FIR Filters

Finite Impulse Response (FIR) Filter Design:

Note: The relationship between the impulse response of a filter (f(t)) and its frequency response (F(s)) is defined by the LaPlace transform:

$$f(t) = \frac{1}{j2\pi}\int_{-j\infty}^{+j\infty} F(s)e^{st}\cdot ds$$

$$F(s) = \int_{-\infty}^{+\infty} f(t)e^{st}dt$$

This implies that

- **If two linear filters have identical impulse responses, the two filters will have the same frequency response.**

It really doesn't matter how you generate the impulse response of a filter. All that matters is the result.

For example, suppose you have the filter:

$$Y = \left(\frac{1}{z-0.9}\right)X$$

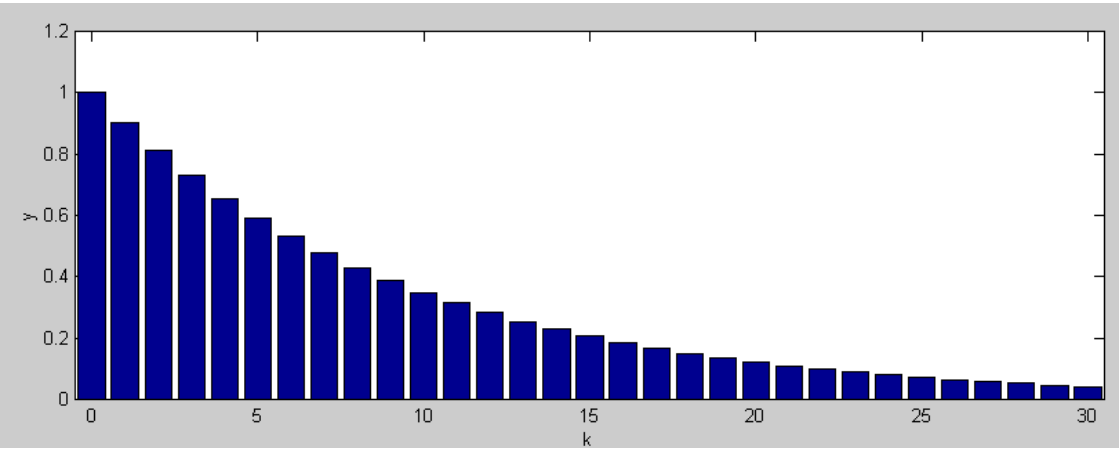
The series expansion of this filter is

$$Y = \left(1 + \frac{0.9}{z} + \frac{0.9^2}{z^2} + \frac{0.9^3}{z^3} + \ldots\right)X$$

or, in other words, the impulse response is

$$y(k) = (0.9)^k u(k)$$

which is shown below:



This suggests two methods to implement this filter. As a recursive filter, you would program

```
zY = 0.9*Y + X
 Y = zY
```

Alternatively, you could use the series expansion and add the previous inputs with weightings of 0.9^k:

```
for (i=100; i>0; i--) X[i] = X[i-1];     // remember the past 100 inputs
X[0] = a2d_read(0);

Y = 0;
for (i=0; i<=100; i++) Y += W[i] * X[i];   // W[i] = weighting = 0.9^i
```

Both methods are identical if you use an infinite number of terms in the series expansion. After a point, the weightings are small enough that you can truncate the series (at 100 terms in this example.) This results in a filter with a finite impulse response (101 samples after an impulse the output will be zero. You quit remembering the input past 100 samples in this case.)

Result: A Finite Impulse Response Filter

- Remembers the previous N values of the input (X),
- Combines these previous N inputs with weightings corresponding to the impulse response of the filter you want to implement,
- Thus generating your desired filter.

The neat thing about FIR filters is

- If you know the impulse response of the filter you want,
- You know how to implement this filter.

Causality:

One problem with FIR filters is many impulse responses are non-causal (the time response happens before t=0). For example,

Problem: Find the impulse response of an ideal low pass filter which passes frequencies between 0 and 1 rad/sec.

Solution: Apply the definition of LaPlace transforms:

$$F(s) = \begin{cases} 1 & |\omega| < 1 \\ 0 & otherwise \end{cases}$$

$$f(t) = \frac{1}{j2\pi} \int_{-j\infty}^{+j\infty} F(s) e^{st} \cdot ds$$

$$f(t) = \frac{1}{j2\pi}\int_{-j}^{+j} e^{st}\cdot ds$$

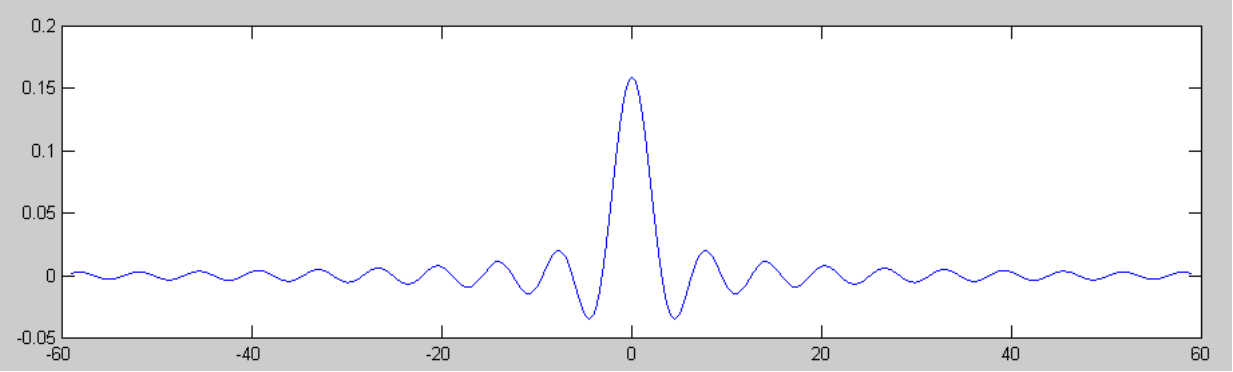
$$f(t) = \left(\frac{1}{j2\pi}\right)\left(\frac{1}{t}e^{st}\right)_{s=-j}^{s=+j}$$

$$f(t) = \left(\frac{1}{j2\pi}\right)\left(\frac{1}{t}\right)(e^{jt} - e^{-jt})$$

$$f(t) = \left(\frac{1}{2\pi}\right)\left(\frac{\sin(t)}{t}\right)$$

This is a sinc(t) function which is non-zero from $-\infty < t < \infty$.

- To build an ideal low pass filter, you need a filter which is non-causal
- If you could build such a filter, you can make lots of money in LasVegas. (Push a button whenever 10-black comes up in roulette. When you see the voltage start to bounce, you know 10-black is going to come up in a few seconds.)

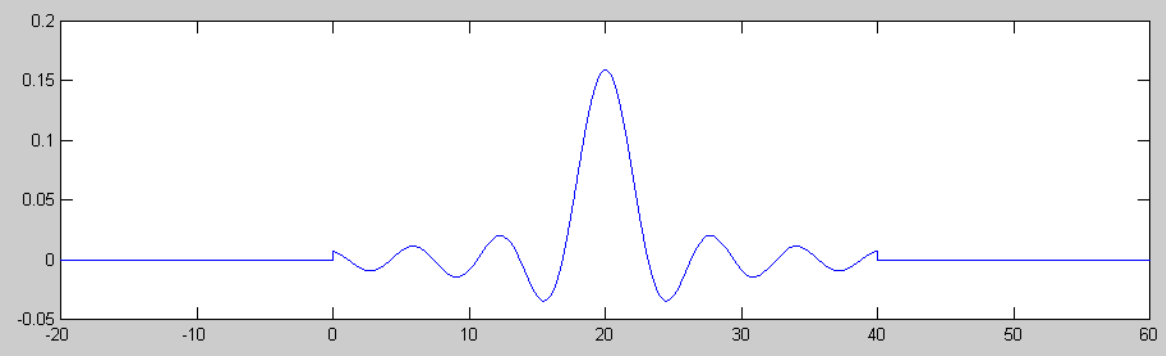


Impulse response of an ideal low-pas filter with a cut-of at 1 rad/sec

In order to build this filter, the following trick can be used:

- Approximate this filter by assuming the impulse response is zero beyond -20 and +20 seconds
- Delay the output by 20 seconds.

This results in the following impulse response:



Note that this impulse response is close to that of an ideal low-pass filter (meaning the frequency response should be close). Note also that it is different, meaning that this is not truly an ideal low pass filter.

You can see this by plotting the frequency response of this filter.

a) Assume a sampling rate of 10ms (meaning time is sampled every 0.01 second, or you remember the previous 4000 inputs over the past 40 seconds.

b) Assume the weightings on these 4000 inputs is as shown above.

The frequency response will be equal to

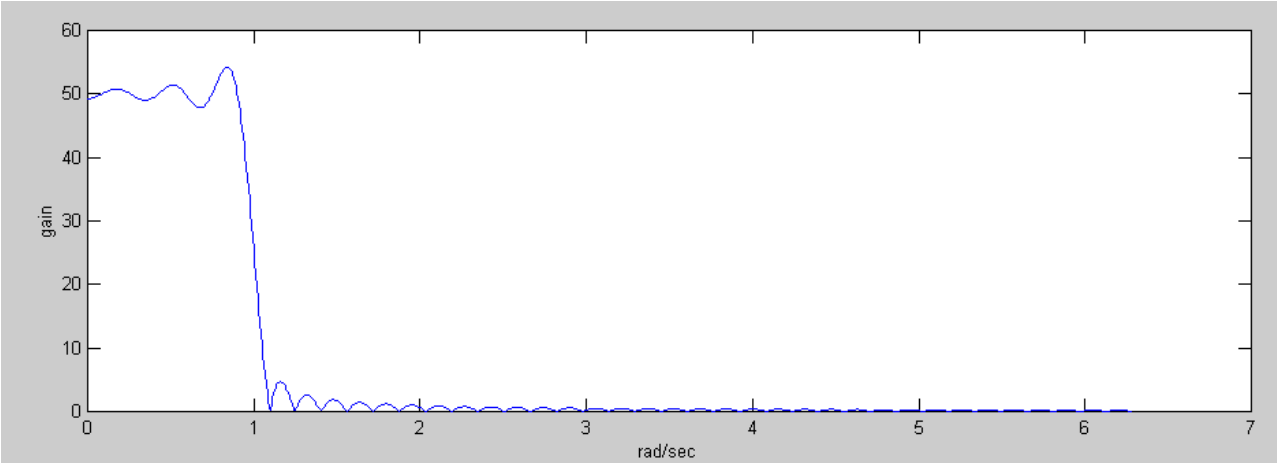
$$G(s) = \sum W(i) \cdot z^{-i}$$

where

$$z = e^{j\omega T}$$

MATLAB Code: (y1 = the impulse response shown above sampled every 0.01 second)

```
» f = [0:0.001:1]';
» s = j*2*pi*f;
» z = exp(s*0.01);
» G = 0*f;
» for i=1:4000
   G = G + y1(i) * (z .^ (4000-i));
   end
»
» plot(f*2*pi,abs(G))
» xlabel('rad/sec')
» ylabel('gain')
```



On the plus side:

- This is a very good low-pass filter, closely approximating an ideal low-pass filter.
- If you want a better filter, extend the tails of the impulse response
- This filter is very easy to implement: all you need is to know the impulse response of your filter

On the minus side:

- It involves remembering 4000 previous inputs (requiring more RAM than is available of a PIC).
- It involves 4000 floating point multiplies and 4000 floating point additions
- It would take a PIC about 0.8 seconds to compute y(k) each sample and you only have 0.01 second to do so.

In short, a FIR filter is not a good option for a PIC. A DSP (digital signal processor) is designed specifically for this type of filter.

Texas Instruments has a line of DSP processors:

The low end of this line (TMS320LF2401AVFA) costs $6.30 each

- 32-pin package
- 40MHz clock
- 5 x 10-bit A/D
- 1k RAM
- 16-bit add & multiply = one instruction

A mid-range (TMS320V) is a 144 LQFP package costs $19.80 each.

- 75MHz

- 136k RAM
- 40-bit floating point operations
- 8ns for a floating point operation (120MFLOP)