, we have

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

or

$$y_n = \frac{1}{1+\tau-\tau z^{-1}} x_n$$

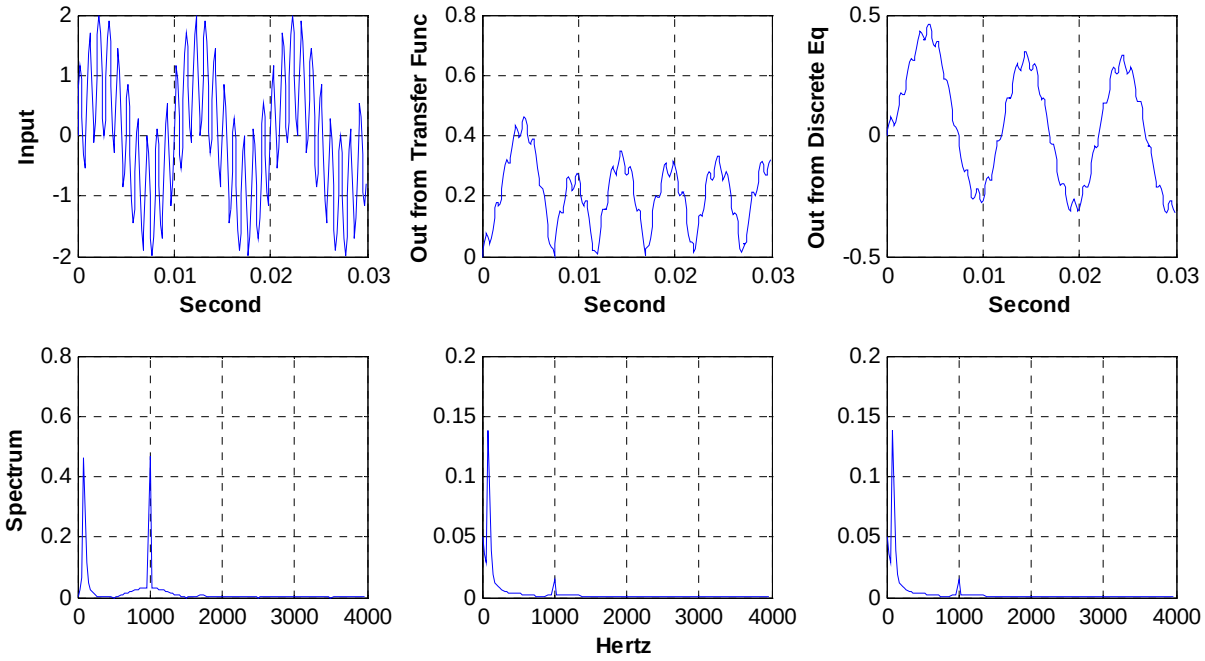
thus the transfer function

$$H_z = \frac{1}{1+\tau-\tau z^{-1}}$$

```
clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tau = input('TAU value <40> : ');

if isempty(tau)
      tau = 40;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Fs = 8e3;     % 8 KHz
Ts = 1/Fs;

N = 240;

tt = Ts*[0:N-1];

Fx1 = 100;
Fx2 = 1e3;

xx = sin(2*pi*Fx1*tt) + sin(2*pi*Fx2*tt);

% Script for a digital implementation of RC low pass filter
close all

a = 1/(1+tau);      % x
b = a*tau;          % yT

len = length(xx);

dlyBuf = zeros(1,len);

for ii = 1:len
      yOut = a*xx(ii) + b*dlyBuf(1);
      dlyBuf(2:end) = dlyBuf(1:end-1);
      dlyBuf(1) = yOut;

      yT(ii) = yOut;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

num = [1];
den = [(1+tau) -tau];

yF = filter(num,den,xx);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NN = 2^round(log2(N));
```

```
NN2 = NN/2;
ff = (Fs/NN)*[0:NN2-1];

xx_sp = (1/NN)*fft(xx,NN);
xx_mag = abs(xx_sp(1:NN/2));

yT_sp = (1/NN)*fft(yT,NN);
yT_mag = abs(yT_sp(1:NN/2));

yF_sp = (1/NN)*fft(yF,NN);
yF_mag = abs(yF_sp(1:NN/2));

len_mag = length(yT_mag);

yT_max = max(yT_mag)/max(yT_mag(20:len_mag));
yF_max = max(yF_mag)/max(yF_mag(20:len_mag));

fprintf('Ratio max of 1st and 2nd peak T:%f F:%f\n', yT_max, yF_max);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clf
figure(gcf)
subplot(2,3,1), plot(tt,xx), grid
      ylabel('Input', ...
             'FontWeight','bold')
      xlabel('Second', ...
             'FontWeight','bold')

subplot(2,3,2), plot(tt,abs(yF)), grid
      ylabel('Out from Transfer Func', ...
             'FontWeight','bold')
      xlabel('Second', ...
             'FontWeight','bold')

subplot(2,3,3), plot(tt,yT), grid
      ylabel('Out from Discrete Eq', ...
             'FontWeight','bold')
      xlabel('Second', ...
             'FontWeight','bold')

subplot(2,3,4), plot(ff,xx_mag), grid
      ylabel('Spectrum', ...
             'FontWeight','bold')
subplot(2,3,5), plot(ff,yT_mag), grid
      xlabel('Hertz', ...
             'FontWeight','bold')
subplot(2,3,6), plot(ff,yF_mag), grid
```

Simulation results with tau = 40


Input
Second
Out from Transfer Func
Second
Out from Discrete Eq
Second
Spectrum
Hertz

Remark 1

- Output from LPF has no high frequency as expected
- Simulation using Discrete Equation looks better than using Transfer Function
- Spectrum of Input has 2 same high peak at 100 Hz and 1000 Hz as expected
- Spectrum of Output has peak at 1000 Hz much lower at 100 Hz, as expected after LPF, the ratio is about 9 times
- There's some transient time before setlling in steady-state

4.2. Linear Phase FIR Filters

There're no feedback from *y* in FIR filters

$$H(z)=\sum_{k=0}^{M-1} b_k z^{-k} \tag{40}$$

(40)
or

$$y(n)=\sum_{k=0}^{M-1} b_k x(n-k) \tag{41}$$

When phase response is a linear function of frequency

$$\angle H\left(e^{j\omega}\right)=\beta-\alpha\omega,\quad \beta=0,\pm\pi/2,\quad -\pi<\omega<\pi$$

so, $H(z)$ must be in the form

$$H\left(e^{j\omega}\right)=e^{j\alpha\omega}H_{\#} \tag{42}$$

Eq(40) gives

$$H\left(e^{j\omega}\right)=\sum_{k=0}^{M-1} b_k e^{-i\omega k}=b_0+b_1e^{-j\omega}+b_2e^{-j2\omega}+b_3e^{-j3\omega}+\cdots \tag{43}$$

To satify Eq(42), Eq.(43) must be factorized with term $e^{j\alpha\omega}$ and $H_{\#}$ must not be a complex number, either real or imaginary. In so doing, we must use property below

$$\begin{cases}\left(e^{j\omega k}+e^{-j\omega k}\right)=2\cos\omega\\ \left(e^{j\omega k}-e^{-j\omega k}\right)=2j\sin\omega\end{cases} \tag{44}$$

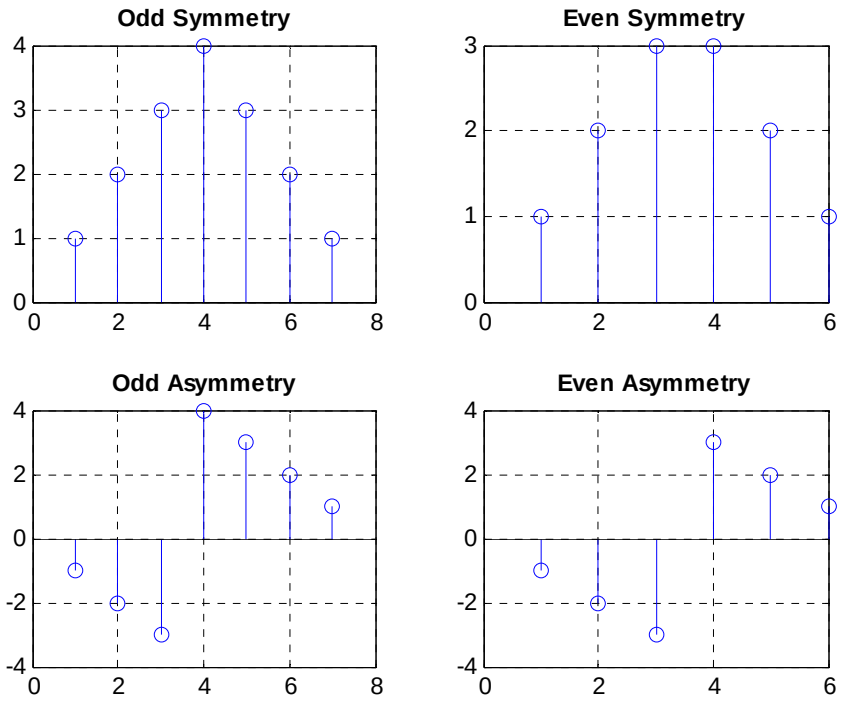
ie z must have a pair of opposite signed integer.

Eq(40) could be written as

$$H(z)=b_0+b_1z^{-1}+b_2z^{-2}+\cdots\pm b_{M-3}z^{-(M-3)}\pm b_{M-2}z^{-(M-2)}\pm b_{M-1}z^{-(M-1)} \tag{45}$$

Let

$$\alpha=\frac{M-1}{2},\quad \alpha^*=floor(\alpha) \tag{46}$$



If $H(z)$ is symmetry or asymmetry (negative symmetry), then

$$H(z)=b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_{\alpha^*}z^{-\alpha^*}\pm\cdots\pm b_2z^{-(M-3)}\pm b_1z^{-(M-2)}\pm b_0z^{-(M-1)}$$

$$H(z)=z^{-\alpha}\left[\underbrace{b_0\left(z^{\frac{M-1}{2}}\pm z^{-\frac{M-1}{2}}\right)+b_1\left(z^{\frac{M-3}{2}}\pm z^{-\frac{M-3}{2}}\right)+b_2\left(z^{\frac{M-5}{2}}\pm z^{-\frac{M-5}{2}}\right)+\cdots+b_{\alpha^*}z^{-\alpha^*}}_{\mathrm{Re}(+)\ \ or\ \ \mathrm{Im}(-)}\right]$$

$$H(z)=z^{-\alpha}\left[b_{\alpha^*}+\underbrace{\sum_{k=0}^{\alpha^*-1}b_k\left(z^{(\alpha^*-k)}\pm z^{-(\alpha^*-k)}\right)}_{\mathrm{Re}(+)\ \ or\ \ \mathrm{Im}(-)}\right] \tag{47a}$$

or

$$H(z)=z^{-\alpha}\left[b_{\alpha^*}+\underbrace{\sum_{k=0}^{\alpha^*-1}b_{\alpha^*-k}\left(z^{k}\pm z^{-k}\right)}_{\mathrm{Re}(+)\ \ or\ \ \mathrm{Im}(-)}\right] \tag{47b}$$

so

$$\angle H\left(e^{j\omega}\right)=\begin{cases}-\alpha^*\omega\\ \dfrac{\pi}{2}-\alpha^*\omega\end{cases} \tag{48}$$

Advantages of Linear Phase

- this technique is widely used in frequency sampling design method
- design problem contains only real arithmetic but not complex arithmetic
- no delay distortion and only fixed amount of delay

- reduce number of operations by half, *ie* only *M*/2 operations for *M*-size filter, in case to save resource, if so required, but not necessary;

4.2.1.1. Type 1 : Symmetry Odd Length *M*

Eq(47b) becomes

$$H\left(e^{j\omega}\right)=e^{-j\omega\frac{M-1}{2}}\left[b_{\frac{M-1}{2}}+\underbrace{\sum_{k=0}^{\frac{M-1}{2}-1}b_{\frac{M-1}{2}-k}\left(e^{j\omega k}+e^{-j\omega k}\right)}_{\mathrm{Re}(+)}\right]=e^{-j\omega\frac{M-1}{2}}\left(b_{\frac{M-1}{2}}+2\sum_{k=0}^{\frac{M-1}{2}-1}b_{\frac{M-1}{2}-k}\cos k\omega\right) \tag{49}$$

4.2.1.2. Type 2 : Symmetry Even Length *M*

Eq(47b) becomes

$$H\left(e^{j\omega}\right)=e^{-j\omega\frac{M-1}{2}}\left[\underbrace{\sum_{k=0}^{\frac{M}{2}-1}b_{\frac{M-2}{2}-k}\left(e^{j\omega k}+e^{-j\omega k}\right)}_{\mathrm{Re}(+)}\right]=2e^{-j\omega\frac{M-1}{2}}\left(\sum_{k=0}^{\frac{M}{2}-1}b_{\frac{M-2}{2}-k}\cos k\omega\right) \tag{50}$$

4.2.1.3. Type 3 : Asymmetry Odd Length *M*

Eq(47b) becomes

$$H\left(e^{j\omega}\right)=e^{-j\omega\frac{M-1}{2}}\left[b_{\frac{M-1}{2}}+\underbrace{\sum_{k=0}^{\frac{M-1}{2}-1}b_{\frac{M-1}{2}-k}\left(e^{j\omega k}-e^{-j\omega k}\right)}_{\mathrm{Im}(-)}\right]=j2e^{-j\omega\frac{M-1}{2}}\left(\sum_{k=0}^{\frac{M-1}{2}-1}b_{\frac{M-1}{2}-k}\sin k\omega\right),\quad b_{\frac{M-1}{2}}=0 \tag{51}$$

4.2.1.4. Type 4 : Asymmetry Even Length *M*

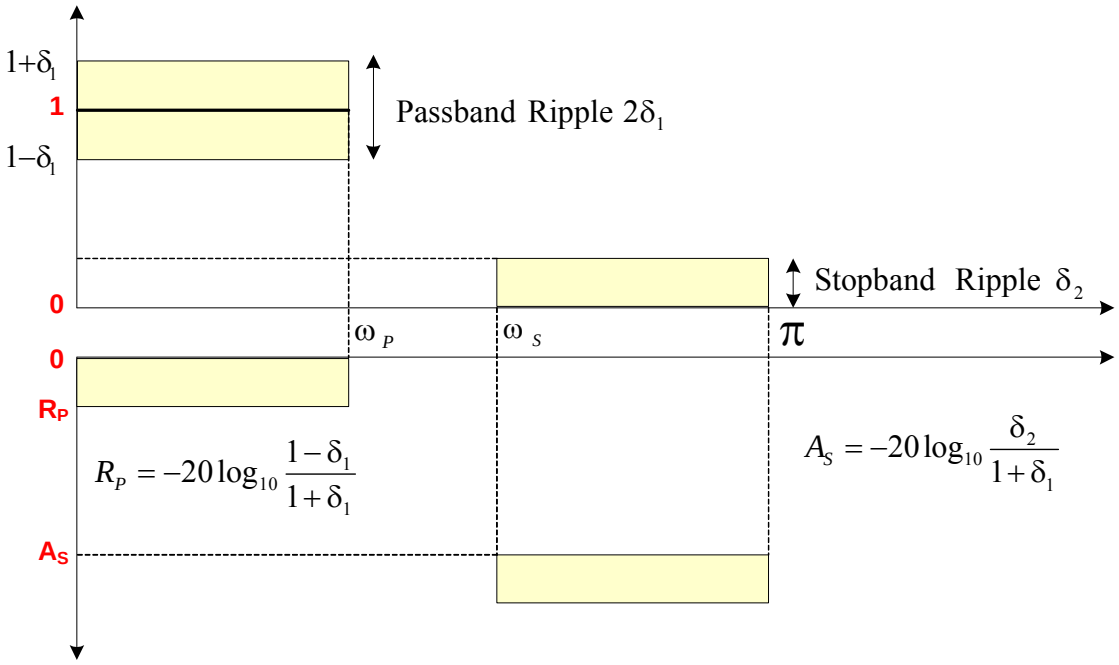
Eq(47b) becomes

$$H\left(e^{j\omega}\right)=e^{-j\omega\frac{M-1}{2}}\left[\underbrace{\sum_{k=0}^{\frac{M}{2}-1}b_{\frac{M-2}{2}-k}\left(e^{j\omega k}+e^{-j\omega k}\right)}_{\mathrm{Im}(-)}\right]=j2e^{-j\omega\frac{M-1}{2}}\left(\sum_{k=0}^{\frac{M}{2}-1}b_{\frac{M-2}{2}-k}\sin k\omega\right),\quad b_{\frac{M-2}{2}}=0 \tag{52}$$

4.3. FIR Filter Design Methods

FIR filters have several design and implementation advantages

- Phase response can be exactly linear
- Easy to design since no stability problem
- Efficient implementation since DFT/FFT can be used



4.3.1. Window method

Given a desired frequency response, a filter could be obtained by using inverse Fourrier Transform (section 2.2)

$$h_d(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(\omega)e^{j\omega n}d\omega, \quad -\infty < n < \infty \tag{53}$$

It's an infinite sequence, so cannot be used, so it has to be truncated using some finite window functions

$$h(n) = h_d(n) \times w(n)$$

where

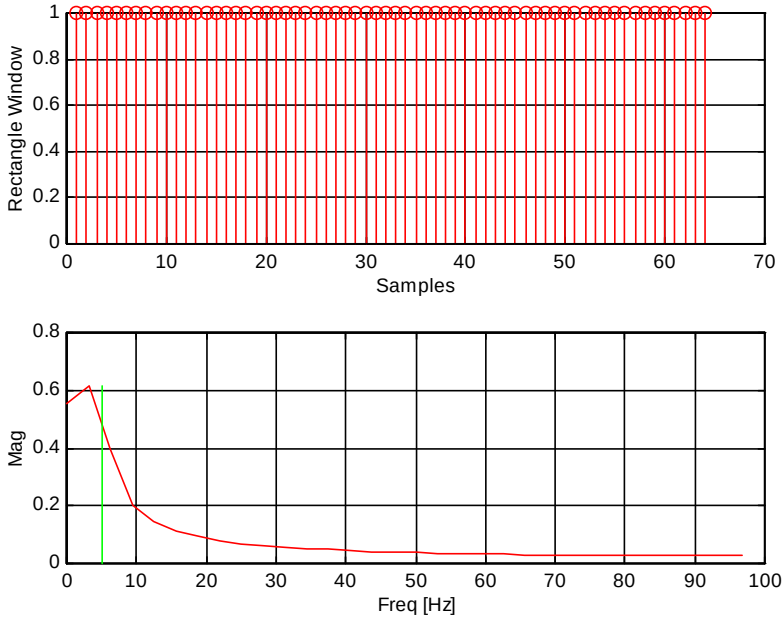
$$w(n) = \begin{cases} f(n), & 0 \le n < M-1 \\ 0, & otherwise \end{cases} \tag{54}$$

Note the difference between these 2 method where the former uses Discrete Fourrier transform while the Continuous Fourrier one used in the latter.

4.3.1.1. Rectangle Window

This is the simplest window function but provides the worst performance from the viewpoint of stop band attenuation

$$w_k \quad \begin{matrix} 0 \le k \le N-1 \\ \text{otherwise} \end{matrix} \tag{55}$$



Remark 1: Gibb phenomenon

The large stopband ripple due to the sudden transition from 0 to 1 (or 1 to 0) in the rectangle window, this is known as *Gibbs phenomenon*. The following windows solves this problem.

```
% Spectrum Analyzer using FFT

clear

J = sqrt(-1);

Ts = 0.005;
Fs = 1/Ts;

N = 64;
nn = [0:N-1] + eps;
nn = nn';

N2 = N/2;

ff = (Fs/N)*[0:N2-1];

Fp = 5;
Wp = 2*pi/Fs*Fp;

Hd = sin(Wp*nn)./(pi*nn);

Win = boxcar(N);

H = Hd .* Win;

yy = fft(H,N);

Mag = abs(yy(1:N/2));

figure(gcf)
subplot(211), stem(Win),xlabel('Samples'),ylabel('Rectangle Window'),grid
subplot(212), plot(ff,Mag,'-',[Fp,Fp],[0,max(Mag)],'-'),xlabel('Freq [Hz]'),ylabel('Mag'),grid
```

4.3.1.2. Kaiser Window

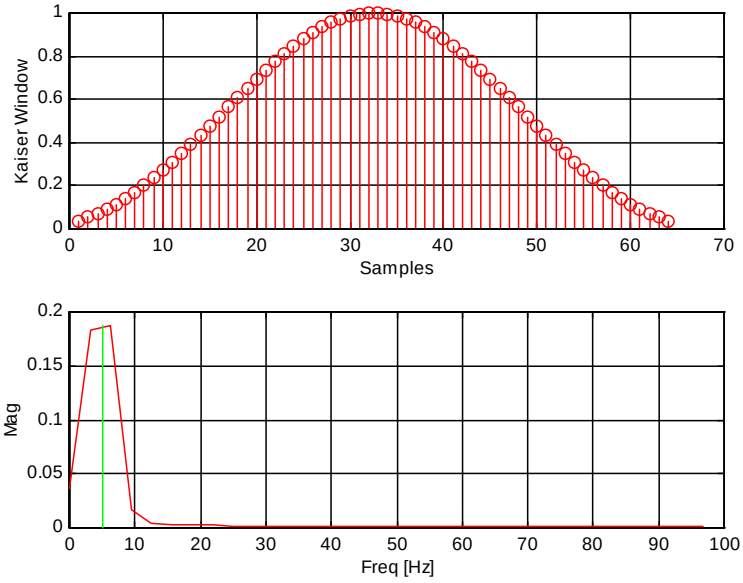
This is one of the most useful and optimum windows. It is optimum in the sense of providing a large main lobe width for a given stopband attenuation, which implies the sharpest transition width.

$$w_k = \begin{cases} \dfrac{I_0\left[\beta\sqrt{1-\left(\frac{2k}{N-1}\right)^2}\right]}{I_0[\beta]}, & 0 \le k \le N-1 \\ \text{otherwise} \end{cases} \tag{56}$$

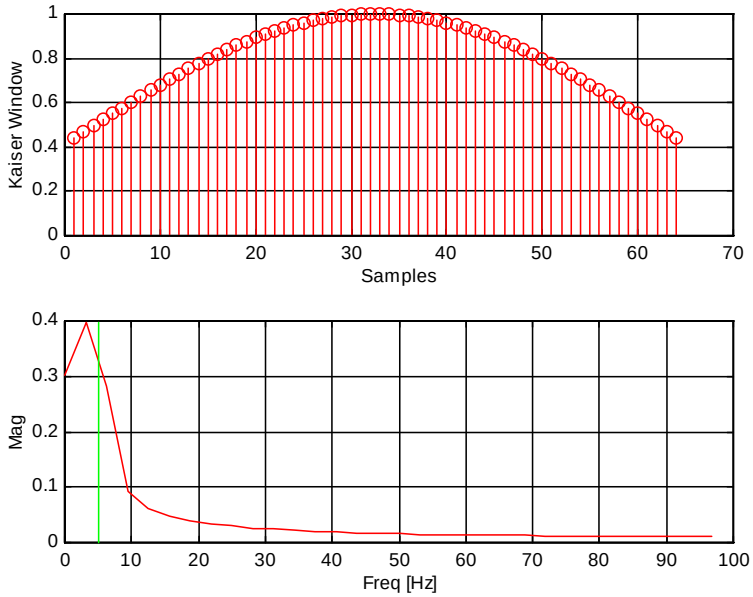
where I_0 is the *modified zero-order Bessel function*, and β is a parameter that depends on N and can be chosen to yield various transition widths and near-optimum stopband attennuation.

```
bta = inp('Beta = <5> = ', 5);
Win = kaiser(N, bta);
```

Choose $\beta = 5$, we have



Choose $\beta = 2$, we have


Kaiser Window
Samples
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50
60
70

Mag
Freq [Hz]
0
0.1
0.2
0.3
0.4
0
10
20
30
40
50
60
70
80
90
100

4.3.1.3. Gaussian Window (NDK)

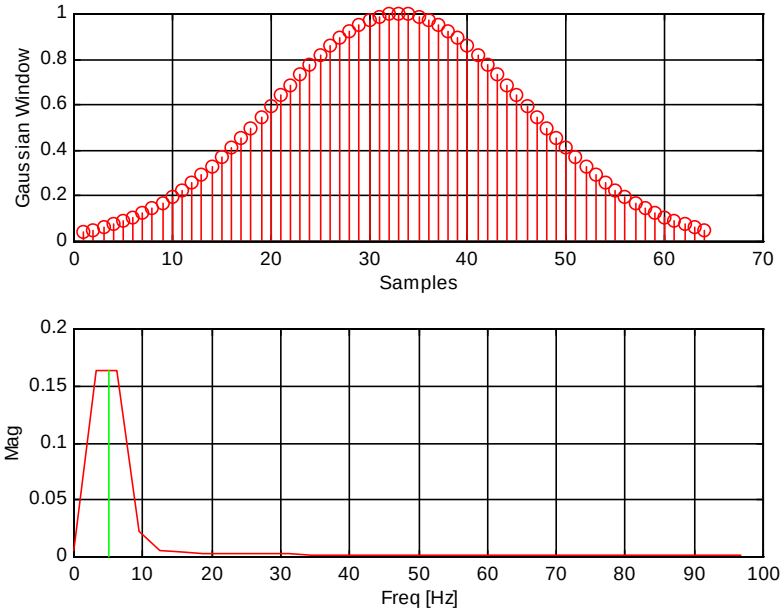
A Gaussian window is proposed to replace Kaiser which uses complicated Bessel function

$$w(k) = \begin{cases} \exp\left(-\frac{1}{2}\left(\frac{k - N/2}{\gamma}\right)^2\right) & 0 \le k \le N \\ \text{otherwise} \end{cases} \quad (57)$$

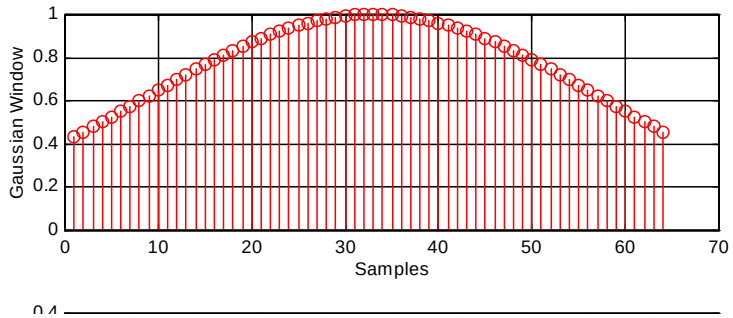
where γ is chosen equivalent to β in Kaiser window

```
gama = inp('Gamma = <5> = ', 5);
Win = gauss(N, gama);
```

Choose $\gamma = 5$, we have



Choose $\gamma = 2$, we have

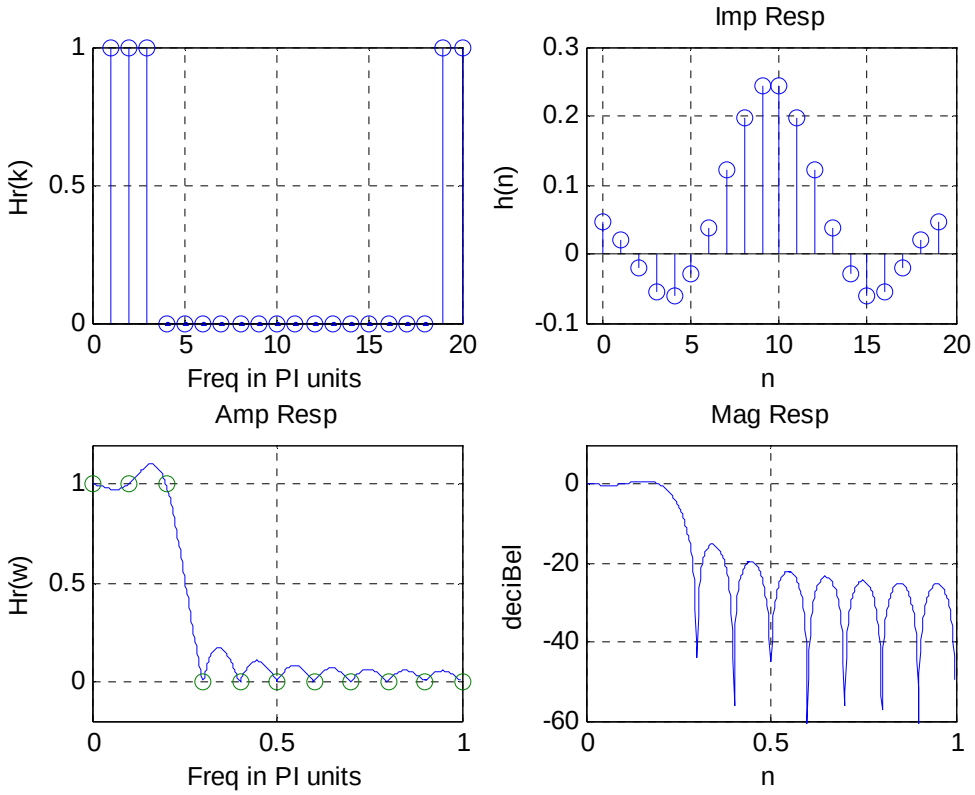


4.3.2. Frequency Sampling method

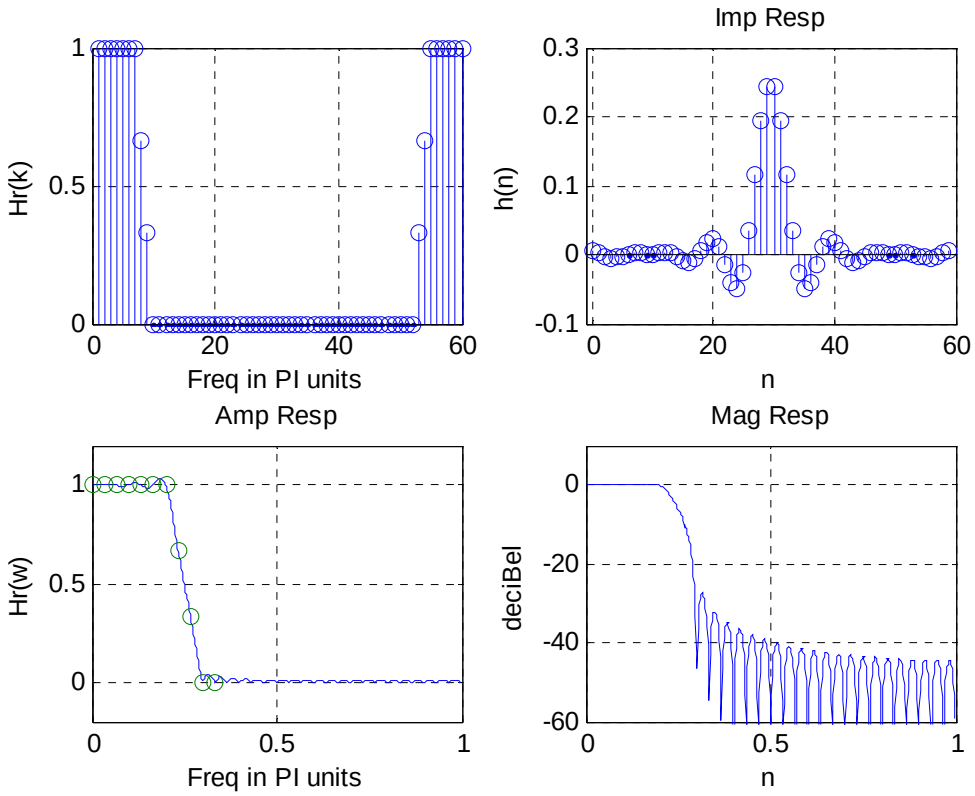
Given a desired frequency response, the filter coefficient could be obtained by inverse DFT of the desired frequency response. By Eq(5.17), we have

$$H\left(e^{j\omega}\right)=\left(\frac{1-e^{-j\omega M}}{M}\right)\sum_{k=0}^{M-1}\frac{H(k)}{1-e^{-j\omega}e^{j\frac{2\pi}{M}k}} \tag{58}$$

```
clear all

M = 20;
ll = 0:M-1;
wl = (2*pi/M)*ll;

% Ref Samples
Hrs = [ones(1,3), zeros(1,M-5), ones(1,2)];   % Ideal Amp Resp sampled

k1 = 0:floor((M-1)/2);           % 1st half
k2 = floor((M-1)/2)+1:M-1; % 2nd half

% Linear phase
alpha = (M-1)/2;
Hang = [-k1, (M-k2)]*(2*pi*alpha/M);

% DFT
H = Hrs .* exp(j*Hang);

h = real(ifft(H,M));

[HH, ww] = freqz(h,1);

HHmag = abs(HH);
HHdb = 20*log10(HHmag);

clf(gcf)
figure(gcf)


subplot(2,2,1); plot(wl(1:11)/pi,Hrs(1:11),'o');
      axis([0,1, -0.1,1.1]); title('Freq Sample M = 20')
      xlabel('Freq in PI units'); ylabel('Hr(k)')

subplot(2,2,2); stem(ll, h); axis([-1,M,-0.1,0.3]); grid
      title('Imp Resp'); xlabel('n'); ylabel('h(n');

subplot(2,2,3), plot(ww/pi,HHmag, wl(1:11)/pi,Hrs(1:11),'o');
      axis([0,1, -0.2,1.2]); title('Amp Resp')
      xlabel('Freq in PI units'); ylabel('Hr(w)')

subplot(2,2,4), plot(ww/pi,HHdb); grid
      title('Mag Resp'); xlabel('n'); ylabel('h(n)');
      ylabel('deciBel')
      axis([0,1 -60,10]);
```



With transition band samples




```
clear all

M = 60;
ll = 0:M-1;
wl = (2*pi/M)*ll;
```

```
% Ref Samples
Hrs = [ones(1,7), 0.6666, 0.3333, zeros(1,43), 0.333, 0.6666, ones(1,6)];% Ideal Amp Resp sampled

k1 = 0:floor((M-1)/2);           % 1st half
k2 = floor((M-1)/2)+1:M-1; % 2nd half

% Linear phase
alpha = (M-1)/2;
Hang = [-k1, (M-k2)]*(2*pi*alpha/M);

% DFT
H = Hrs .* exp(j*Hang);

h = real(ifft(H,M));

[HH, ww] = freqz(h,1);

HHmag = abs(HH);
HHdb = 20*log10(HHmag);

clf(gcf)
figure(gcf)

% subplot(2,2,1); plot(wl(1:11)/pi,Hrs(1:11),'o'); grid
%      axis([0,1, -0.1,1.1]); title('Freq Sample M = 20')
subplot(2,2,1); stem(Hrs); grid
       xlabel('Freq in PI units'); ylabel('Hr(k)')

subplot(2,2,2); stem(ll, h); axis([-1,M,-0.1,0.3]); grid
       title('Imp Resp'); xlabel('n'); ylabel('h(n');

subplot(2,2,3), plot(ww/pi,HHmag, wl(1:11)/pi,Hrs(1:11),'o'); grid
       axis([0,1, -0.2,1.2]); title('Amp Resp')
       xlabel('Freq in PI units'); ylabel('Hr(w)')

subplot(2,2,4), plot(ww/pi,HHdb); grid
       title('Mag Resp'); xlabel('n'); ylabel('h(n)');
       ylabel('deciBel')
       axis([0,1 -60,10]);
```

5. Multirate DSP Systems

5.1. Decimation By Factor D

Given a downsampled signal

$$\tilde{x}(n)=\begin{cases}x(n), & n=0,\pm D,\pm 2D,\cdots\\ 0, & otherwise\end{cases} \tag{59}$$

(59)

Clearly, $\tilde{x}(n)$ can be viewed as a sequence obtained by multiplying $x(n)$ with a periodic train of impulse $p(n)$, with period D. The DFS of $p(n)$ is

$$\tilde{x}(n)=\begin{cases}1, & n=0,\pm D,\pm 2D,\cdots\\ 0, & otherwise\end{cases}=\frac{1}{D}\sum_{k=0}^{D-1}e^{j\frac{2\pi}{D}kn} \tag{60}$$

We have

$$Y(z)=\sum_{m=-\infty}^{\infty}y[m]z^{-m}=\sum_{m=-\infty}^{\infty}\tilde{x}[mD]z^{-m}=\sum_{m=-\infty}^{\infty}\tilde{x}[m]z^{-m/D} \tag{61}$$

By Eq(59) & (60), Eq(61) becomes

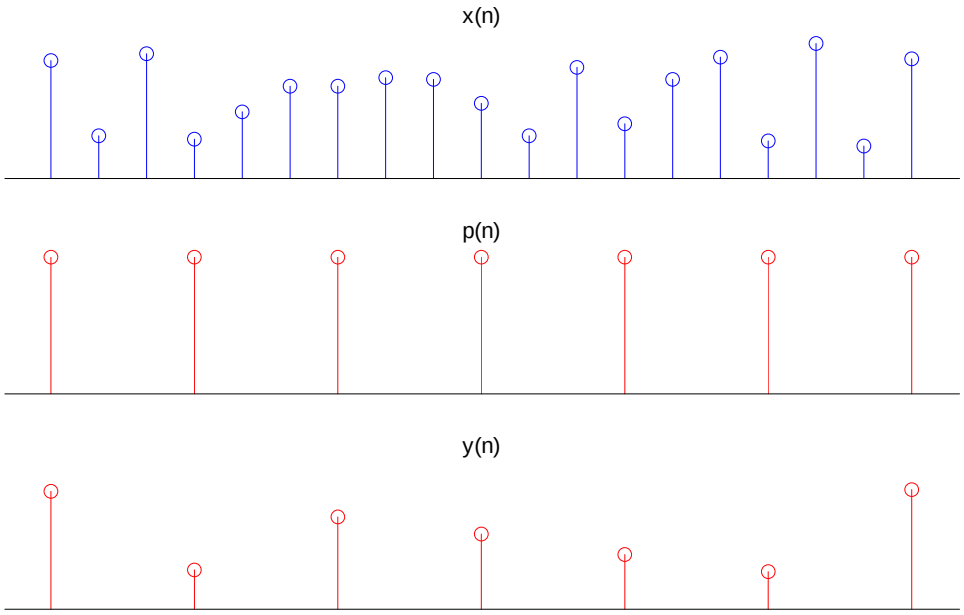
$$Y(z)=\sum_{m=-\infty}^{\infty}x[m]\left(\frac{1}{D}\sum_{k=0}^{D-1}e^{j\frac{2\pi}{D}km}\right)z^{-m/D}=\frac{1}{D}\sum_{k=0}^{D-1}\sum_{m=-\infty}^{\infty}x[m]\left(z^{1/D}e^{-j\frac{2\pi}{D}k}\right)^{-m}$$

By z-transform, we have

$$Y(z)=\frac{1}{D}\sum_{k=0}^{D-1}X\left(z^{1/D}e^{-j\frac{2\pi}{D}k}\right) \tag{62}$$

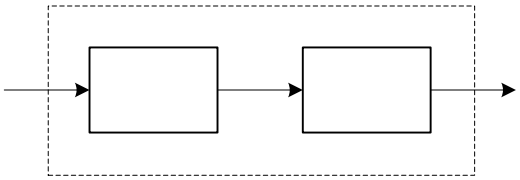
or

$$Y(e^{j\omega})=\frac{1}{D}\sum_{k=0}^{D-1}X\left(e^{j\frac{\omega-2\pi k}{D}}\right)=\frac{1}{D}\sum_{k=0}^{D-1}X\left(\frac{\omega-2\pi k}{D}\right) \tag{63}$$



5.2. Ideal Decimator

The ideal decimator must have an ideal LPF with cut-off frequency equal to decimated one.



So, by Eq(63), we have

$$Y(e^{j\omega_y})=\frac{1}{D}\sum_{k=0}^{D-1}H\left(\frac{\omega_y-2\pi k}{D}\right)X\left(\frac{\omega_y-2\pi k}{D}\right)$$

$$Y(e^{j\omega_y})=\frac{1}{D}\left(H\left(\frac{\omega_y}{D}\right)X\left(\frac{\omega_y}{D}\right)+H\left(\frac{\omega_y-2\pi}{D}\right)X\left(\frac{\omega_y-2\pi}{D}\right)+\cdots+H\left(\frac{\omega_y-2\pi(D-1)}{D}\right)X\left(\frac{\omega_y-2\pi(D-1)}{D}\right)\right)$$

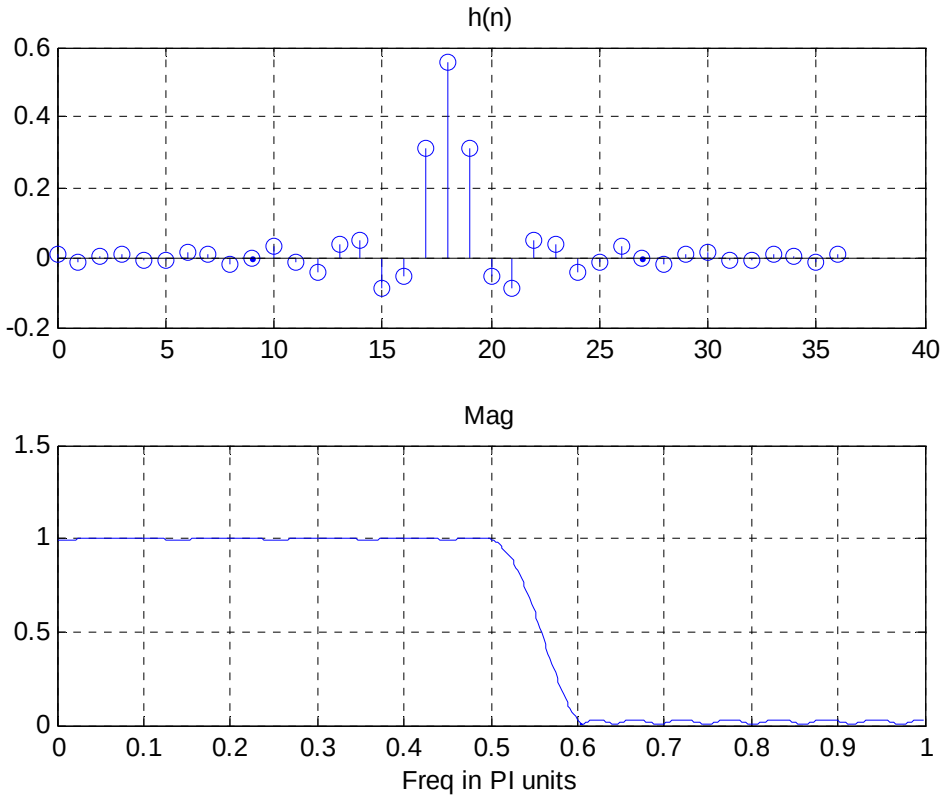
To be ideal decimator, all but the 1st term vanish

$$Y(e^{j\omega_y})=\frac{1}{D}H_D\left(\frac{\omega_y}{D}\right)X\left(\frac{\omega_y}{D}\right) \tag{64}$$

Example

Design a decimator with D = 2, Rp = 0.1, As = 30, wp = π/D, ws = wp + 0.1*π

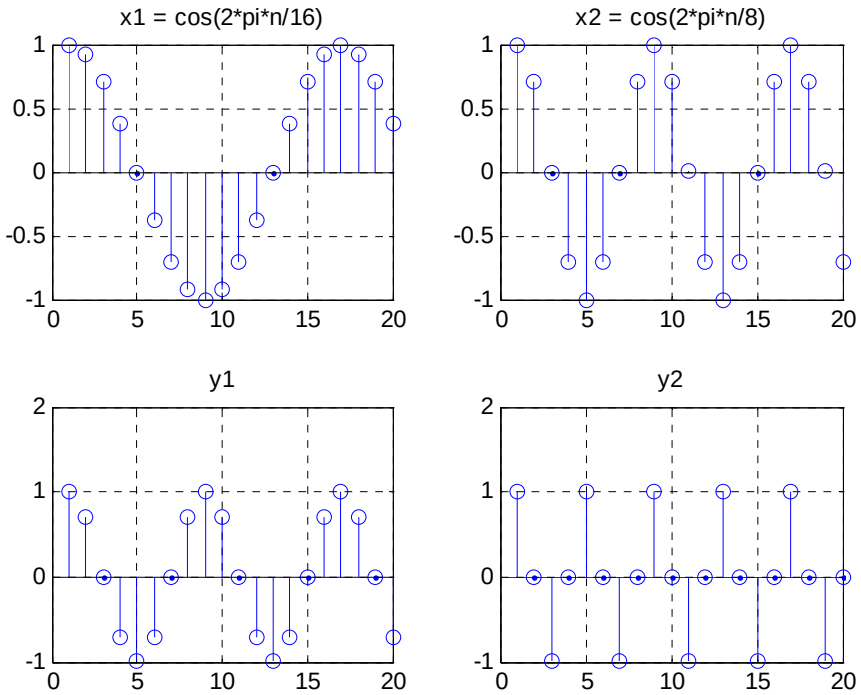
```
% Decimator Design

clear all

D = 2;
Rp = 0.1;
As = 30;
wp = pi/D;
ws = wp+0.1*pi;

% Convert db to delta
[delta1,delta2] = db2delta(Rp,As);

% Estimate filter params using Parks-McClellan optimal equiripple FIR filter design
[N,F,A,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2],2);
h = firpm(N,F,A,weights);
n = [0:length(h)-1];

[H,w] = freqz(h,1);
Hmag = abs(H);

clf
figure(gcf)
subplot(2,1,1), stem(n,h), grid
   title('h(n)')
subplot(2,1,2), plot(w/pi, Hmag), grid
   title('Mag')
   xlabel('Freq in PI units')
```



Example

The same decimator above, but with input

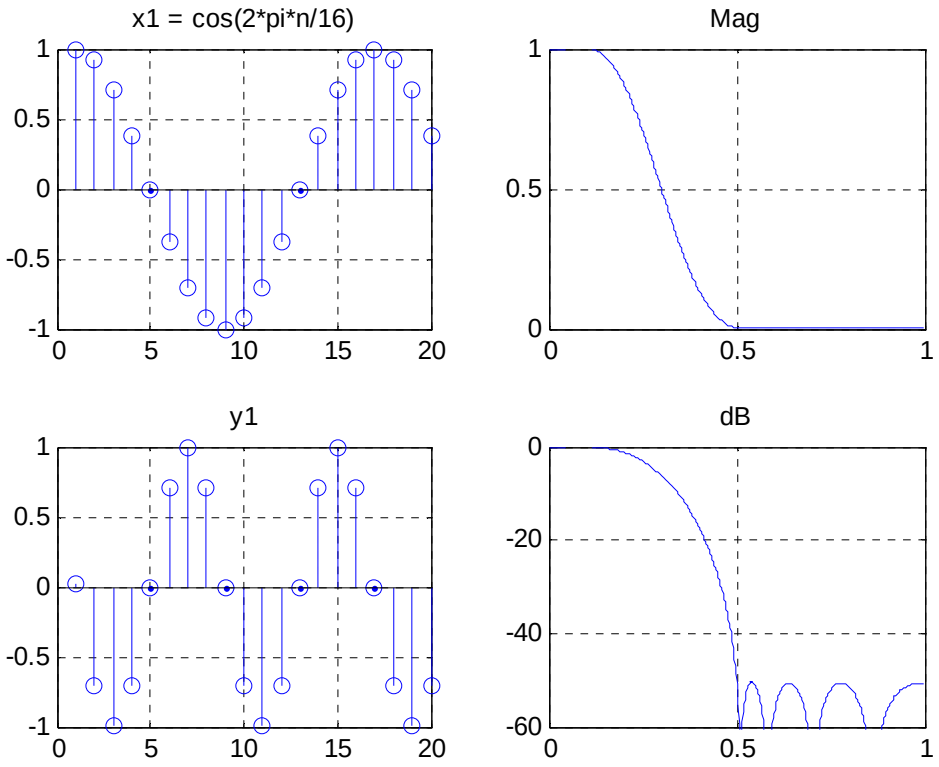
```
% Decimator Design

clear all

D = 2;
Rp = 0.1;
As = 30;
wp = pi/D;
ws = wp+0.1*pi;

% Convert db to delta
[delta1,delta2] = db2delta(Rp,As);

% Estimate filter params using Parks-McClellan optimal equiripple FIR filter design
[N,F,A,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2],2);
h = firpm(N,F,A,weights);


% Input 1
n = [0:256];
x1 = cos(2*pi*n/16);

% Input 2
x2 = cos(2*pi*n/8);

% Decimate
y1 = downsample(conv(x1,h),D);
y2 = downsample(conv(x2,h),D);

clf
figure(gcf)
subplot(2,2,1), stem(x1(1:20)), grid
   title('x1 = cos(2*pi*n/16)')
subplot(2,2,3), stem(y1(N/2:N/2+19)), grid
   title('y1')

subplot(2,2,2), stem(x2(1:20)), grid
   title('x2 = cos(2*pi*n/8)')

subplot(2,2,4), stem(y2(N/2:N/2+19)), grid
   title('y2')
```



Example

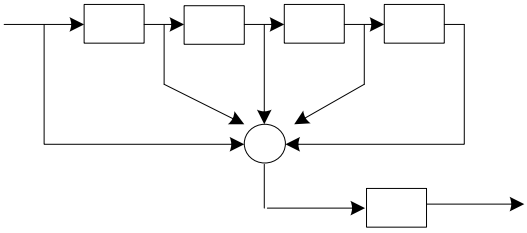
The decimator above with better stopband attenuation As = 50 dB

```
% Decimator Design

clear all

D = 2;
Rp = 0.1;
As = 50;

wxp = pi/8;
wxs = pi/D;

wp = wxp;
ws = (2*pi/D)-wxs;

% Convert db to delta
[delta1,delta2] = db2delta(Rp,As);

% Estimate filter params using Parks-McClellan optimal equiripple FIR filter design
[N,F,A,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2],2);
N = 2*ceil(N/2);
h = firpm(N,F,A,weights);


% Input 1
n = [0:256];
x1 = cos(2*pi*n/16);

% Input 2
x2 = cos(2*pi*n/8);

% Decimate
y1 = downsample(conv(x1,h),D);
y2 = downsample(conv(x2,h),D);

% Spectrum
[H,w] = freqz(h,1);
Hmag = abs(H);
Hdb = 20*log10(Hmag);

clf
figure(gcf)
subplot(2,2,1), stem(x1(1:20)), grid
   title('x1 = cos(2*pi*n/16)')
subplot(2,2,3), stem(y1(N/2:N/2+19)), grid
   title('y1')

subplot(2,2,2), plot(w/pi, Hmag), grid
   title('Mag')
   axis([0 1 0 1])

subplot(2,2,4), plot(w/pi, Hdb), grid
   title('dB')
   axis([0 1 -60 0])
```



6. FIR Low-Pass Filter

Low-Pass Filter is an integrator with action of accumulation. So we could have a simple FIR low-pass filter like below



So, for *M*-tab FIR low-pass filter

$$y_k = \frac{x_k + x_{k-1} + x_{k-2} + \cdots + x_{k-P+1}}{P}$$

similarly, we have

$$y_{k-1} = \frac{x_{k-1} + x_{k-2} + x_{k-3} + \cdots + x_{k-P}}{P}$$

subtract these two, we have

$$y_k - y_{k-1} = \frac{x_k - x_{k-P}}{P}$$

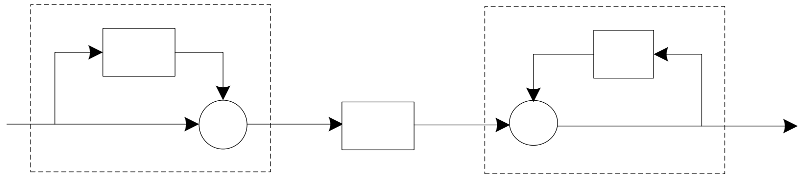
or

$$y_k = y_{k-1} + \frac{x_k - x_{k-P}}{P}$$

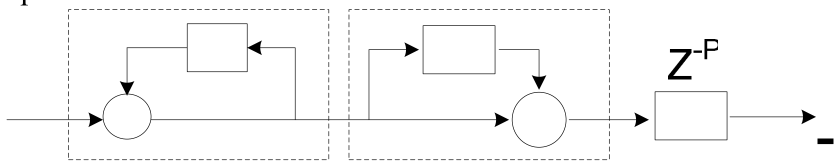
This version takes much less HW resource than the original one

Hence, the transfer function is

$$H_z = \frac{1}{P}\frac{1-z^{-P}}{1-z^{-1}} \tag{65}$$



For a linear system, it's equivalent to



X

1/P

From Eq(65) we have frequency response, *ie* spectrum, ignoring the factor 1/*P*, below

+

$$H\left(e^{j2\pi f}\right) = \frac{1-e^{-j2\pi fP}}{1-e^{-j2\pi f}} = \frac{e^{-j\pi fP}\left(e^{+j\pi fP}-e^{-j\pi fP}\right)}{e^{-j\pi f}\left(e^{+j\pi f}-e^{-j\pi f}\right)} = e^{-j\pi f(N-1)}\frac{\sin \pi fP}{\sin \pi f}$$

Comb

so the frequency magnitude is

$$\left|H\left(e^{j2\pi f}\right)\right| = \frac{\sin \pi fP}{\sin \pi f} \tag{66}$$

```
% cicdn01d.m

% CIC Decimation : Digital Down Converter

clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% M = 10;
D = input('Delay value <4> : ');

if isempty(D)
       D = 4;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

R = 10;
% R = input('R decimation value <10> : ');

if isempty(R)
       R = 10;
end

M = R * D;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Fs = 8e3;     % 8 KHz
Ts = 1/Fs;

N = 250;

tt = Ts*[0:N-1];

Fx1 = 100;
Fx2 = 1200;

xx = sin(2*pi*Fx1*tt) + sin(2*pi*Fx2*tt);
```

Z-1

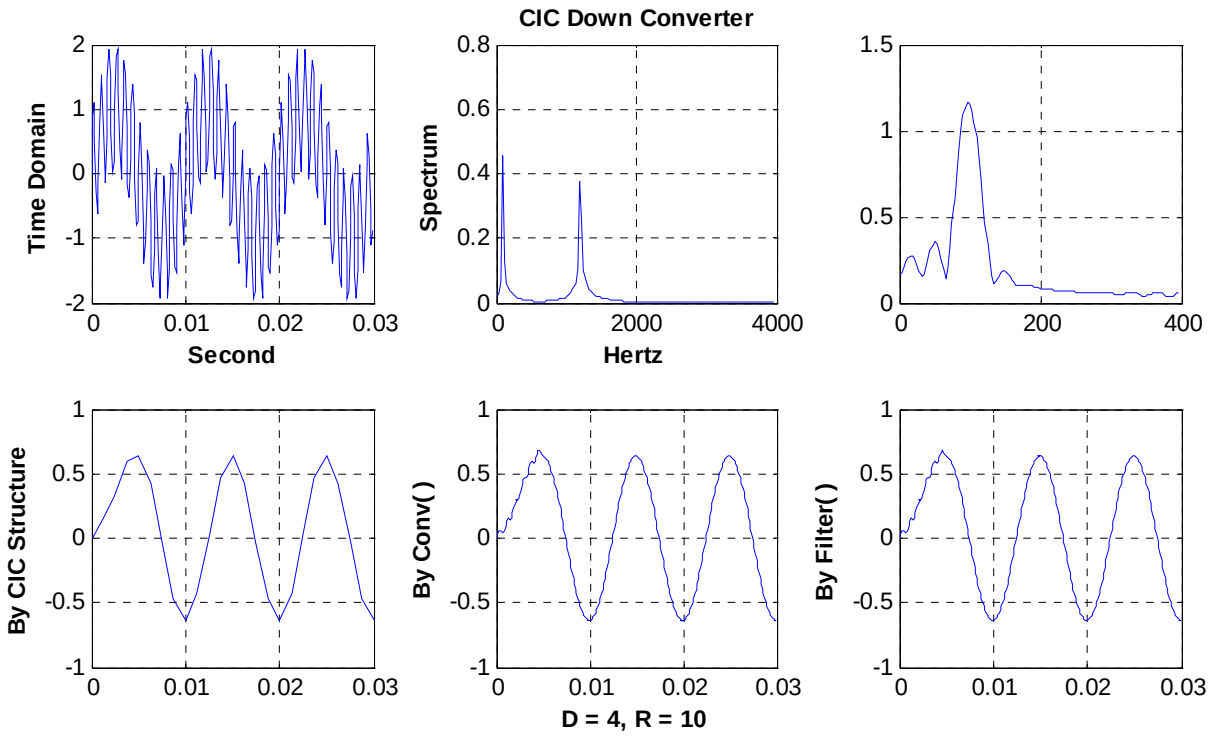
X +

Integrator

```
% Script for a digital implementation of RC low pass filter

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Integrator - Comb

dlyBuf = zeros(1,N);
intOut = 0;
y3n_ii = 1;

for ii = 1:N
       % integrator
       intOut = intOut + xx(ii);

       % comb section
       combOut = intOut - dlyBuf(M);
       dlyBuf(2:end) = dlyBuf(1:end-1);
       dlyBuf(1) = intOut;

       % Decimation
       if mod(ii,R)==1
              y3n(y3n_ii) = combOut;
              y3n_tt(y3n_ii) = (y3n_ii - 1)*R*Ts;
              y3n_ii = y3n_ii + 1;
       end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Integrator - Comb

dlyBuf = zeros(1,N);
intOut = 0;
y4n_ii = 1;

for ii = 1:N
       % integrator
       intOut = intOut + xx(ii);

       % Decimation
       if mod(ii,R)==1
              % comb section
              combOut = intOut - dlyBuf(D);
              dlyBuf(2:end) = dlyBuf(1:end-1);
              dlyBuf(1) = intOut;

              y4n(y4n_ii) = combOut;
              y4n_tt(y4n_ii) = (y4n_ii - 1)*R*Ts;
              y4n_ii = y4n_ii + 1;
       end

end

err34 = y3n - y4n;
err34rms = sqrt(err34*err34'/length(err34));  % '

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NN = 2^round(log2(N));

NN2 = NN/2;
ff = (Fs/NN)*[0:NN2-1];

xx_sp = (1/NN)*fft(xx,NN);
xx_mag = abs(xx_sp(1:NN/2));

Y_NN = NN;
Y_NN2 = Y_NN/2;
Y_ff = (1/R)*(Fs/Y_NN)*[0:Y_NN2-1];

y3n_sp = (1/Y_NN)*fft(y3n,Y_NN);
y3n_mag = abs(y3n_sp(1:Y_NN/2));

y4n_sp = (1/Y_NN)*fft(y4n,Y_NN);
y4n_mag = abs(y4n_sp(1:Y_NN/2));

str = sprintf('D = %d, R = %d', D, R);
```

```
[y_m, y_i] = max(y4n_mag);

fprintf('Err between CIC Type 1 and 2 %f with max spectrum at %f Hz\n', err34rms, Y_ff(y_i));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CIC : (R*M)-cascade LPF - CONV()

S_N = 1;     %% # stages in cascade

hrec = ones(1,R*D);
hcic = 1;

for k=1:S_N
    hcic = conv(hcic, hrec);     %% cascading
end;

[Hcic, wt] = freqz(hcic, 1, 4096, Fs);      %% CIC Freq. Response
Mcic = 20*log10(abs(Hcic)/max(abs(Hcic)));     %% CIC Freq. Response

% output by conv()
yn = conv(hcic, xx);

yn = yn/M;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% By : num, den, filter()
num = zeros(1,M+1);
num(1) = 1;
num(M+1) = -1;

den = [1 -1];

y_F = filter(num,den, xx)/M;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_lim = N - 10;

tt = tt(1:N_lim);
xx = xx(1:N_lim);

y4n = y4n / M;
clf
figure(1)
subplot(2,3,1), plot(tt,xx), grid
       xlabel('Second', ...
              'FontWeight','bold')
       ylabel('Time Domain', ...
              'FontWeight','bold')

subplot(2,3,2), plot(ff,xx_mag), grid
       title('CIC Down Converter', ...
              'FontWeight','bold')
       xlabel('Hertz', ...
              'FontWeight','bold')
       ylabel('Spectrum', ...
              'FontWeight','bold')

subplot(2,3,3), plot(Y_ff,y4n_mag), grid

subplot(2,3,4), plot(y4n_tt, y4n), grid
       ylabel('By CIC Structure', ...
              'FontWeight','bold')

subplot(2,3,5), plot(tt,yn(1:length(tt))), grid
       ylabel('By Conv( )', ...
              'FontWeight','bold')
       xlabel(str, ...
              'FontWeight','bold')

subplot(2,3,6), plot(tt,y_F(1:length(tt))), grid
       ylabel('By Filter( )', ...
              'FontWeight','bold')
```

Simulation results with 40 tabs

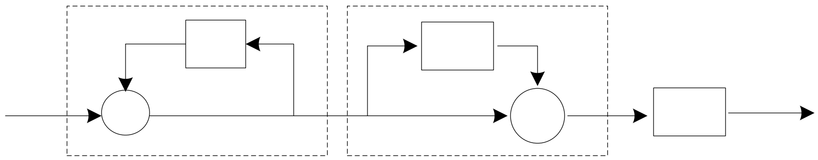


Remark 2

- There's no transient in FIR implementation
- Spectrum has ratio between 1st peak and 2nd one of value 47, a lot better than IIR with valua of 9

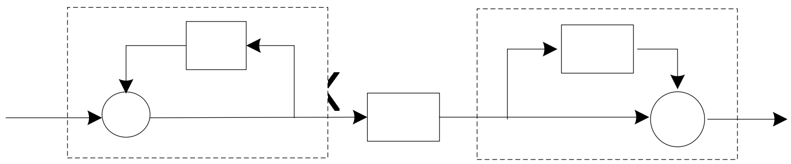
6.1. CIC Decimation DDC

Decimation Type-I



Z^{-1}

or, equivalently Type-II

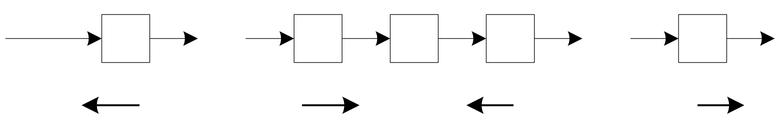


Integrator

Due to delay of integrator, the delay rate into Comb of Type-II is R times slower than Type-I, so Comb in Type-II must have delay rate R times smaller, *ie* P → P/R

Output taken at every *R*th, ie one and ignore th

Use Type-II for simplicity, we have a cascade of *N* stages below



Z^{-1}

By Eq(66) with $f \to f/R$, we have frequency response

$$\left|H\left(e^{j2\pi f}\right)\right| = \left(\frac{\sin\frac{\pi f P}{R}}{\sin\frac{\pi f}{R}}\right)^N = \left(\frac{\sin \pi M R f_R}{\sin \pi f_R}\right)^N, \quad f_R = \frac{f}{R} \tag{67}$$

↓R

6.2. CIC Spectrum

Integrator

We look at 2 ways to get CIC spectrum (1) using Transfer Function (2) using SINC() function

Output taken at every *R*th, ie one and ignore

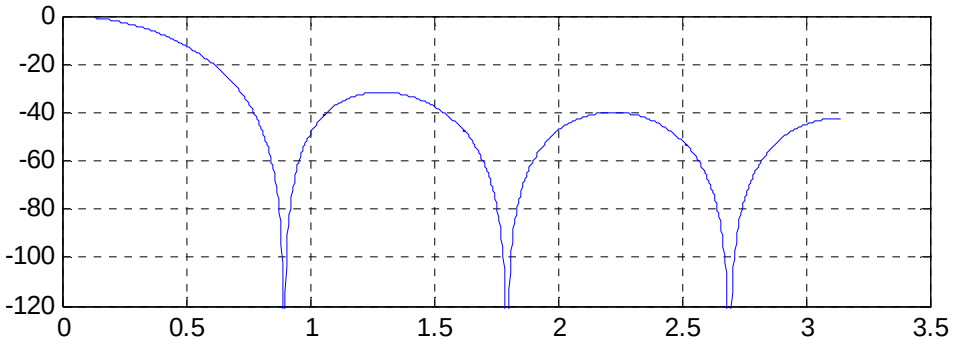
Note that the Delay factor reduces to R times

```
% cicsp01.m

clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% N = 4;
N = inp('Stage N : <4> : ', 4);

% M = 1;
M = inp('Delay M : <1> : ', 1);

% R = 7;
R = inp('Rate R : <7> : ', 7);

P = M*R;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

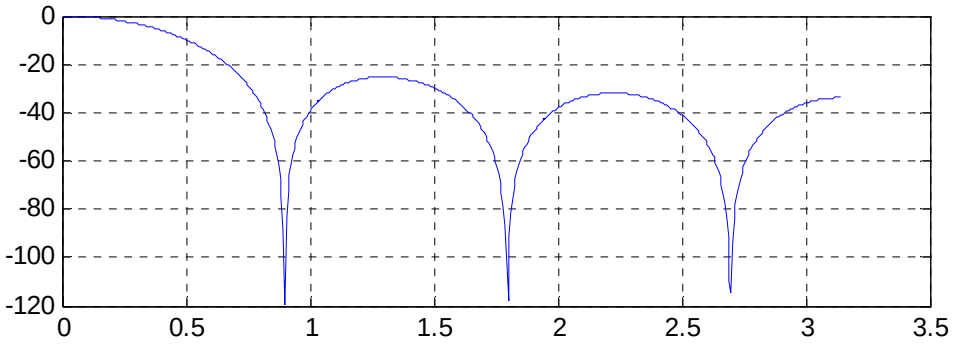

X Hi · · · Hi

```
NN = 1024;

num = zeros(1,P+1);
num(1) = 1;
num(P+1) = -1;

den = [1 -1];

num1 = 1;
den1 = 1;

for k=1:N+1
   num1 = conv(num1, num);
   den1 = conv(den1, den);
end

[H w] = freqz(num1, den1, NN);

Hmag = abs(H);

start = 20;

Hmag = Hmag(start:end);
w = w(start:end);

Hmag = Hmag/max(Hmag);
Hdb = 10*log10(Hmag);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ff = 0:0.001:0.5;

num = sin(pi*ff*M*R);
den = sin(pi*ff);

hh = (num ./ den) .^ N;

hh = hh/max(hh);
hhdb = 10*log10(hh);

hhdb = hhdb(start:end);
ff = ff(start:end);

ff = max(w)*ff/max(ff);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

str = sprintf('N = %d, M = %d, R = %d', N, M, R);

clf
% figure(gcf)
figure(1)

subplot(2,1,1), plot(w, Hdb), grid
   axis([0 3.5 -120 0])
   xlabel(['Spectrum by Transfer Func : ' str], ...
      'FontWeight','bold')

subplot(2,1,2), plot(ff,hhdb), grid
   axis([0 3.5 -120 0])
   xlabel(['Spectrum by Sinc() func : ' str], ...
      'FontWeight','bold');

figure(2)
plot(w, Hdb), grid
%  axis([0 3.5 -150 0])
   xlabel([str], ...
      'FontWeight','bold');
```

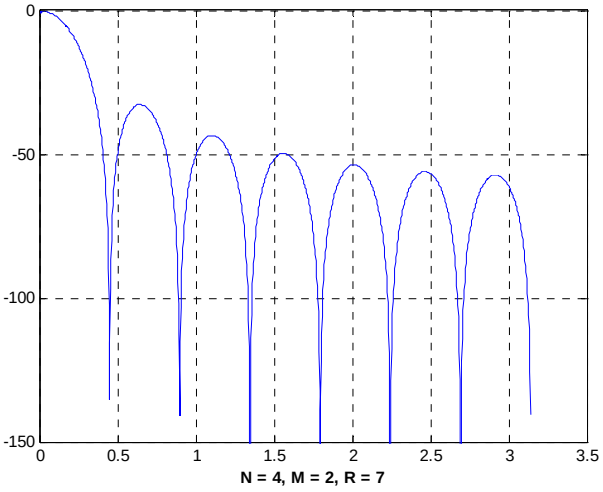
Below is CIC spectrum by using transfer function and by using SIN()/SIN() function. They appear identical as expected.



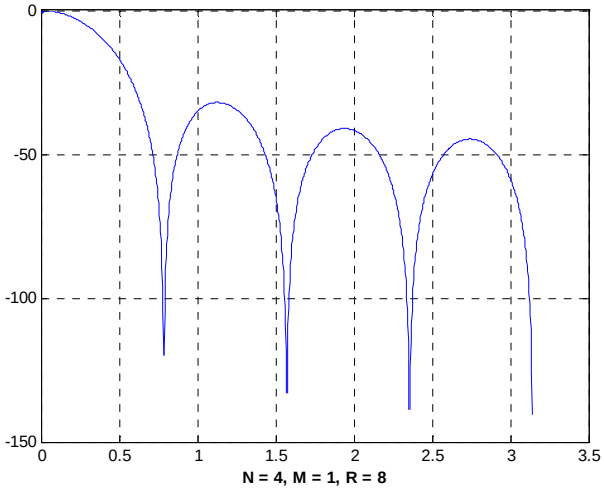
Spectrum by Transfer Func : N = 4, M = 1, R = 7



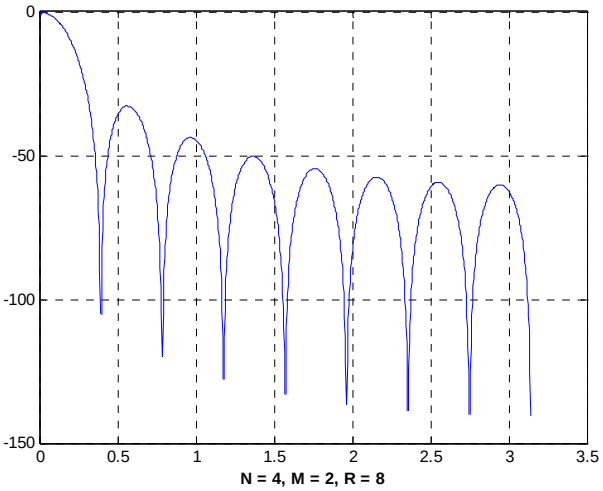
Spectrum by Sinc() func : N = 4, M = 1, R = 7

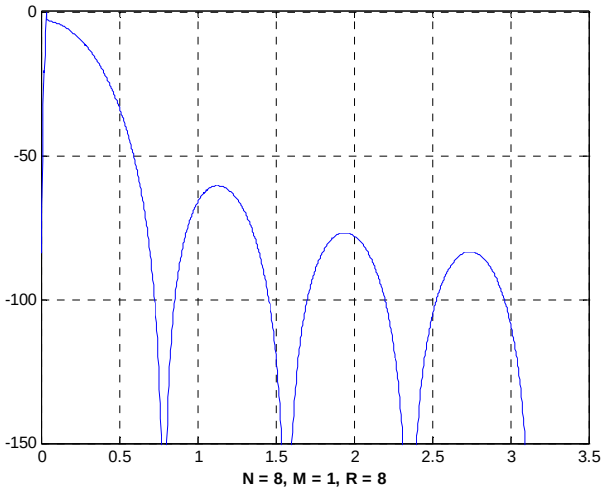


N = 4, M = 2, R = 7

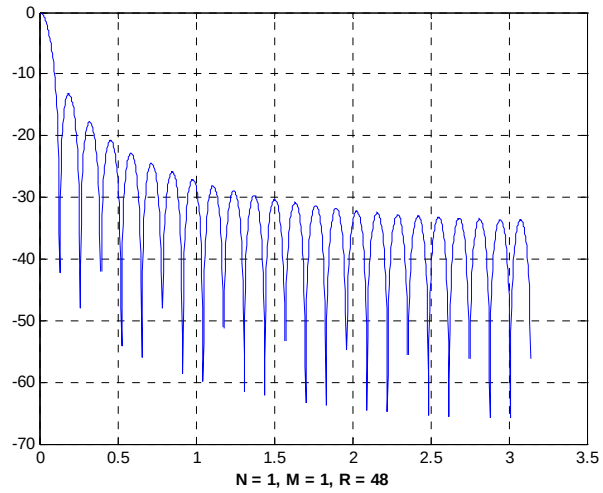


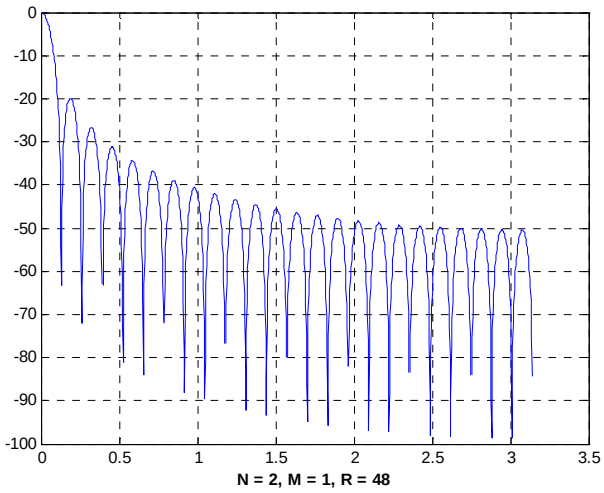
N = 4, M = 1, R = 8

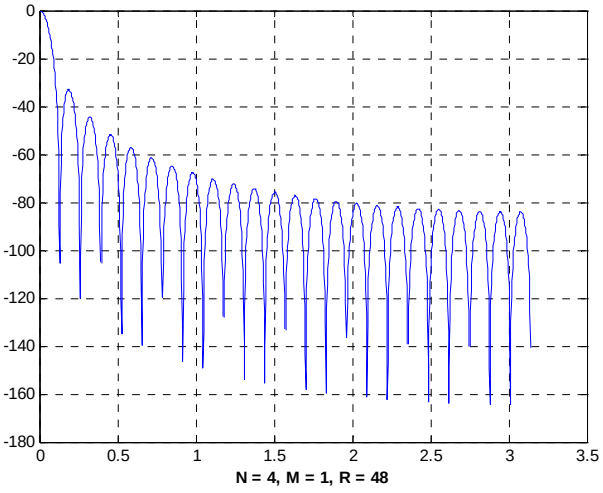


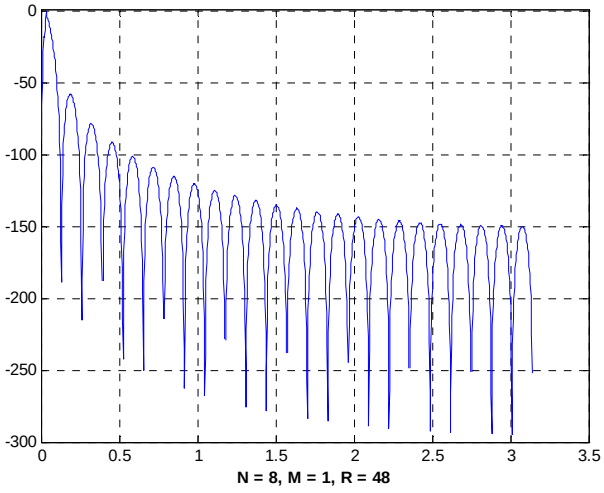


Increasing number of stages









These figures show the frequency response plots for various combinations of the design parameters N, R and M. Few last figures provide insight into the filter behavior as the number of integrator and differentiator (N) stages are varied. In all these cases, the differential delay is held constant at M = 1 and the sample rate change is fixed at R = 48. For any CIC filter, there are always RM zeros in the transfer function. The zeros are equally spaced

around the unit cirlce in the z-plane at integer multiples of 1 / RM - there is of course no zero at z = 0. Increasing N has the effect of increasing the order of the zeros. This, in turn, increases the attenuation at frequencies in the locality of the zero. This effect is clearly illustrated in Figures 9 to 13 where we see increasing attenuation of the filter sidelobes as N is increased. Also note that as the order of the zeros increase, the passband droop also increases, thus narrowing the filter bandwidth. The increased droop may not be acceptable in some applications. The droop is frequently corrected using an additional (non-CIC-based) stage of filtering after the CIC decimator. In the case of a CIC interpolator, the signal may be precompensated to account for the impact in the passband as the signal is up-sampled by the CIC interpolator.

6.3. CIC Compensation Filter

A compensation filter (not part of the CIC Core) can be used to flatten the passband frequency response. For a CIC decimator, the compensation filter operates at the decimated sample rate. The compensation filter provides $(x/\sin(x))^N$ shaping. An example of a 3rd order (N = 3) R = 64 compensated CIC system is shown in Figure below. The plot shows the uncompensated CIC frequency response, the compensation filter frequency response and the compensated CIC. In this case, since the number of CIC stages is 3, the compensation filter has a cubic response of the form $(x/\sin(x))^3$.

To flatten the spectrum we need to work on precise spectrum equation, by Eq(1) and Eq(2), we have

$$|H(f)| = \frac{1}{MR} \times \frac{\sin \pi f M}{\sin \frac{\pi f}{R}} \tag{68}$$

Use its inverse for compensation

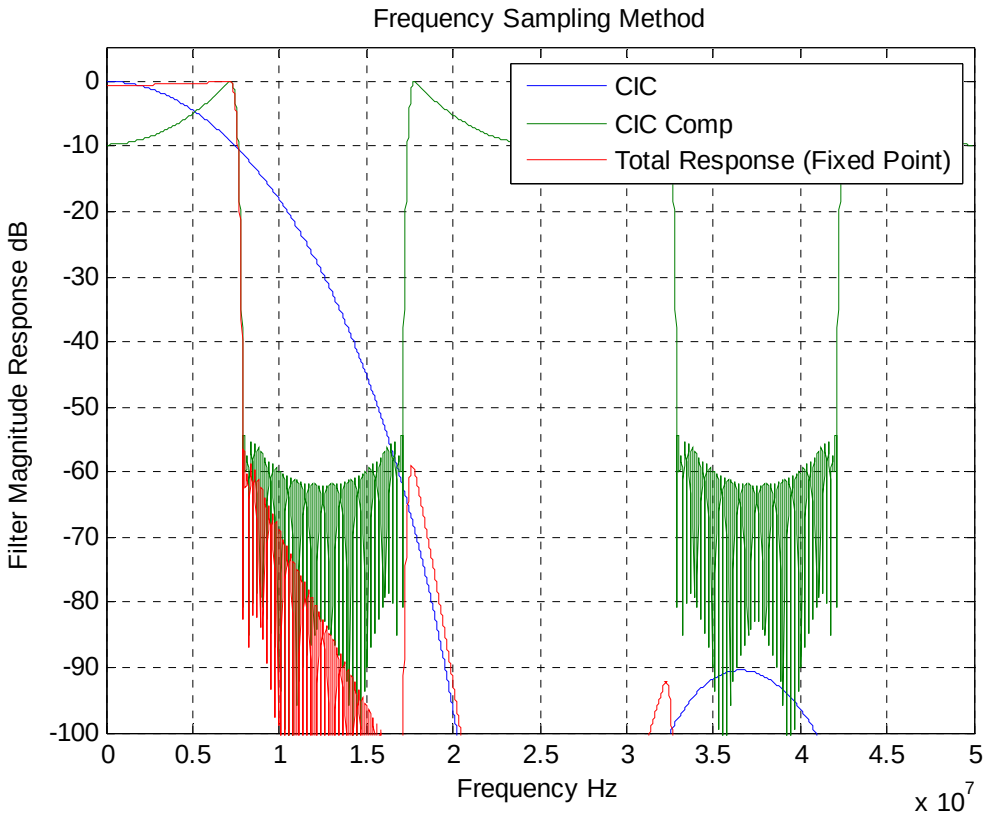
$$|G(f)| = MR \times \frac{\sin \frac{\pi f}{R}}{\sin \pi f M} \approx MR \times \frac{\frac{\pi f}{R}}{\sin \pi f M} = \frac{\pi f M}{\sin \pi f M} \tag{69}$$

For *N*-stage

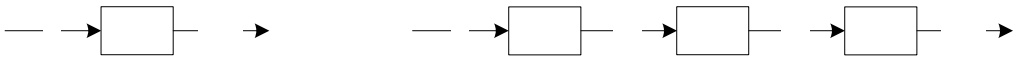
$$|G_N(f)| \approx \left(\frac{x}{\sin x} \right)^N, \quad x = \pi f M \tag{70}$$

```
%% CIC compensating filter design

clear all
close all

%%%%%% CIC filter parameters %%%%%%
R = 4;       %% Decimation factor
M = 1;      %% Differential Delay

N = 8;    %% Number of Stages

B = 18;     %% Number of bits to represent fixed point filter coefficients
Fs = 100e6; %% (High) Sampling frequency in Hz (before decimation)

Fc = 7.55e6; %% Cut-off Freq for Passband edge in Hz

%%%%%%% fir2.m parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = 120;  %% Order of filter taps; must be an even number

Fco = R*Fc/Fs;      %% Normalized Cutoff freq; 0<Fco<=0.5/M; xdong: modified 10/30/06: taken out
/M.
                    %% Fco should be less than 1/(4M), if not, bad performance is guranteed
% Fco = 0.5/M;      %% use Fco=0.5 if we don't care responses outside passband

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ff_o = [ff_po ff_so] : passband stopband
```

```
F_NN = 4000;                    %% Must be Even : end = 1
ff_pso = [0:F_NN]/(2*F_NN);             %% End = 0.5

end_po = 1 + max(find(ff_pso < Fco));   %% passband
ff_po = ff_pso(1:end_po);
ff_so = ff_pso(end_po+1:end);

ff_o = [0:F_NN]/F_NN;           %% End = 1.0

%%%%%%% CIC Compensator Design using fir2.m %%%%%%
% Inverrse CIC
FF_po = ones(1,length(ff_po));   %% Passband response; FF_po(1)=1



xx = pi*M*ff_po(2:end);

%% FF_po(2:end) = abs( M*R*sin(pi*ff_po(2:end)/R)./sin(pi*M*ff_po(2:end))).^N; %% Inverse sinc
FF_po(2:end) = abs( xx ./sin(xx) ).^N; %% Inverse sinc

FF_so = [FF_po zeros(1,length(ff_so))];

h = fir2(L,ff_o,FF_so);    %% Filter Coeff with length L+1
h = h/max(h);              %% Floating point coefficients

%% Interested Freq. is at output after decimation, so it's Low Freq.
%% Simulation is at High Freq. before decimation, so upsample
hp = upsample(h, R);
[Hciccomp, wt] = freqz(hp, 1, 4096, Fs);                    %% CIC Comp. response using fir2
Mciccomp = 20*log10(abs(Hciccomp)/max(abs(Hciccomp)));  %% CIC Comp. response using fir2


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CIC : (R*M)-cascade LPF

hrec = ones(1,R*M);
tmph = hrec;

for k=1:N-1
    tmph = conv(hrec, tmph);     %% cascading
end;
hcic = tmph;
hcic = hcic/norm(hcic);

[Hcic, wt] = freqz(hcic, 1, 4096, Fs);      %% CIC Freq. Response
Mcic = 20*log10(abs(Hcic)/max(abs(Hcic)));    %% CIC Freq. Response

%%%%%%% Total Response %%%%%%%%%%%%%%%

%% Floating Point
ht = conv(hcic, hp);                   %% Concatenation of CIC and floating-point C_FIR at high
freqency
[Ht, wt] = freqz(ht, 1, 4096, Fs);     %% Total response for CIC + floating point C_FIR
Mt = 20*log10(abs(Ht)/max(abs(Ht)));   %% Total response for CIC + floating point C_FIR

% Fixed Point
hz = round(h*power(2,B-1)-1);    %% Quantization of filter coefficients
hzp = upsample(hz,R);

hzt = conv(hcic, hzp);                      %% Concatenation of CIC and fixed-point C_FIR at high
freqency
[Hzt, wt] = freqz(hzt, 1, 4096, Fs);        %% Total response for CIC + fixed point C_FIR
Mzt = 20*log10(abs(Hzt)/max(abs(Hzt)));     %% Total response for CIC + fixed point C_FIR

%%%%%%% Display Response %%%%%%%%%%%%%%%

figure;
       plot(wt, Mcic, wt, Mciccomp, wt, Mzt);
       legend('CIC','CIC Comp','Total Response (Fixed Point)')
       ylim([-100 5]);
       title('Frequency Sampling Method');
       grid
       xlabel('Frequency Hz');
       ylabel('Filter Magnitude Response dB');
```



7. Fractional Rate Resampler



A LPF is designed at Fs*I for a I/R resampler. So, there're few methods to check its spectrum given coefficient set Cn, aka impulse response

- FFT(Cn, N) where N/2 is at 0.5*Fs*I, *ie*. highest rate
- Yn = conv(Cn, randn); FFT(Yn, N) where N/2 is at 0.5*Fs*I/R, *ie*. output rate

8. Some Notes on MatLab for DSP

8.1. conv( )

For system input/output

```
y(n) = conv(h, x);
```

For cascading system blocks

```
hrec = ones(1,R*D);
hcic = 1;

for k=1:S_N
    hcic = conv(hcic, hrec);     %% cascading
end;
```

8.2. filter( )

System input/out from difference equation

```
y = filter(num, den, x);
```

8.3. fir2( )

Filter coefficients for filter of length N with with the frequency response specified by vectors F and A such that PLOT(F,A) would show a plot of the desired frequency response

```
h = fir2(N, F, A);
```

8.4. freqz( )

Frequency response

```
[H, w] = freqz(num, den, N);

Hang = angle(H);
Hmag = abs(H);
HdB = 20*log10(Hmag);
```

8.5. spectrum

Spectrum could be found in different ways

- FREQZ() as above
- FFT(Cn, N) where N/2 @ 0.5*I*Fs for I/R resampler or I=1 in regular LPF
- Yn=conv(Cn, randn); FFT(Yn, N) where N/2 @ 0.5*I*Fs/R for I/R resampler or I=R=1 in regular LPF