Signal Processing

Young Won Lim
4/7/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

Based on

Signal Processing with Free Software : Practical Experiments
F. Auger

filter (1)

```
: y = filter (b, a, x)
: [y, sf] = filter (b, a, x, si)
: [y, sf] = filter (b, a, x, [], dim)
: [y, sf] = filter (b, a, x, si, dim)
```

https://octave.sourceforge.io/octave/function/filter.html

filter (2)

Apply a 1-D digital filter to the data x.

filter returns the solution to the following
linear, time-invariant difference equation:

$$\sum_{k=0}^{N} a(k+1)\, y(n-k) \;=\; \sum_{k=0}^{M} b(k+1)\, x(n-k) \qquad \mathit{for}\;\; 1 \le n \le \mathit{length}(x)$$

where N=length(a)-1 and M=length(b)-1.

$$\boldsymbol{a} = [a(1), a(2), \cdots, a(N+1)]$$
$$\boldsymbol{b} = [b(1), b(2), \cdots, b(M+1)]$$
$$\mathit{length}(\boldsymbol{a}) = N+1$$
$$\mathit{length}(\boldsymbol{b}) = M+1$$

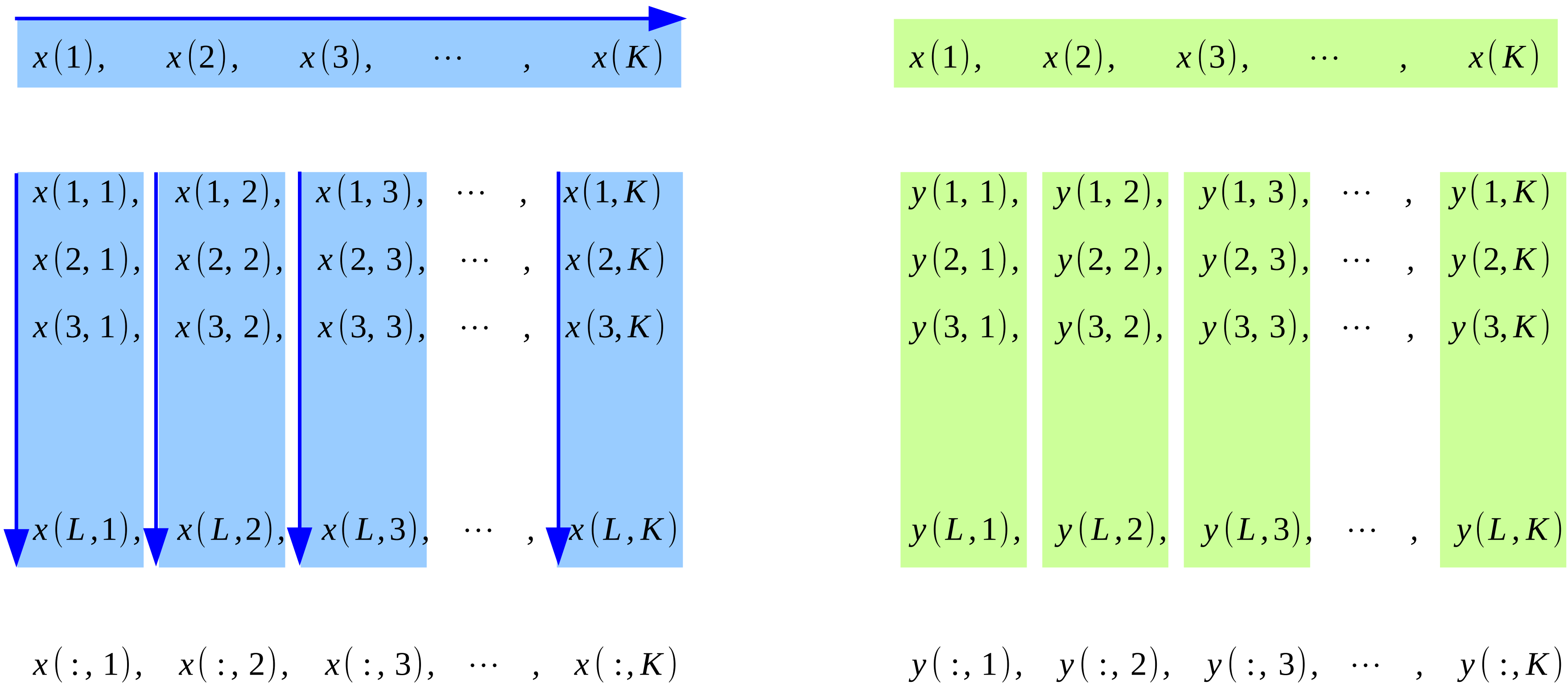
$$\boldsymbol{x} = [x(1), x(2), \cdots, x(L+1)]$$
$$\mathit{length}(\boldsymbol{x}) = L+1$$

$$1 \le n \le L+1$$

https://octave.sourceforge.io/octave/function/filter.html

filter (3)

The result is calculated over the **first** non-singleton dimension of x
or over **dim** if supplied.



https://octave.sourceforge.io/octave/function/filter.html

filter (4)

$$\sum_{k=0}^{N} a(k+1)\,y(n-k) = \sum_{k=0}^{M} b(k+1)\,x(n-k) \qquad \mathit{for}\ \ 1 \le n \le \mathit{length}(x)$$

$$a(1)\,y(n) + \sum_{k=1}^{N} a(k+1)\,y(n-k) = \sum_{k=0}^{M} b(k+1)\,x(n-k)$$

$$a(1)\,y(n) = -\sum_{k=1}^{N} a(k+1)\,y(n-k) + \sum_{k=0}^{M} b(k+1)\,x(n-k)$$

$$y(n) = -\sum_{k=1}^{N} \frac{a(k+1)}{a(1)}\,y(n-k) + \sum_{k=0}^{M} \frac{b(k+1)}{a(1)}\,x(n-k)$$

$$y(n) = -\sum_{k=1}^{N} c(k+1)\,y(n-k) + \sum_{k=0}^{M} d(k+1)\,x(n-k) \qquad \mathit{for}\,1 \le n \le \mathit{length}(x)$$

where c = a/a(1) and d = b/a(1).

https://octave.sourceforge.io/octave/function/filter.html

filter (5)

si : the initial state of the system
sf : the final state

the state vector is a column vector
whose length is equal to the length of
the longest coefficient vector – 1

No **si** is presented, the zero initial state.

in terms of the z transform,
y is the result of passing the discrete-time signal **x**
through a system characterized
by the following rational system function:

$$H(z) = \frac{\sum_{k=0}^{M} d(k+1) z^{-k}}{1 + \sum_{k=1}^{N} c(k+1) z^{-k}}$$

https://octave.sourceforge.io/octave/function/filter.html

freqz (1)

```
: [h, w] = freqz (b, a, n, "whole")
: [h, w] = freqz (b)
: [h, w] = freqz (b, a)
: [h, w] = freqz (b, a, n)
: h = freqz (b, a, w)
: [h, w] = freqz (..., Fs)
: freqz (...)
```

https://octave.sourceforge.io/octave/function/freqz.html

freqz (2)

Return the complex frequency response **h**
of the rational **IIR** filter
with the numerator coefficients **b** and
the denominator coefficients **a**

The response is evaluated
at **n** angular frequencies between **0** and **2*pi**.

The output value **w** is a vector of the frequencies.

h : the frequency response vector
w : the frequency vector

https://octave.sourceforge.io/octave/function/freqz.html

freqz (3)

If **a** is omitted, the denominator is assumed to be **1**
(this corresponds to a simple **FIR** filter).

If n is omitted, a value of 512 is assumed.
For fastest computation, n should factor
into a small number of small primes.

If the fourth argument, "**whole**", is omitted
the response is evaluated
at frequencies between **0** and **pi**.

https://octave.sourceforge.io/octave/function/freqz.html

freqz (4)

freqz (**b**, **a**, **w**)

Evaluate the response
at the specific frequencies in the vector **w**.
The values for **w** are measured in radians.

freqz (…)

Plot the magnitude and phase response
of **h** rather than returning them.

https://octave.sourceforge.io/octave/function/freqz.html

freqz (5)

[…] = **freqz** (…, Fs)

Return frequencies in Hz instead of radians
assuming a sampling rate Fs.
If you are evaluating the response
at specific frequencies **w**,
those frequencies should be requested
in Hz rather than radians.

[**h**, **w**] = **freqz** (**b**, **a**, n, "whole", Fs)
[**h**, **w**] = **freqz** (**b**, Fs)
[**h**, **w**] = **freqz** (**b**, **a**, Fs)
[**h**, **w**] = **freqz** (**b**, **a**, n, Fs)
h = **freqz** (**b**, **a**, **w**, Fs)

https://octave.sourceforge.io/octave/function/freqz.html

freqz_plot

: **freqz_plot** (**w**, **h**)
: **freqz_plot** (**w**, **h**, freq_norm)

Plot the magnitude and phase response of **h**.

If the optional freq_norm argument is true,
the frequency vector **w** is in units of normalized radians.
If freq_norm is false, or not given,
then **w** is measured in Hertz.

https://octave.sourceforge.io/octave/function/freqz_plot.html

conv

: **conv** (**a**, **b**)
: **conv** (**a**, **b**, shape)

Convolve two vectors **a** and **b**.

The output convolution is a vector
with length equal to length (**a**) + length (**b**) - 1.
When **a** and **b** are the coefficient vectors of two polynomials,
the convolution represents
the coefficient vector of the product polynomial.

The optional shape argument may be
shape = "full"
Return the full convolution. (default)
shape = "same"
Return the central part of the convolution with the length(**a**).

https://octave.sourceforge.io/octave/function/conv.html

fftconv

```
: fftconv (x, y)
: fftconv (x, y, n)
```

Convolve two vectors using the FFT for computation.

c = **fftconv** (**x**, **y**) returns
a vector of length equal to length(**x**) + length(**y**) - 1

If **x** and **y** are the coefficient vectors of two polynomials,
the returned value is the coefficient vector of the product polynomial.

The computation uses the FFT
by calling the function **fftfilt**.

If the optional argument n is specified,
an n-point FFT is used.

https://octave.sourceforge.io/octave/function/fftconv.html

deconv

: **deconv** (**y**, **a**)

Deconvolve two vectors.

[**b**, **r**] = **deconv** (**y**, **a**) solves for **b** and **r** such that **y** = **conv** (**a**, **b**) + **r**.

If **y** and **a** are polynomial coefficient vectors,
b will contain the coefficients of the polynomial quotient and
r will be a remainder polynomial of lowest order.

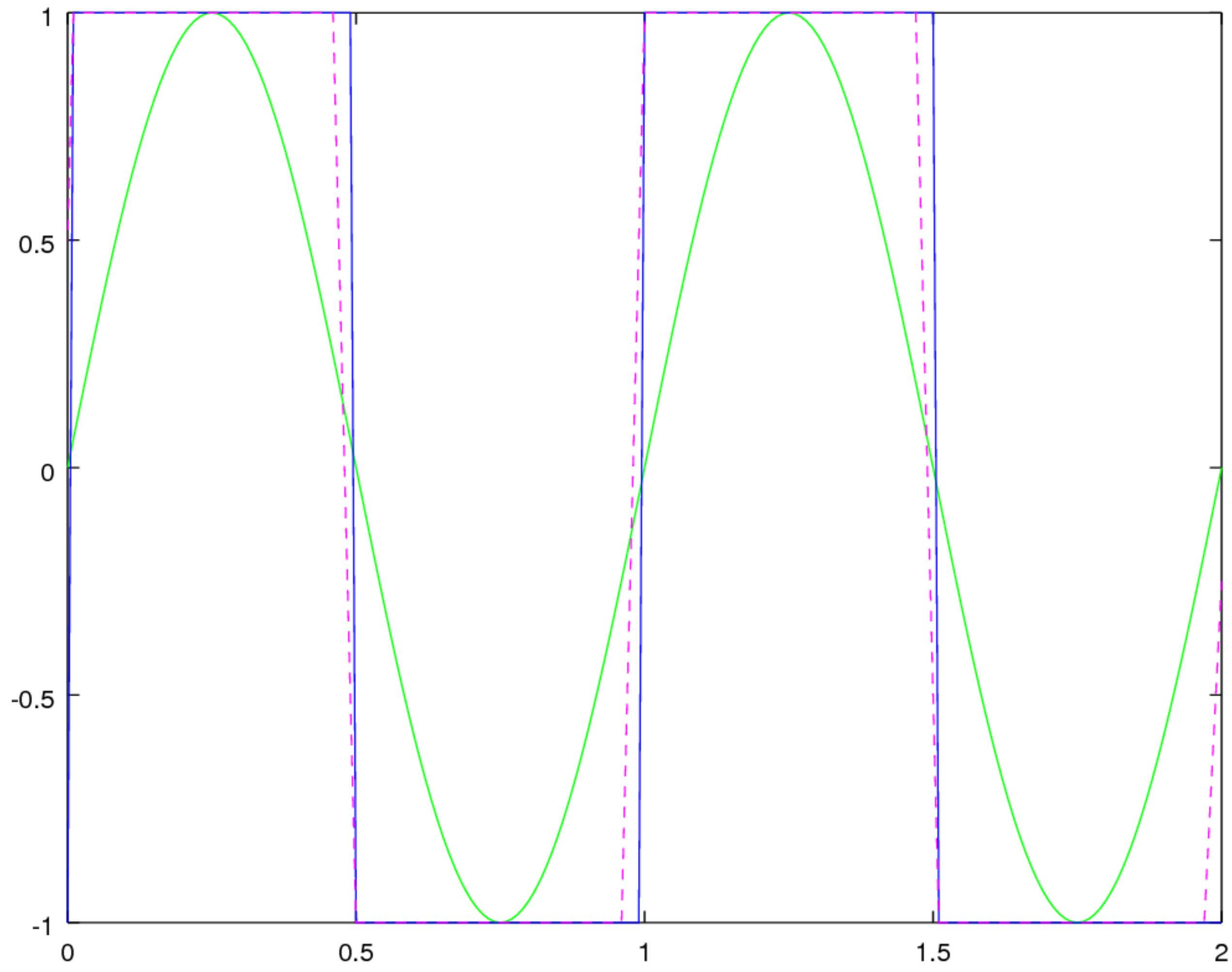
https://octave.sourceforge.io/octave/function/deconv.html

Moving Average Filter

```
t = 0: 1/100 : 1;
s = sin(2 * pi * t);
x  = (s > 0);                    % 1 or 0
x = (x – 0.5) * 2;               % -1 or +1
x = [x 0 0 0];                   % zero padding for i+1, i+2, i+3

for i=1 : length(x)-3
    y(i) = (x(i) + x(i+1) + x(i+2) + x(i+3)) / 4;
endfor

hold
plot(t, s, ‘g’)
plot(t, x)
plot(t, y, ‘m--’);
```

DSP for sound engineers (in Korean), J.W. Chae

Moving Average Filter



DSP for sound engineers (in Korean), J.W. Chae

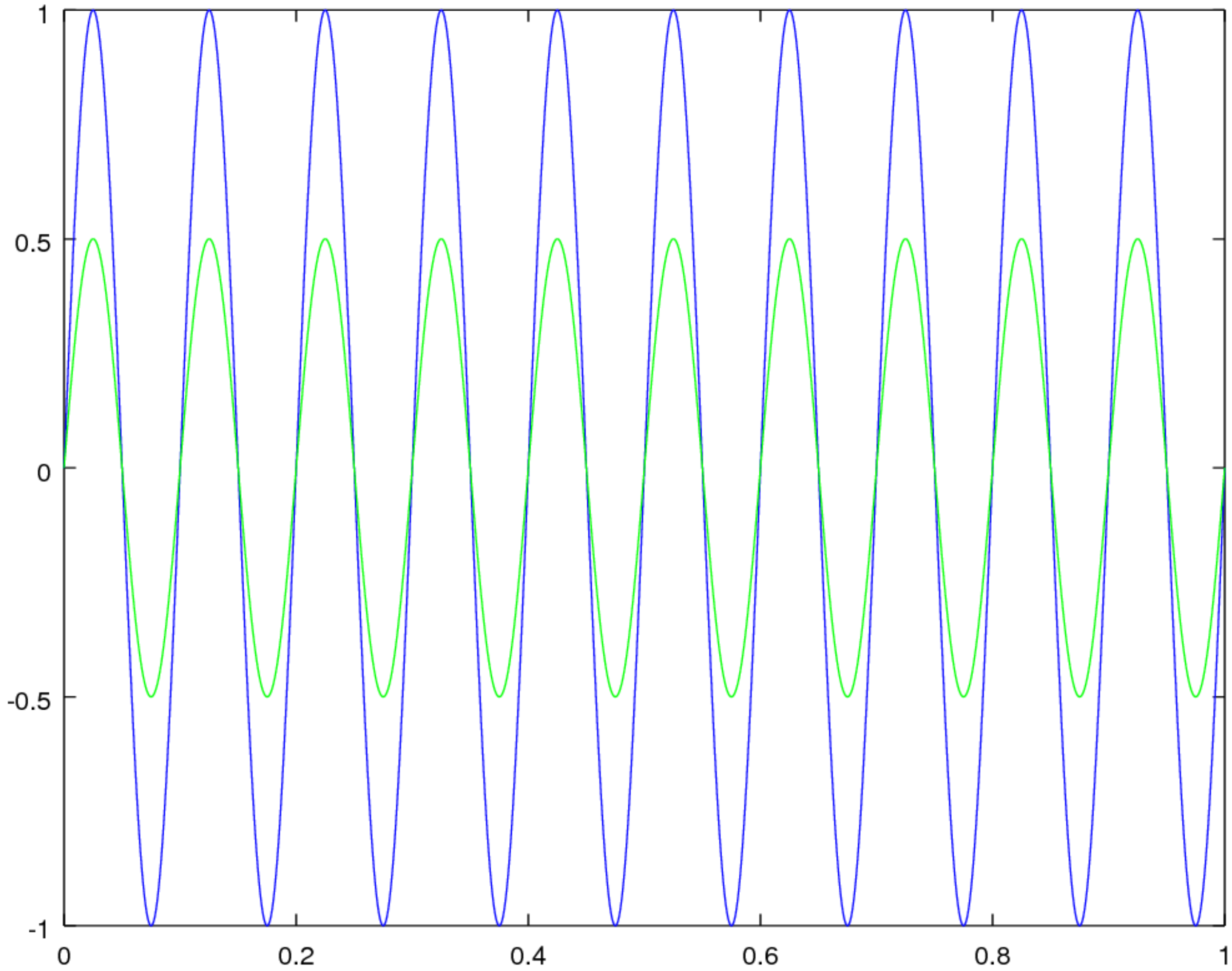
Impulse Response

```
fs=44100;
t=0: 1/fs :1-1/fs;
k = fs * t ;

x=sin(2*pi*10*t);

i=zeros(1, length(t));
i(1)=0.5;

I=fft(i);
X=fft(x);
Y = I .* X;
y = real(ifft(Y));
```

```
clf;
plot(t, x, t, y, 'g')

source "../0.util.octave/util.m"
pause
clf;

TimeFreqPlot(t, x, k, X, 'half')
pause;
TimeFreqPlot(t, i, k, I, 'half')
pause;
```

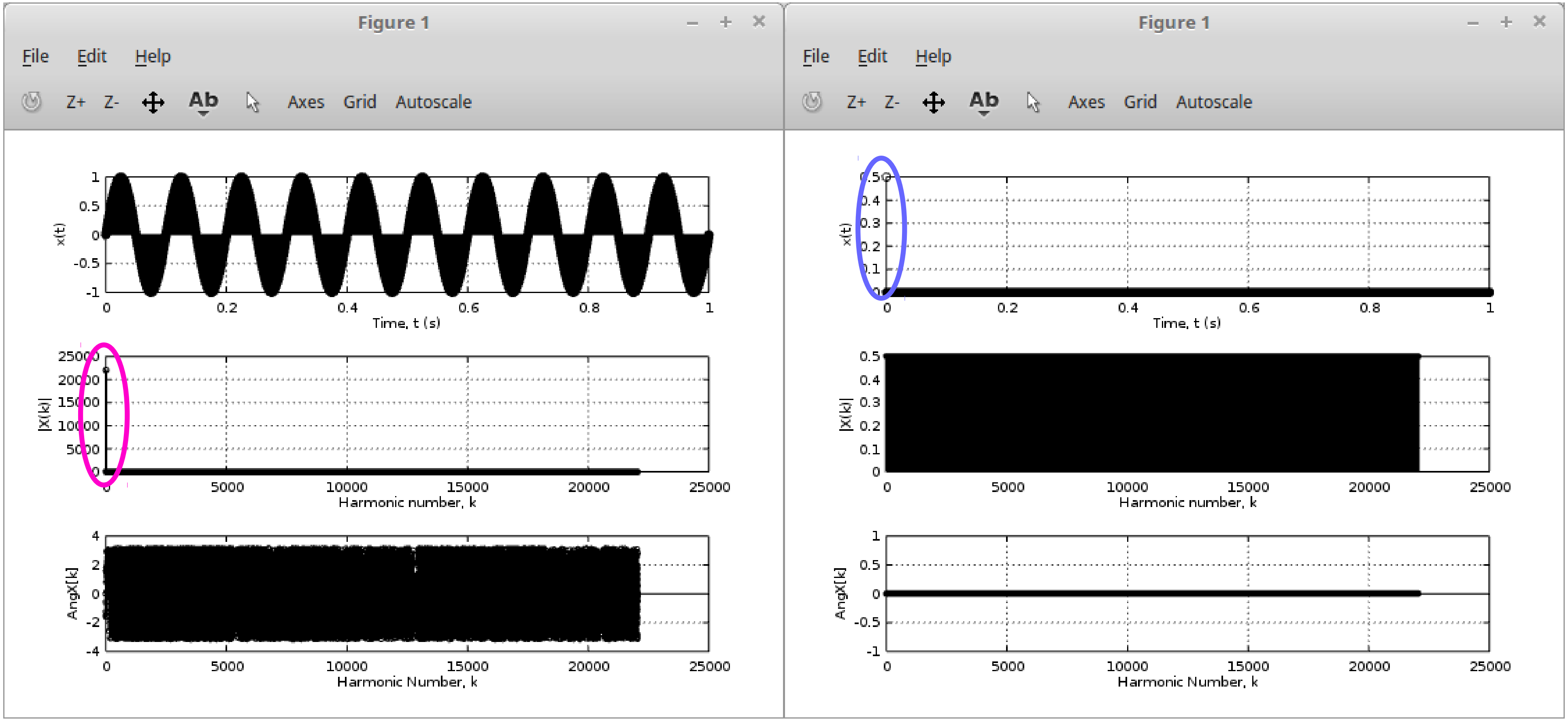
DSP for sound engineers (in Korean), J.W. Chae

Impulse Response



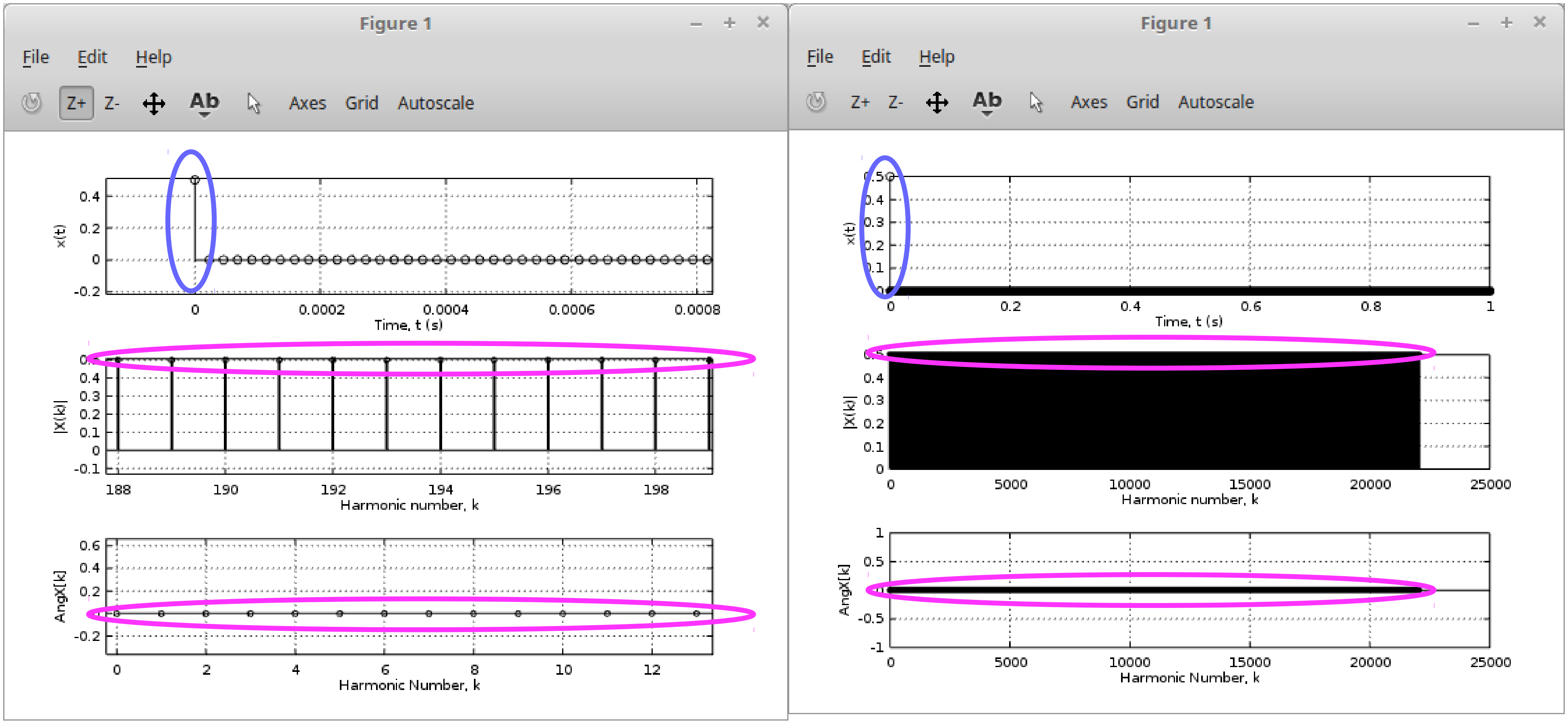
DSP for sound engineers (in Korean), J.W. Chae

Impulse Response



DSP for sound engineers (in Korean), J.W. Chae

Impulse Response



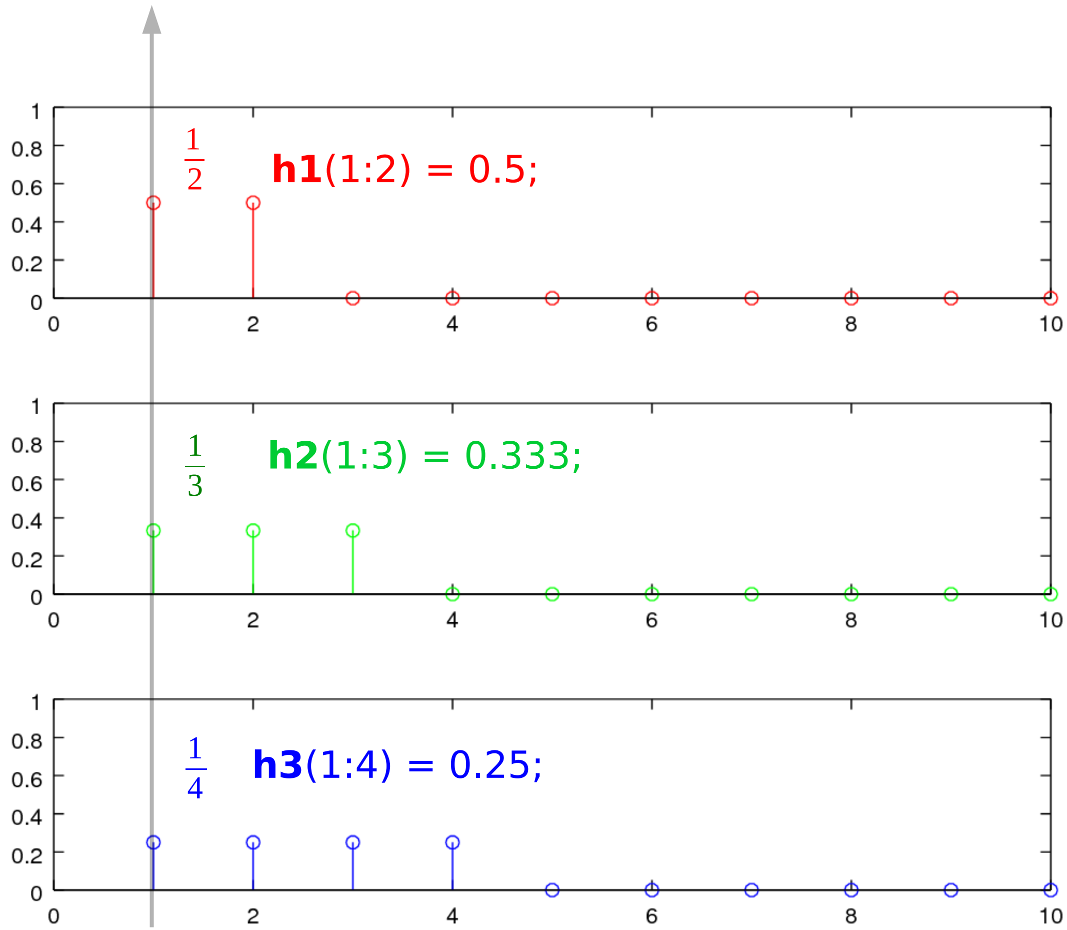
DSP for sound engineers (in Korean), J.W. Chae

FIR - Low Pass filter

```
h1 = zeros(1, 44100);
h1(1:2) = 0.5;
H1 = abs(fft(h1));
H1 = H1(1: 22050);
plot(H1, 'r'); hold on;
```

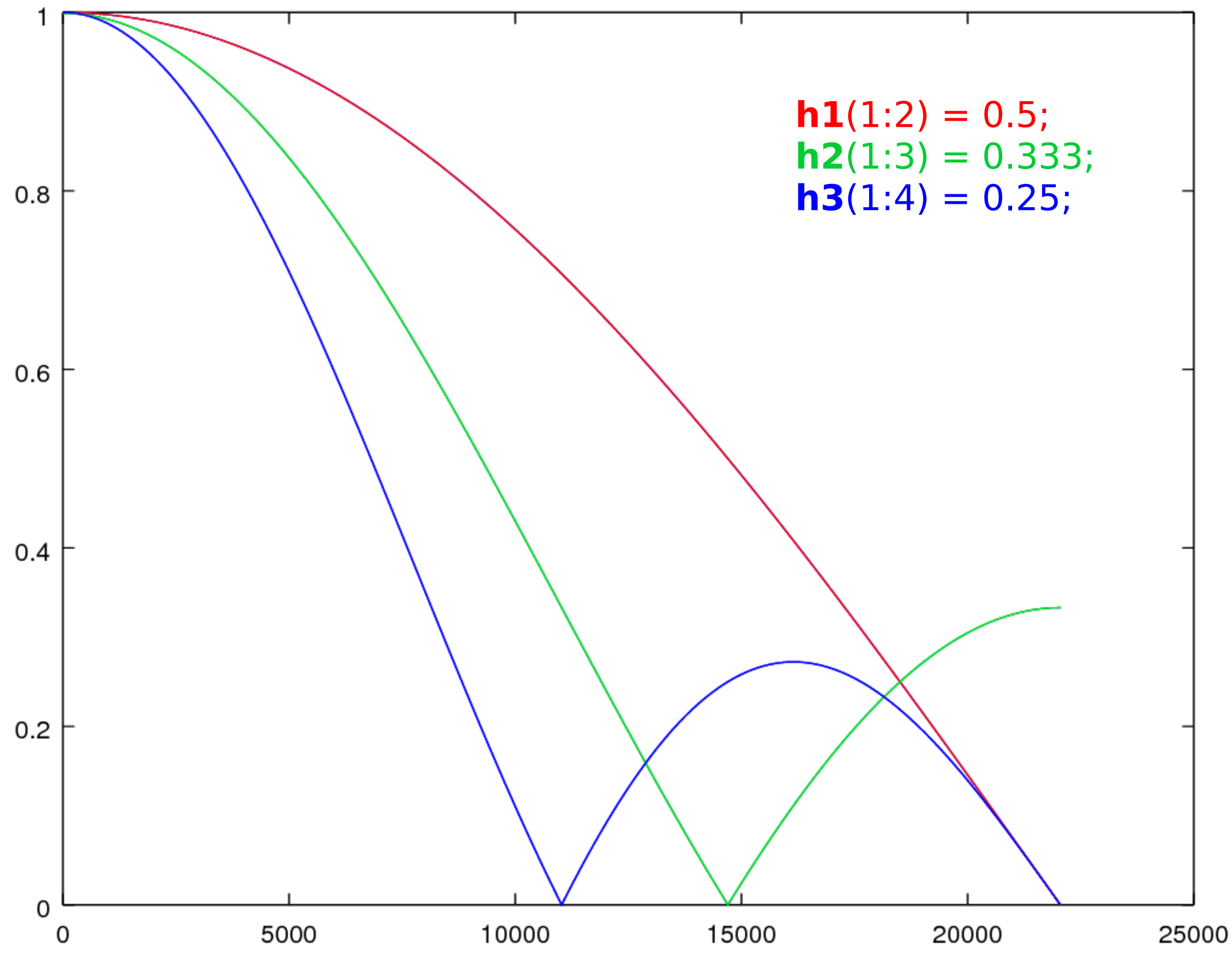
```
h2 = zeros(1, 44100);
h2(1:3) = 0.333;
H2 = abs(fft(h2));
H2 = H2(1: 22050);
plot(H2, 'g'); hold on;
```

```
h3 = zeros(1, 44100);
h3(1:4) = 0.25;
H3 = abs(fft(h3));
H3 = H3(1: 22050);
plot(H3, 'b'); hold on;
```



DSP for sound engineers (in Korean), J.W. Chae

FIR - Low Pass filter

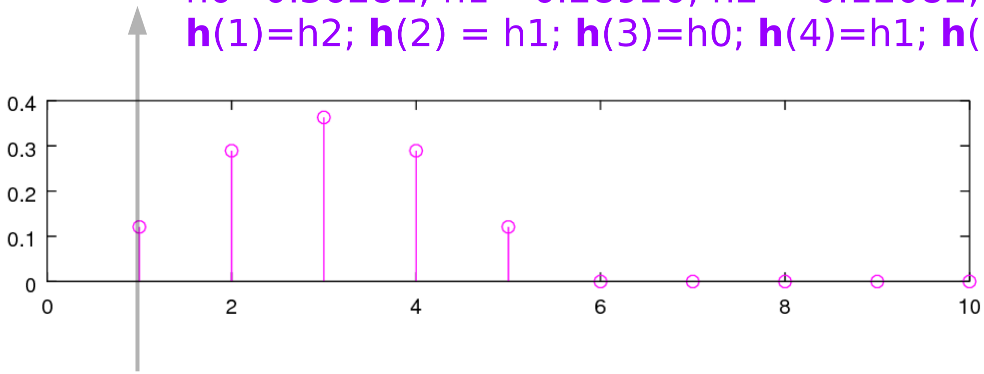


DSP for sound engineers (in Korean), J.W. Chae

FIR - Low Pass filter

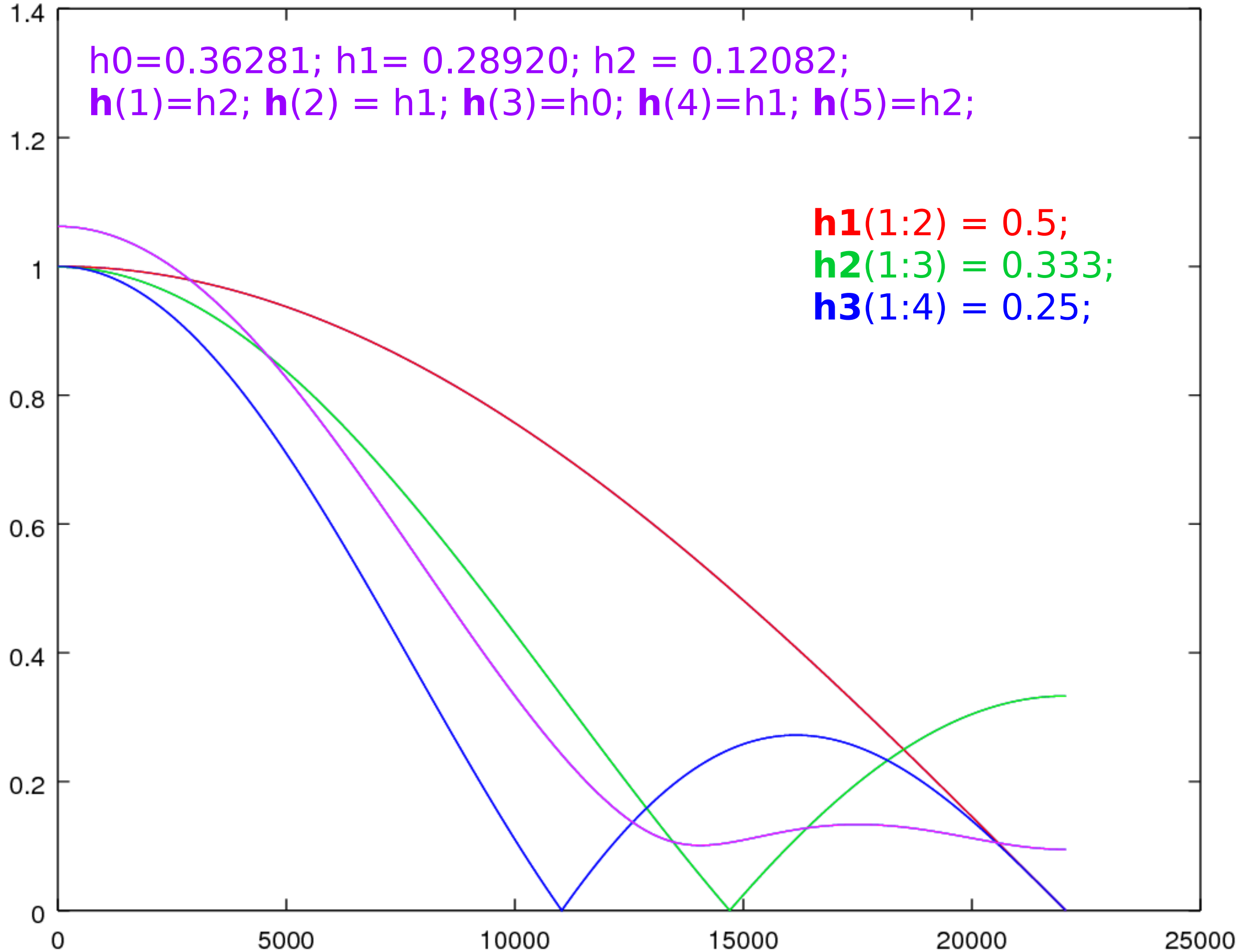
```
h0=0.36281; h1= 0.28920; h2 = 0.12082;

h = zeros(1, 44100);

h(1)=h2; h(2) = h1; h(3)=h0; h(4)=h1; h(5)=h2;
H = abs( fft(h) );
H = H(1: 44100/2);
plot(H, 'v')
```

h0=0.36281; h1= 0.28920; h2 = 0.12082;
h(1)=h2; **h**(2) = h1; **h**(3)=h0; **h**(4)=h1; **h**(5)=h2;



DSP for sound engineers (in Korean), J.W. Chae

FIR - Low Pass filter

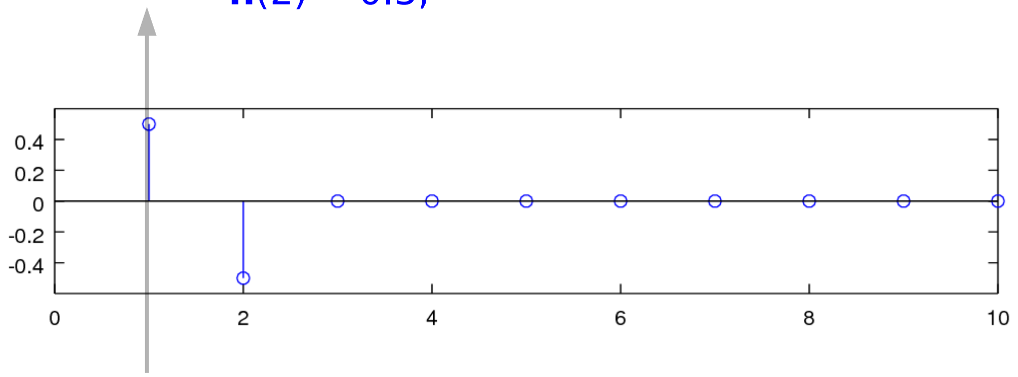


DSP for sound engineers (in Korean), J.W. Chae

FIR - High Pass filter

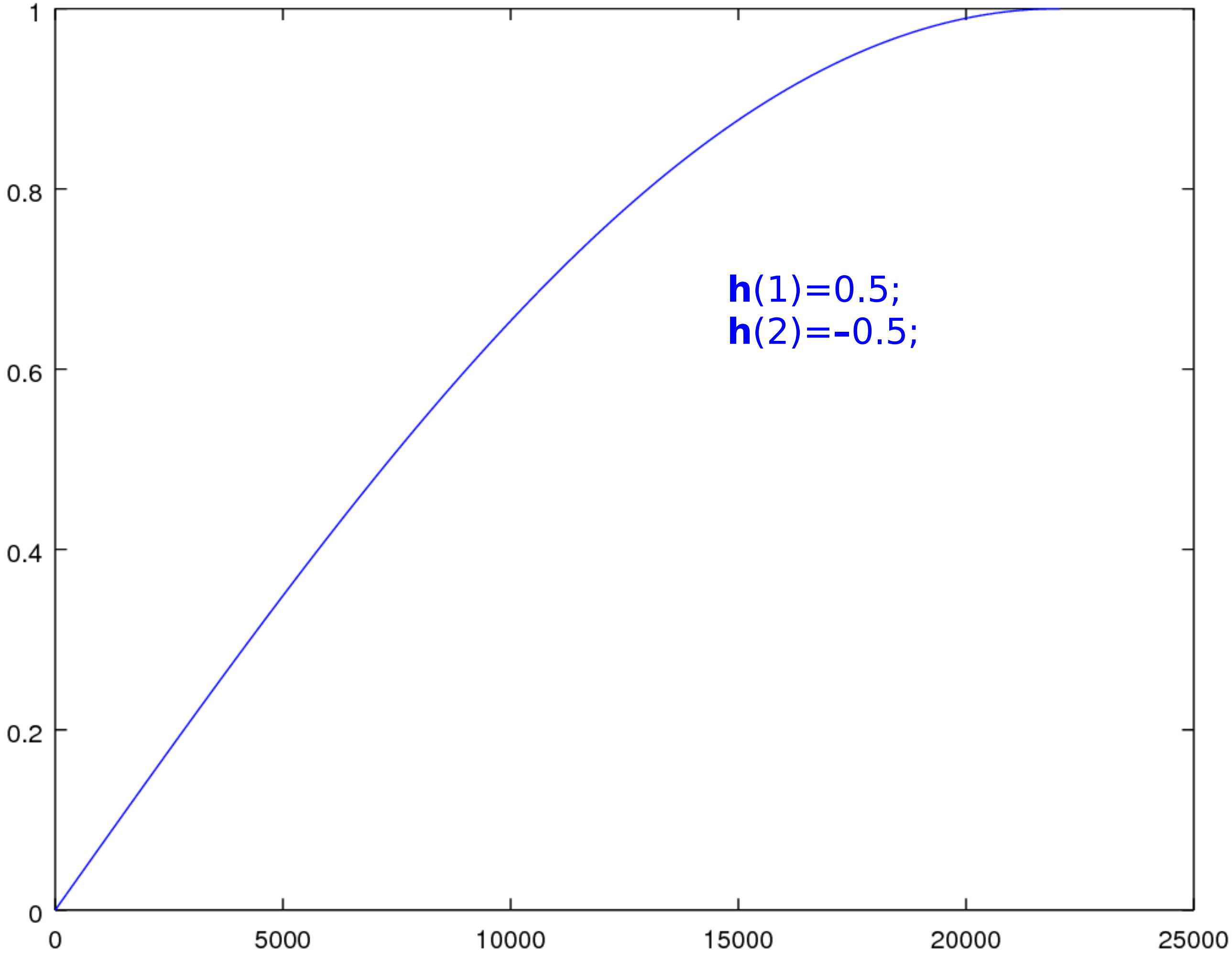
```
h=zeros(1, 44100);
h(1)=0.5;
h(2)=-0.5;
H=abs(fft(h));
H = H(1: 22050);
plot(H);
```

h(1)=0.5;
h(2)=**-**0.5;



DSP for sound engineers (in Korean), J.W. Chae

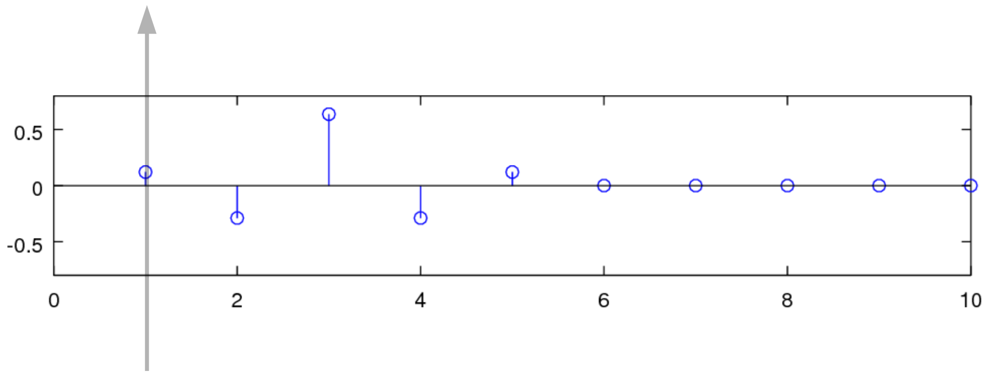
FIR - High Pass filter



DSP for sound engineers (in Korean), J.W. Chae

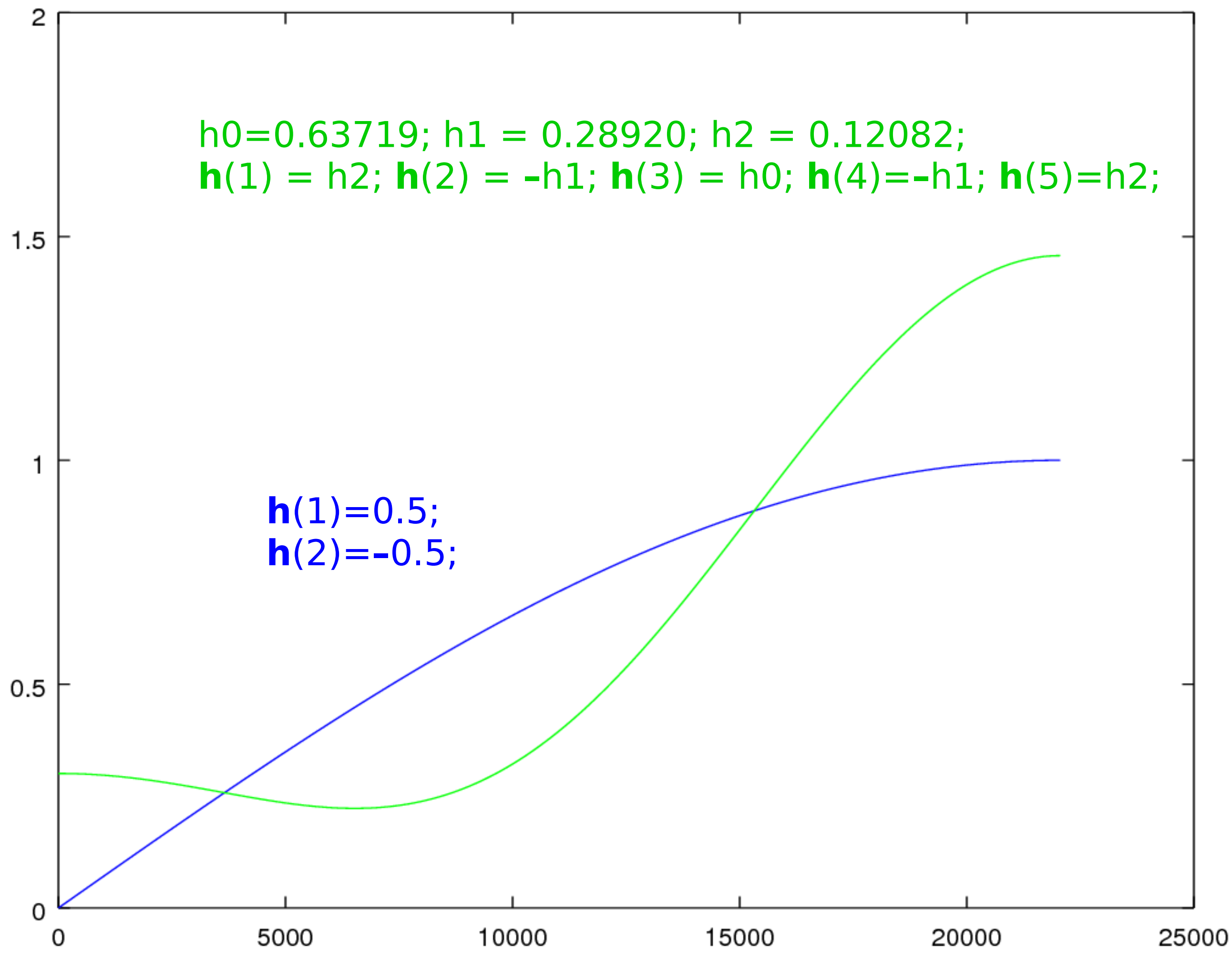
FIR - High Pass filter

```
h0=0.63719; h1 = 0.28920; h2 = 0.12082;
h = zeros(1, 44100);
h(1) = h2; h(2) = -h1; h(3) = h0; h(4)=-h1; h(5)=h2;
H = abs(fft(h));
H = H(1: 44100/2);
plot(H);
```



DSP for sound engineers (in Korean), J.W. Chae

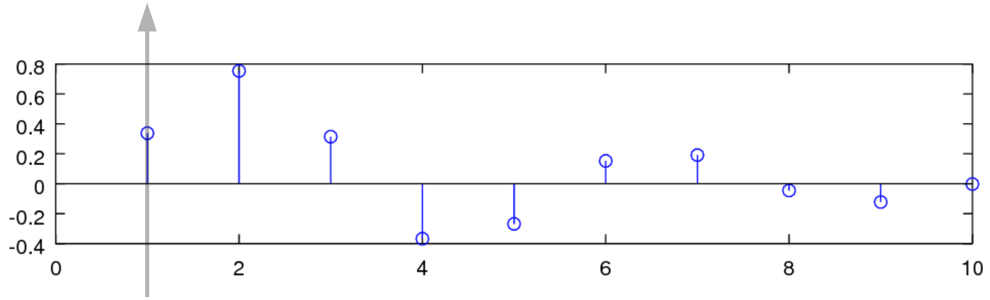
FIR - High Pass filter



DSP for sound engineers (in Korean), J.W. Chae

IIR - Low Pass filter

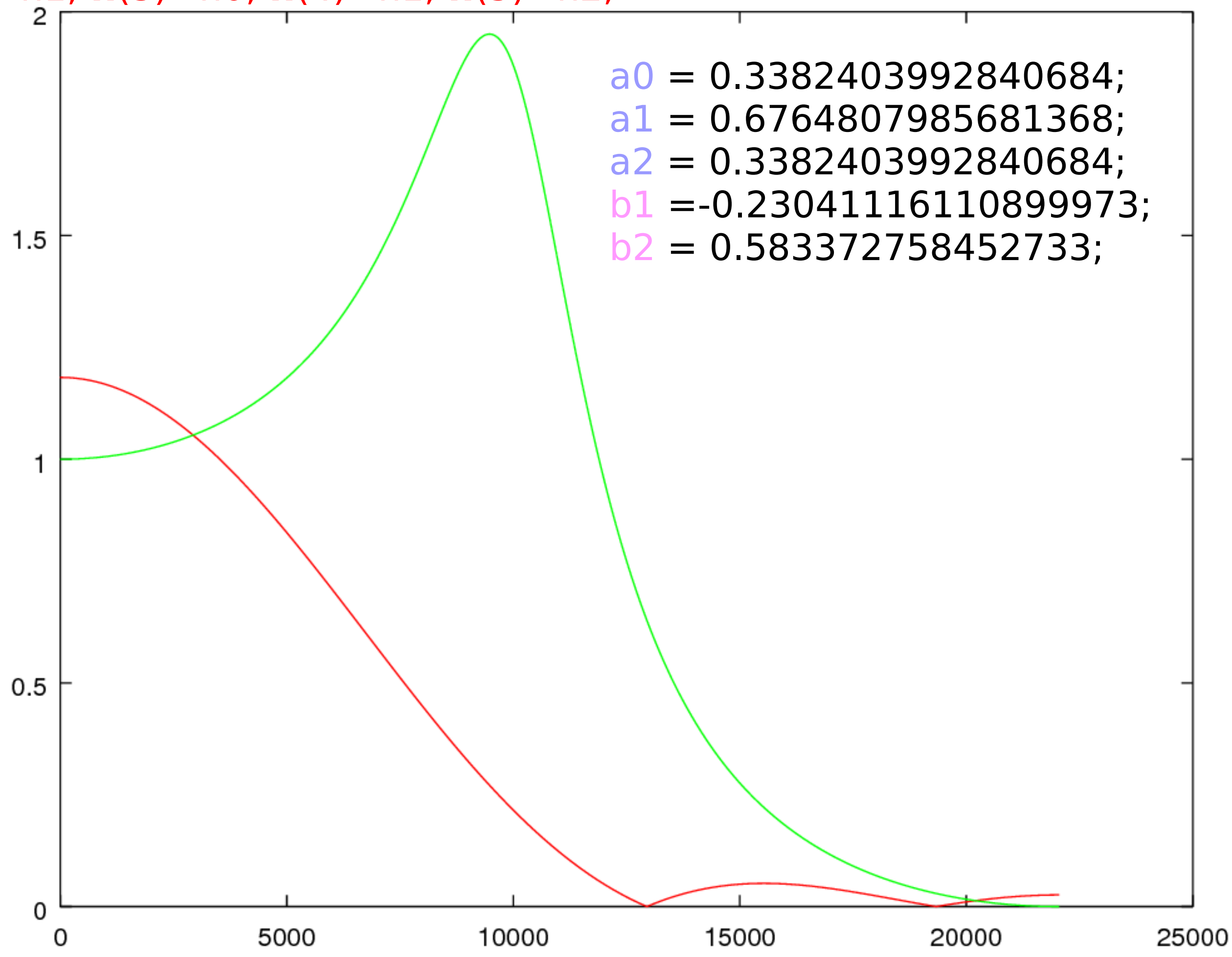
```
a0 = 0.338240392840684;
a1 = 0.676480785681368;
a2 = 0.338240392840684;
b1 =-0.230411611089973;
b2 = 0.58337275845273;

x= zeros(1, 44100);
x(1)=1;

y(1) = a0*x(1);
y(2) = a0*x(2) + a1*x(1) – b1*y(1);
for i=3:length(x)
  y(i) = a0*x(i) + a1*x(i-1) + a2*x(i-2) – b1*y(i-1) – b2*y(i-2);
endfor

Y= abs(fft(y));
Y= Y(1:length(y)/2);
plot(Y);
```



DSP for sound engineers (in Korean), J.W. Chae

IIR - Low Pass filter

h0=0.36281; h1= 0.28920; h2 = 0.12082;
h(1)=h2; **h**(2) = h1; **h**(3)=h0; **h**(4)=h1; **h**(5)=h2;



DSP for sound engineers (in Korean), J.W. Chae

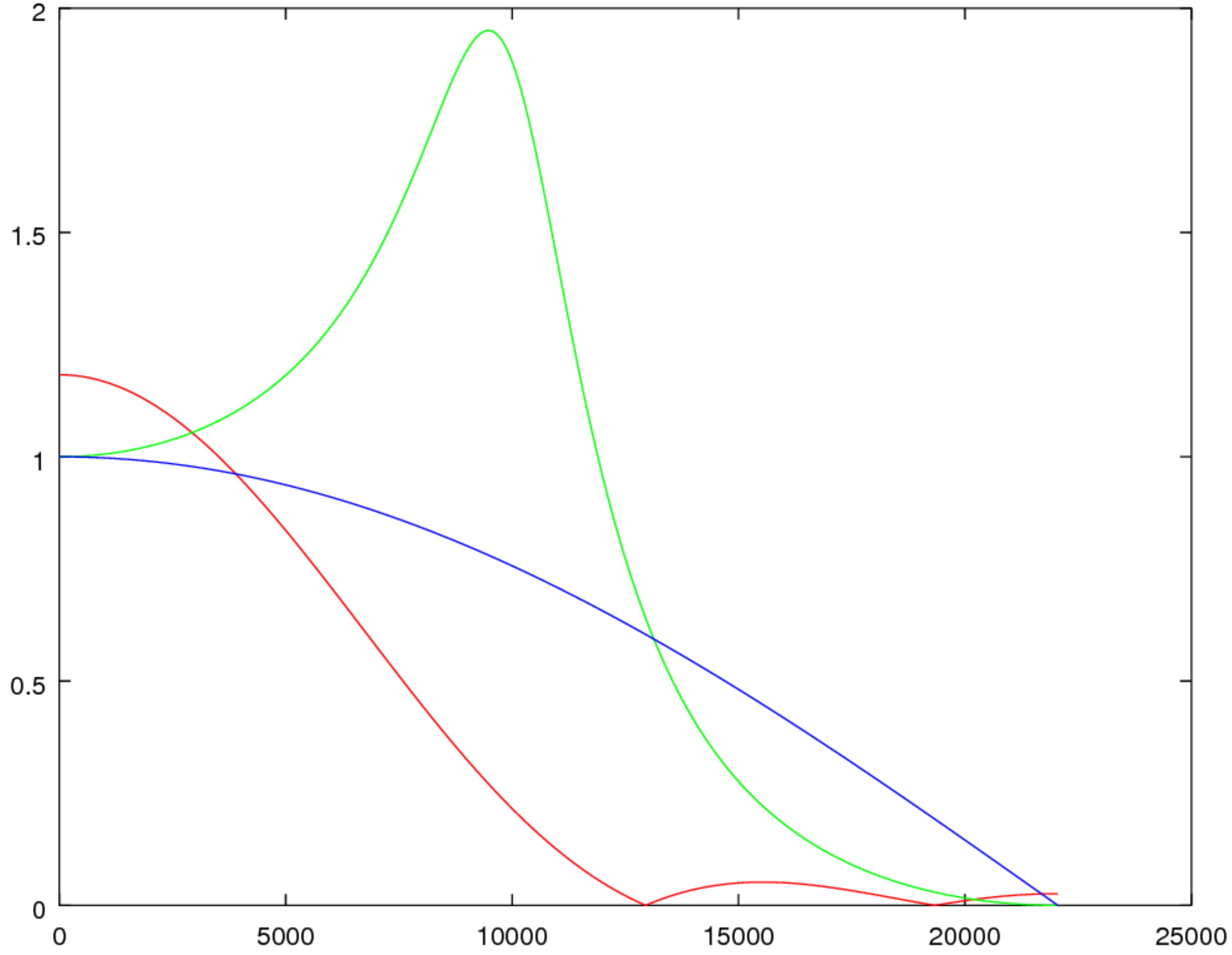
Moving Average Filter

```
a0 = 0.5;
a1 = 0.5;

x= zeros(1, 44100);
x(1)=1;

y(1) = a0*x(1);
for i=2:length(x)
  y(i) = a0*x(i) + a1*x(i-1);
endfor

Y= abs(fft(y));
Y= Y(1:length(y)/2);
plot(Y);
```

```
[y, fs]= waveread('t.wav');
y=y';
y2=[y 0];

for i=1:length(y)
  yy(i) = (y2(i) + y2(i+1)) / 2;
endfor

hold
plot(y, 'linewidth', 5);
plot(yy, 'm', 'linewidth', 2);
axis([100, 140, -1, 1])

sound(y, fs)
sound(yy, fs);
```

DSP for sound engineers (in Korean), J.W. Chae

Moving Average Filter



DSP for sound engineers (in Korean), J.W. Chae

Moving Average Filter - FFT

```
Y = abs(fft(y));
Y= Y(1:length(y)/2);
YY= abs(fft(yy));
YY= YY(1:length(yy)/2);

f=fs*(0:length(y)/2 -1)/length(y)

hold;
plot(f, Y, 'linewidth', 2);
plot(f, YY, 'y', 'linewidth', 0.5)
axis([3200, 15000, 0, 410]);
```

DSP for sound engineers (in Korean), J.W. Chae

Ideal Low Pass Filter

$$\frac{\sin(x)}{x} \qquad \frac{\sin(\pi x)}{\pi x}$$

$$H(f) = \operatorname{rect}\left(\frac{f}{2B}\right)$$

$$h(t) = \mathcal{F}^{-1}\{H(f)\} = 2B\frac{\sin(2\pi Bt)}{2\pi Bt} = 2B\,\operatorname{sinc}(2Bt)$$

$$h_{LPF}(t) = 2B_L\,\operatorname{sinc}(2B_L t)$$

$$H_{LPF}(f) = \operatorname{rect}\left(\frac{f}{2B_L}\right).$$

$$W_L = 2\pi\frac{8000}{44100} = 1.1.398$$

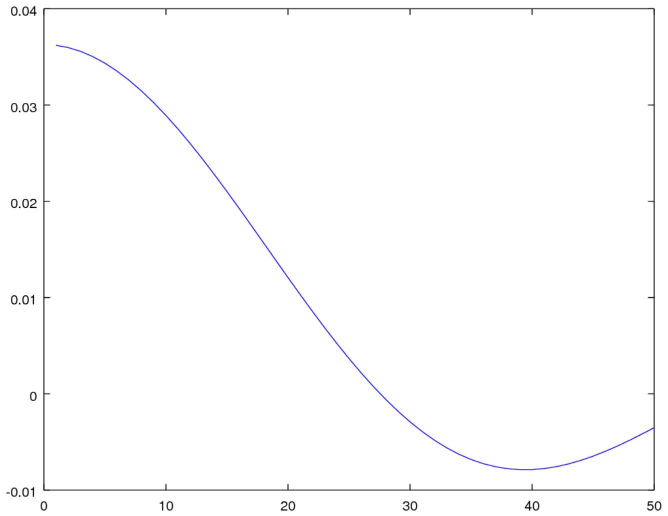
$$H_0 = \frac{W_L}{\pi} = 0.36281$$

$$H_m = H_{-m} = \frac{\sin(m\,W_L)}{m\pi}$$

https://en.wikipedia.org/wiki/Sinc_filter

Low Pass Filter - Manual

```
w = 2 * pi * (800/44100);
h0 = w / pi;
for i=1:50
  h(i) = sin(i*w) / (i*pi);
endfor
g=zeros(1, 44100);
g(51)=h0;

for i=1:101
  if (i>51)
     g(i) = h(i-51);
  elseif (i <51)
     g(i) = h(51-i);
  endif
endfor
```




```
G=abs(fft(g));
G=G(1:22050);
plot(G);
```

DSP for sound engineers (in Korean), J.W. Chae

Low Pass Filter

```
fc = input("input fc: ");
delay = input("input delay number : ");
dv=ceil(delay/2 +1);
w = 2*pi*(fc/44100);

for i=1:ceil(delay/2)
  h(i) = sin(i*w) / (i*pi);
endfor

h_lo=zeros(1, 44100);
h_lo(dv)= h0;
for d=1:(delay+1)
  if (d>dv)
    h_lo(d) = h(dv-d);
  endif
endfor
```

```
H_lo=abs(fft(lo));
H_lo=H_lo(1:22050);
plot(H_lo)
pritnf("save lpf");
disp("");
```

DSP for sound engineers (in Korean), J.W. Chae

High Pass Filter

```
fc = input("input fc: ");
delay = input("input delay number : ");
g = input("input gain : ");
w1 = 2*pi*(fc/44100);
w2 = w2 / pi;

for i=1:ceil(delay/2)
  h(i) = sin(i*w2) / (i*pi);
endfor

h_hi=zeros(1, 44100);
h_hi(dv)= h0;
for d=1:(delay+1)
  if (d>dv)
    h_hi(d) = h(dv-d);
  endif
endfor
```

```
for n=1:length(hi);
  if (mod(n,2) == 0)
    h_hi(n) = -1 * h_hi(n);
  endif
endfor

H_hi = abs(fft(h_hi));
H_hi = g*H_hi(1:22050);
plot(H_hi)
printf("save hpf")
disp("");
```

DSP for sound engineers (in Korean), J.W. Chae

HPF and LPF

```
Y = 4 * (H_hi * H_lo);
plot(Y);
axis([fc -1000, fc+1000, 0, g])
```

DSP for sound engineers (in Korean), J.W. Chae

Peak Filter

```
a0 = 1.0130557885385833;
a1 = -1.465715424005588;
a2=0.923649194879148;
b1=-1.465715424005588;
b2=0.9367049680264983;

x = zeros(1, 44100);
x(1)=1;
y(1)=a0*x(1);
y(2)=a0*x(2) + a*x(1) – b1*y(1);
for i=3:length(x)
  y(i)=a0*x(i) + a1*x(i-1) + a2*x(i-2)
        -b1*y(i-1) – b2*y(i-2);
endfor

Y = abs(fft(y));
Y = Y(1:length(y)/2);
plot(Y);
```

DSP for sound engineers (in Korean), J.W. Chae

One Pole Filter

```
a0=0.4344299454321238;
a1=0;
a2=0;b1=-0.56558005455678762;
B2=0;

x=zeros(1, 44100);
x(1)=1;
y(1)=a0*x(1);
y(2)=a0*x(2) + a1*x(1) – b1*y(1);
for i=3:length(x)
  y(i)=a0*x(i) + a1*x(i-1) + a2*x(i-2)
        - b1*y(i-1) – b2*y(i-2);
endfor

Y=abs(fft(y));
Y= Y(1:length(y)/2);
```

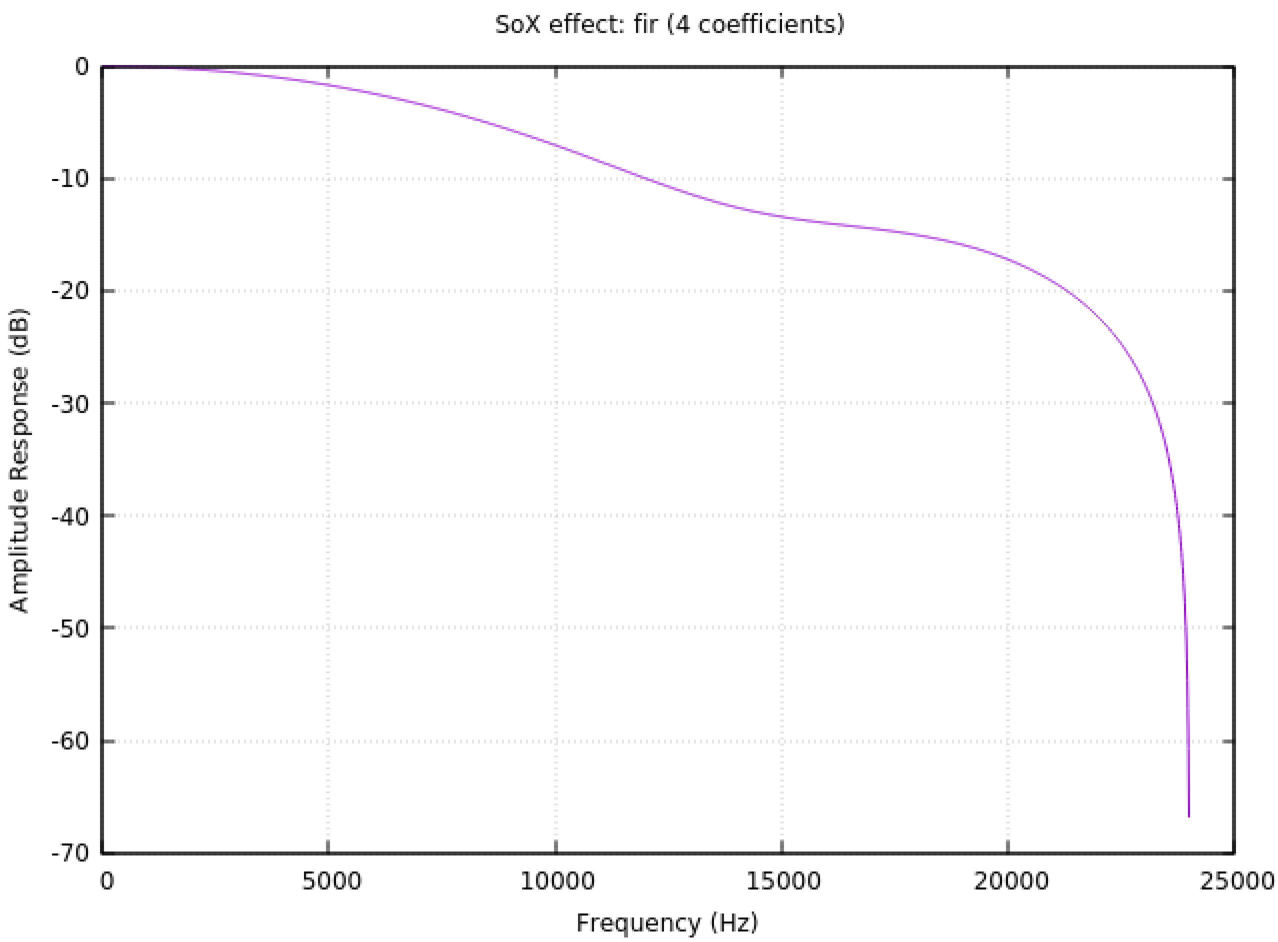
```
[xn, fs] = waveread('t.wav');
xn = xn';
XN=abs(fft(xn));
XN=XN(1:22050);
XN=XN/max(XN);

XF= XN .* Y;
hold
plot(XN, 'm');
plot(XF, 'g');
plot(Y, 'linewidth', 5);

y2=real(ifft(XF));

wavewrite(y2, 44100, 'noise.wav');
```

DSP for sound engineers (in Korean), J.W. Chae

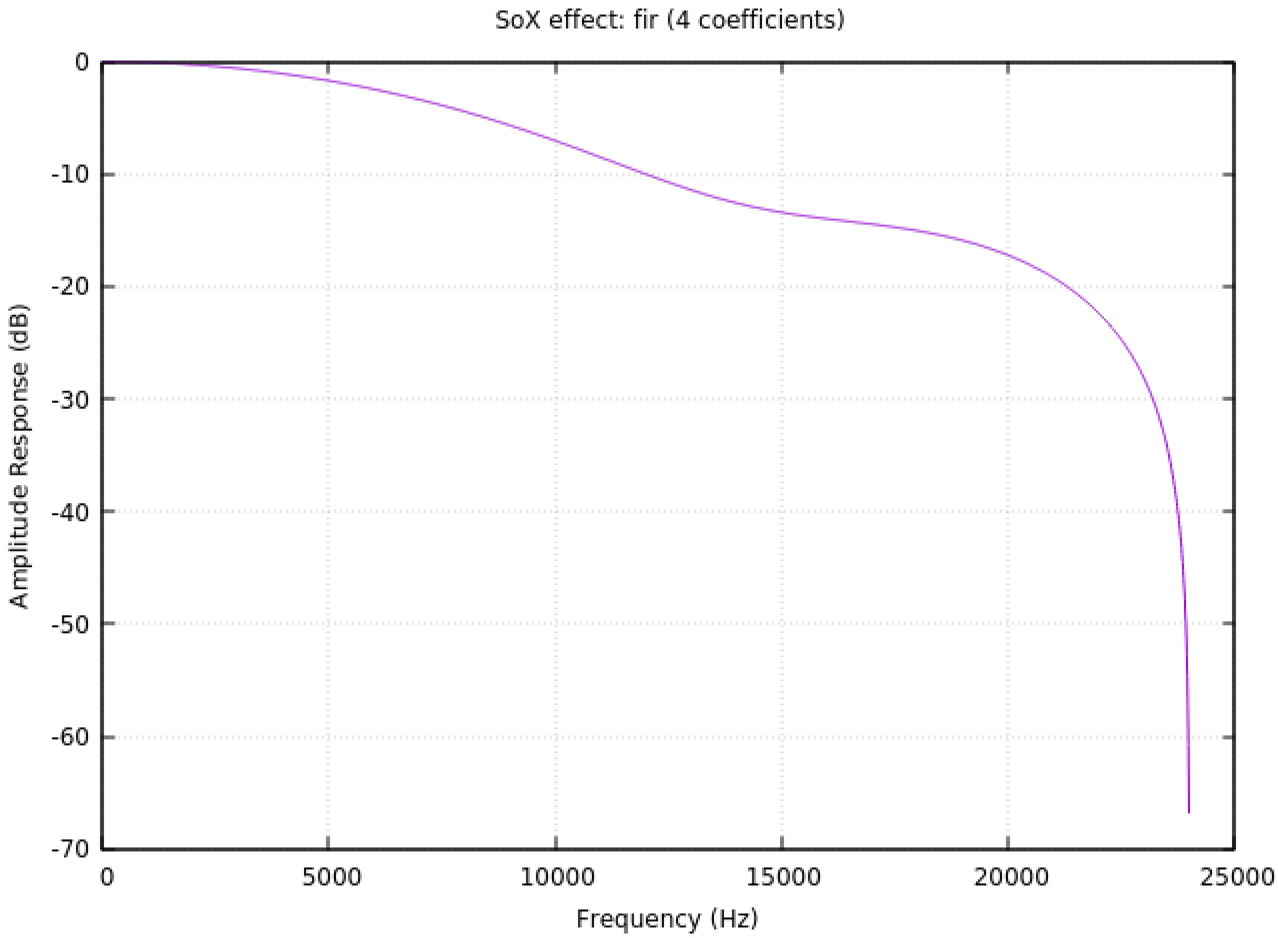
--plot gnuplot | octave

```
sox --plot gnuplot s6s.wav -n fir 0.1 0.2 0.4 0.3            >fir1.plt
sox --plot gnuplot s6s.wav -n fir coeff.txt                  >fir2.plt
sox --plot gnuplot s6s.wav -n biquad .6 .2 .4 1 -1.5 .6 >fir3.plt
sox --plot gnuplot s6s.wav -n fir 0.2 0.2 0.2 0.2 0.2    >fir4.plt
```

--plot gnuplot | octave



--plot gnuplot | octave



References

[1] F. Auger, Signal Processing with Free Software : Practical Experiments