Digital Filters @ 2020

Class Assignment: Remix

https://sampleswap.org/filebrowser-new.php?d=VOCALS+and+SPOKEN+WORD%2FDonald+Trump+Construction+Kit%2F

Class Assignment: Remix

1.Download 4 speech samples

2.Filter it by adding a delayed (500 ms) attenuated (gain < 0.5) version

3. Extract the bass drum or drum-kick using a low-pass filter (or mirfilterbank) from another track

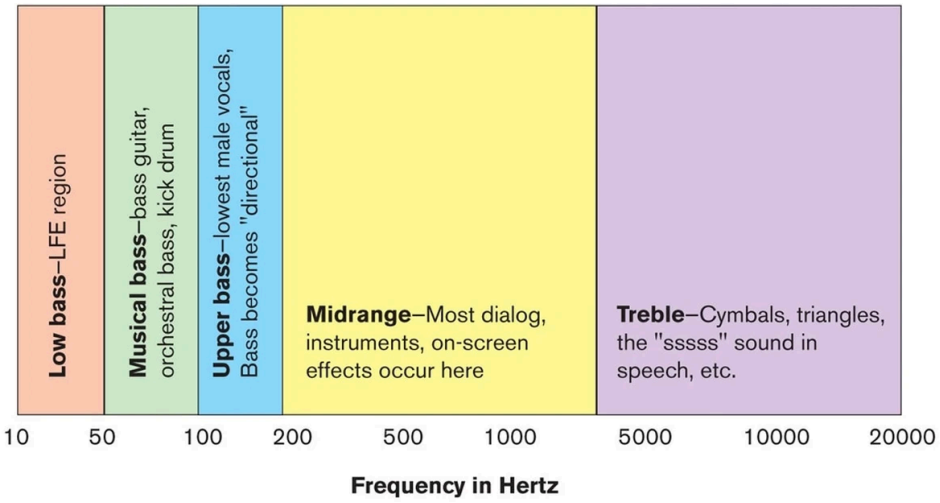
4. Combine 3 with 2 and loop it

5.Create a pitch-shifted version

Digital Filters

Digital Filters

- Computational algorithm to modify a digital signal
  - *attenuate (zero out)* certain frequencies
  - *amplify (boost)* certain frequencies



Digital Filters

- Digital Filters commonly used in sound processing
    - Low-pass, High-Pass, Band Pass filters
    - Peak & notch filters
    - Comb filters
    - Shelving filters & equalizers

Digital Filters

- Lowpass Filter
- Highpass Filter
- Bandpass Filter
- Bandstop Filter

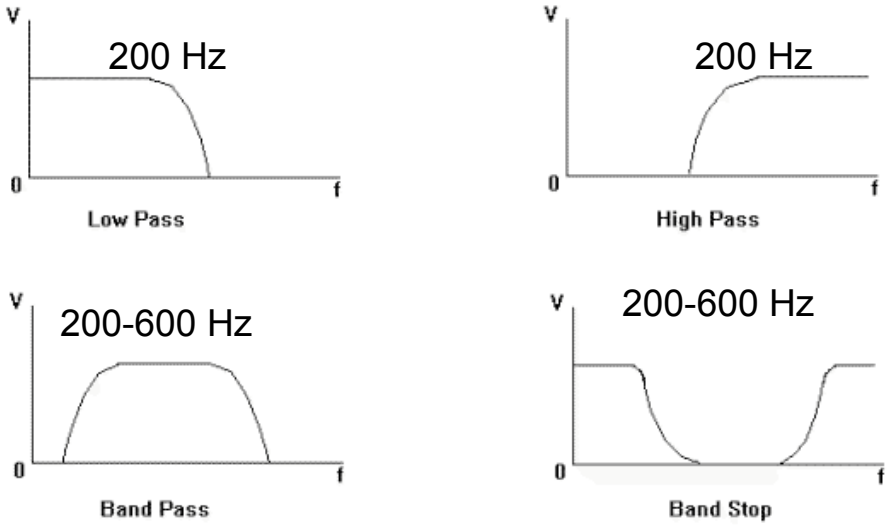
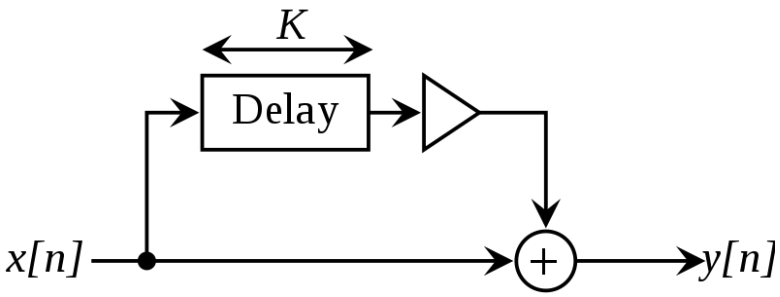
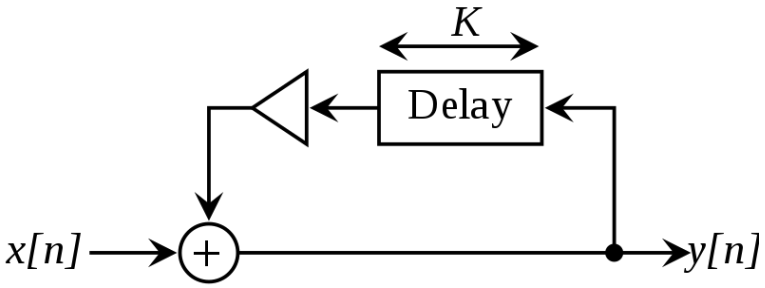


Fig. 3

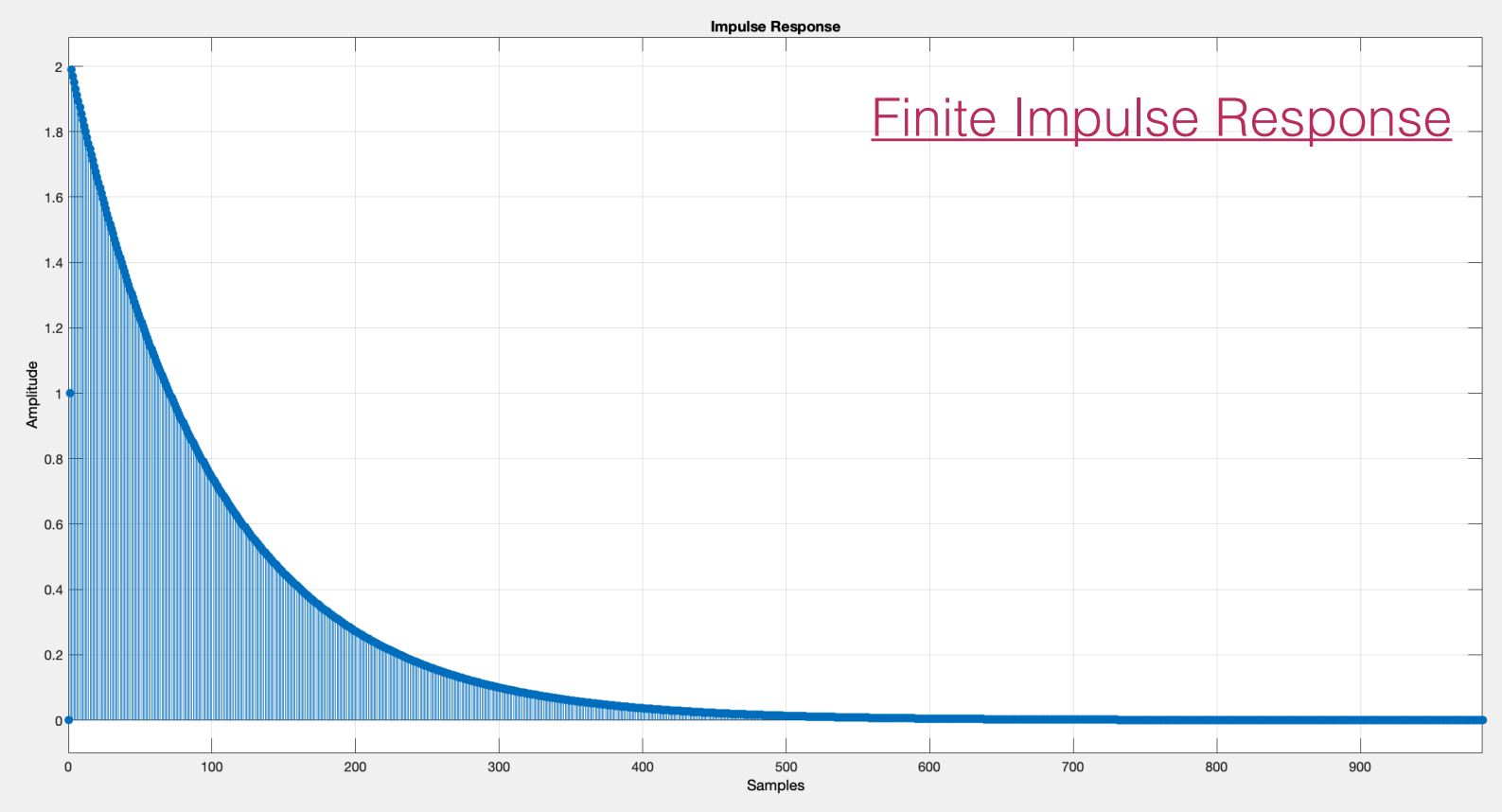
Digital Filters

- **2 main types**
  - Feedforward or Finite Impulse Response filters
  - Feedback or Infinite Impulse Response filters



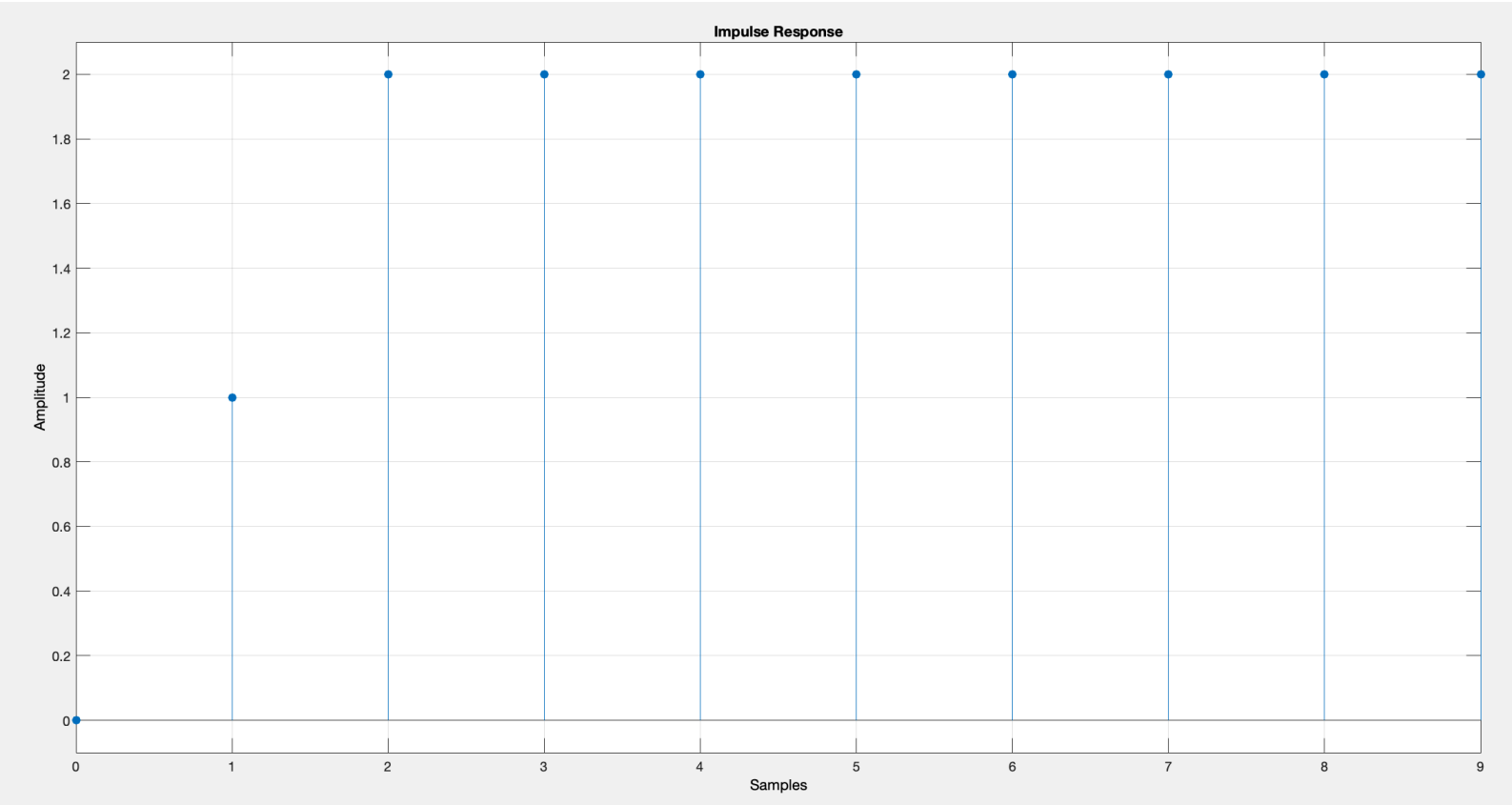


Impulse Response



Impulse Response

Infinite Impulse Response



Digital Filters

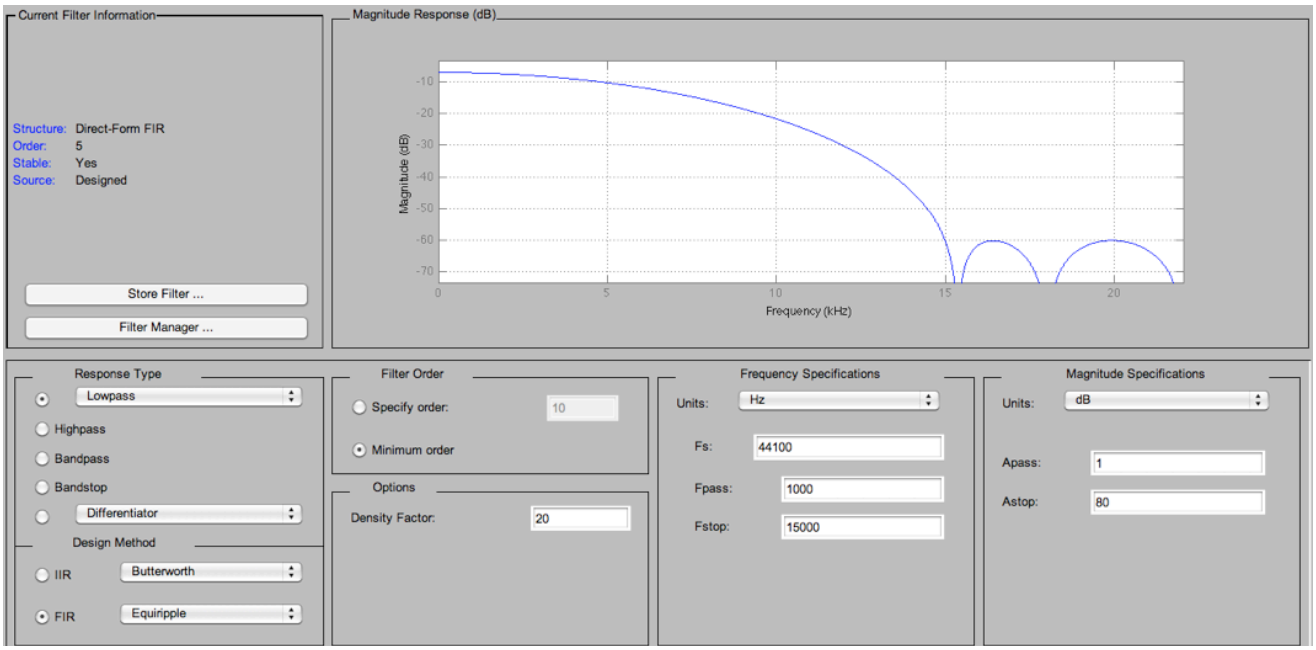
- Described by a *difference equation*
  - *difference equations: relationship between consecutive values of a sequence*
- output is a sum of previous inputs &/or outputs
  - $y(n) = x(n)$
  - $y(n) = x(n-1)$
  - $y(n) = 1/3\ [x(n-2) + x(n-1) + x(n)]$
  - $y(n) = 1/2\ [x(n-2) + x(n-1)] + y(n-1)$ **<— feedback**

Q1: Filter Design

- determining coefficients

fdatool

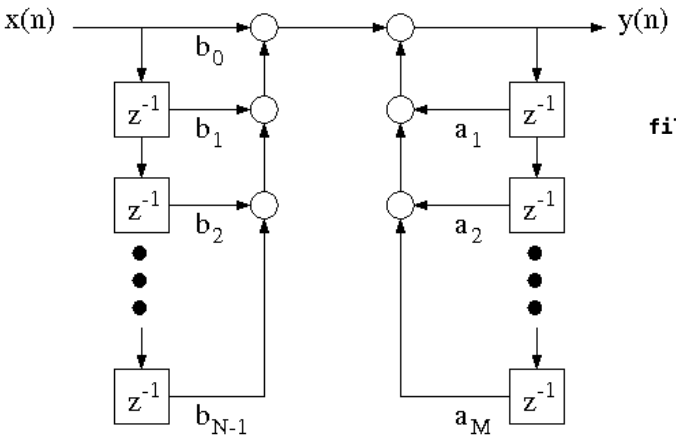
  - desired frequency response



Filter Implementation

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$

Filter coefficients - a_k , b_k




```
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.  The filter is a "Direct Form II Transposed"
   implementation of the standard difference equation:

   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)
```

caution: a_k<— feedback

FIR Filters

- **Finite Impulse Response**
  - Finite-Duration impulse response
  - Output = weighted linear combination of current and limited no. of past inputs

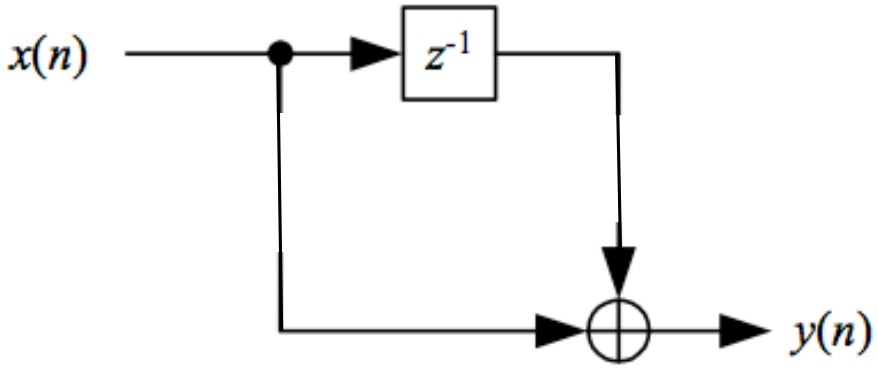
$$y[n] = \sum_{i=0}^{N} b_i x[n-i]$$

- ex: First order FIR filters
  - $y(n) = x(n) + x(n\text{-}1)$
  - $y(n) = x(n) - x(n\text{-}1)$

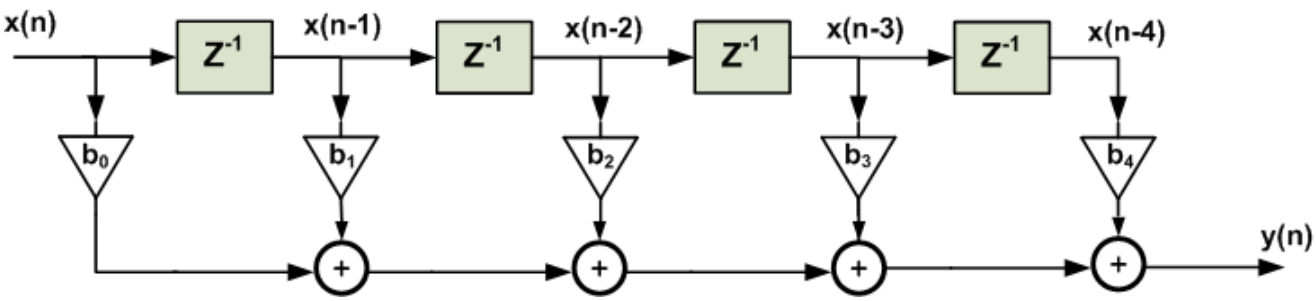
Example

- $y(n) = x(n-1)$

- $y(n) = x(n) + x(n-1)$



Example



$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) + b_3x(n-3) + b_4x(n-4)$$

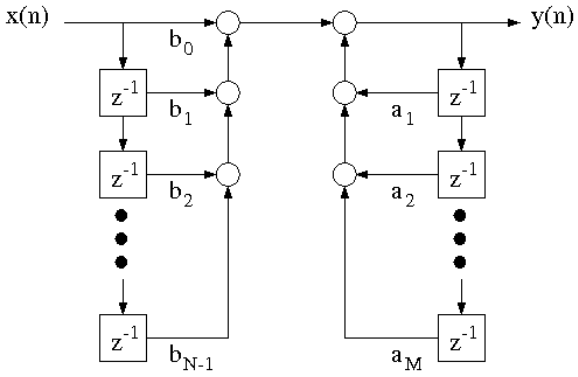
```
filter One-dimensional digital filter.
   Y = filter(B,A,X) filters the data in vector X with the
   filter described by vectors A and B to create the filtered
   data Y.  The filter is a "Direct Form II Transposed"
   implementation of the standard difference equation:

   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)
```

Q2: Feed-forward Echo effect

IIR Filters

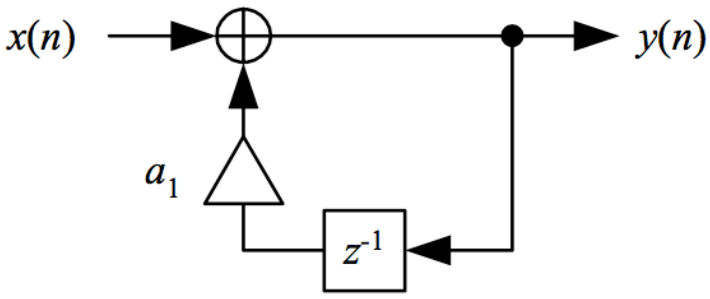
- IIR = Infinite Impulse Response
  - Impulse-response length is infinite in theory
  - Impulse-response can decay to zero
- IIR filter structures are based on *feedback*

$$y[n] = \sum_{i=0}^{N} b_i x[n-i] + \sum_{i=0}^{M} a_i y[n-i]$$



IIR Filters

- ex: First Order IIR Filter
  - $y(n) = x(n) + a_1 y(n-1)$
  - $Y(z) = X(z) + a_1 z^{-1} Y(z)$
  - $Y(z)/X(z) = 1/(1-a_1 z^{-1})$
  - $Y(z)/X(z) = z/(z-a_1)$



"Leaky integrator"
(when $0 < a_1 < 1$)

FIR Vs. IIR

- more stable
- simple to implement
- high memory
- only zeros present

- can be/less stable
- lower order filters hence reduced computational complexity
- less memory
- zeros and poles present

Digital Filters

- frequency response can be changed by changing filter coefficients

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$

- but….
  - what do time-domain coefficients have to do with frequency domain filter response?
  - how do we obtain the coefficients?

Filter Design

- determining coefficients
  - carried out in the **frequency** domain
    - desired frequency response
    - Z - domain representation
    - obtain filter coefficients

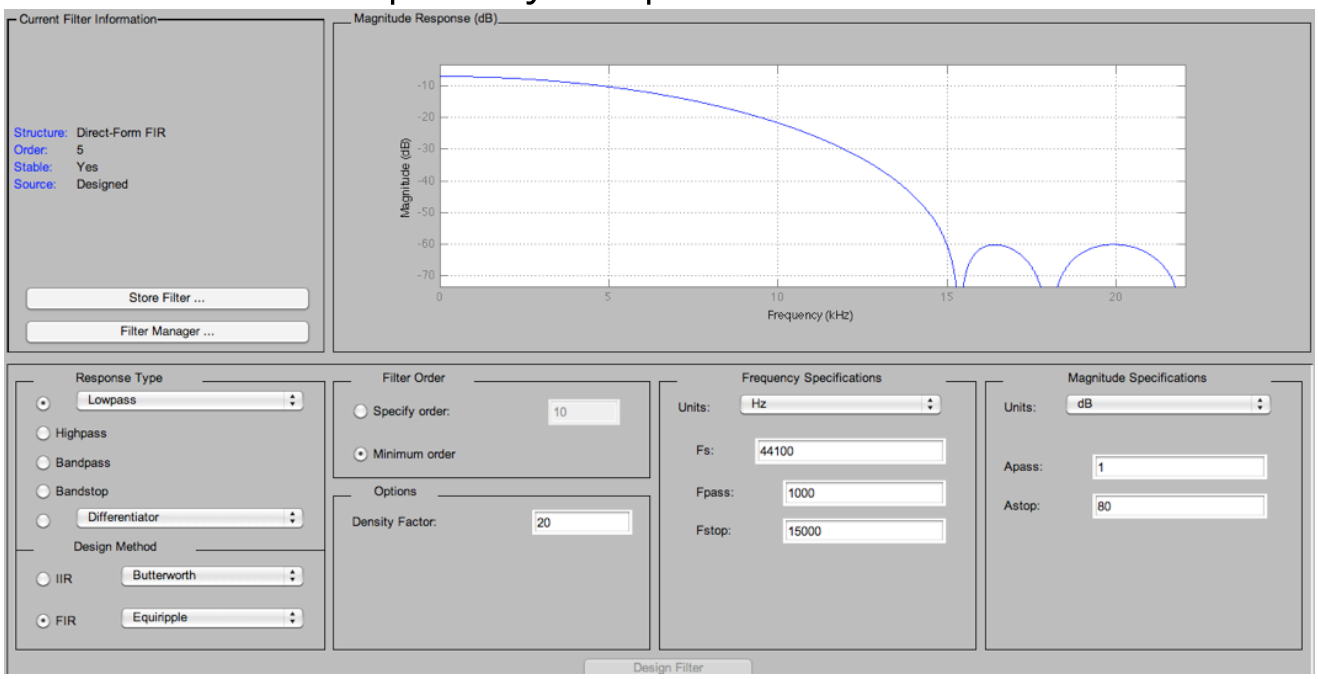
$$y[n] = \sum_{l=1}^{N} a_l y[n-l] + \sum_{k=0}^{M} b_k x[n-k]$$

Filter Design

- determining coefficients

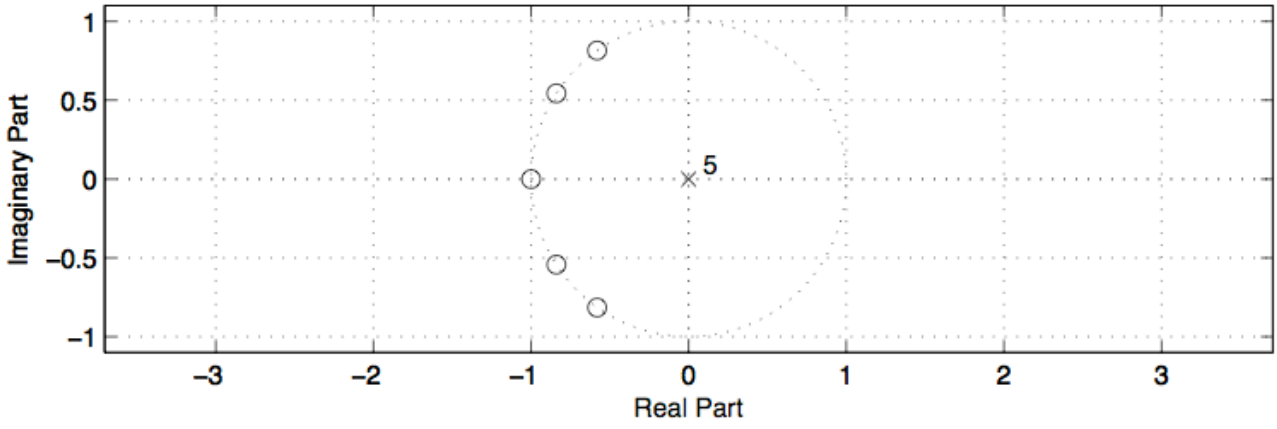
fdatool

  - desired frequency response



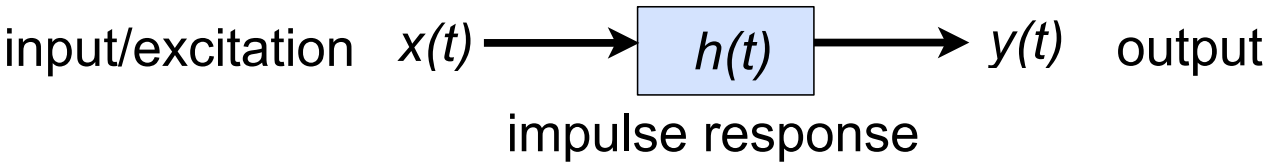
Filter Design

- determining coefficients
  - desired frequency response
  - Z - domain representation



System Basics

- System
  - device/algorithm that performs an operation on a signal
    - eg: filters

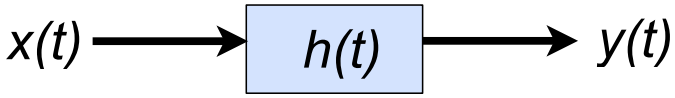


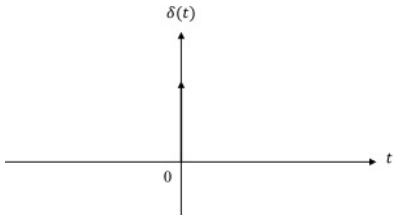
impulse response

$$x(t) * h(t) = y(t)$$

Impulse Response

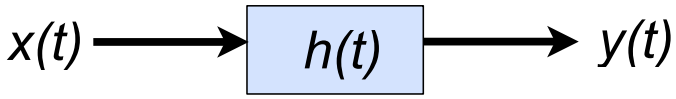
- Impulse Response = response of the system to a unit impulse
- Unit impulse = Digital signal whose first sample is 1 and the rest are 0
- describes a filter in the time domain





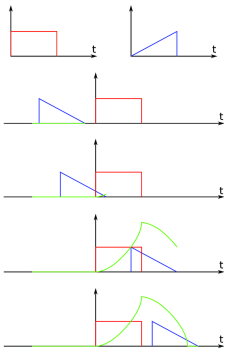
System Basics

- * in time-domain = x in frequency-domain



$$x(t) * h(t) = y(t)$$

$$X(z) \times H(z) = Y(z)$$



Transfer Function

- defined as the z-transform of the impulse response *h(t)*

$$Y(z) = H(z)X(z)$$

$$H(z) = \frac{Y(z)}{X(z)}$$

- Transfer function $H(z)$
  - Consider the system shown in Figure 4.

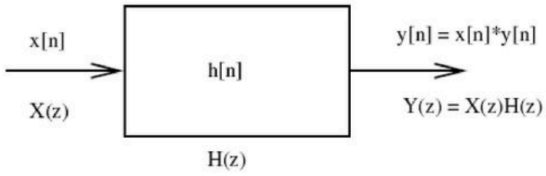


Figure 4: signal - system representation

  - $x[n]$ is the input and $y[n]$ is the output
  - h[n] is the impulse response of the system. Mathematically, this signal-system interaction can be represented as follows

$$y[n] = x[n] * h[n]$$

  - In frequency domain this relation can be written as

$$Y(z) = X(z).H(z)$$

or

$$H(z) = \frac{Y(z)}{X(z)}$$

$H(z)$ is called 'Transfer function' of the given system.
In the time domain if $x[n] = \delta[n]$ then $y[n] = h[n]$,
$h[n]$ is called the 'impulse response' of the system.
Hence, we can say that

$$h[n] \longleftrightarrow H(z)$$

Z-Transform

- converts a discrete-time signal, which is a sequence of real or complex numbers, into a complex frequency-domain representation
- converts the difference equations in time domain into the algebraic equations in z-domain (*k* represents sample number)

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

Z-Transform

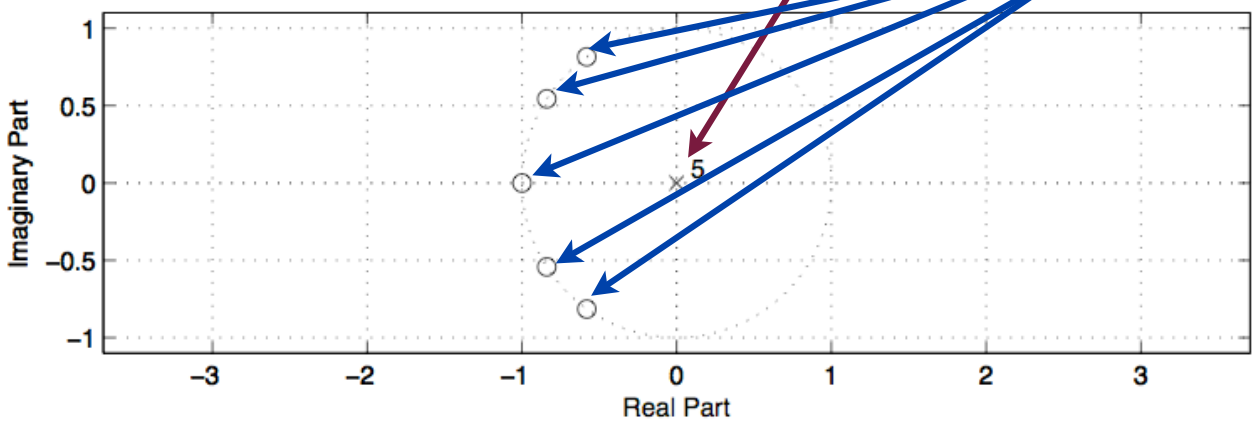
- $x(n) = [1\ 2\ 5\ 7\ 0\ 1]$
- $X(z) = 1 + 2z^{-1} + 5z^{-2} + 7z^{-3} + z^{-5}$

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

Z-Domain

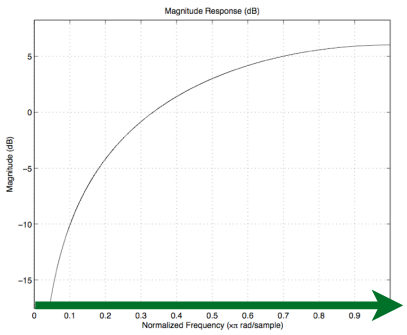
- Complex plane
- Useful in designing, analyzing and predicting system characteristics
- Poles & Zeros

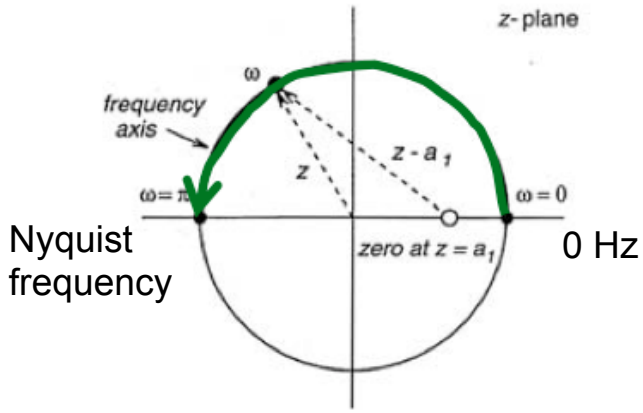
$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$



Poles & Zeros

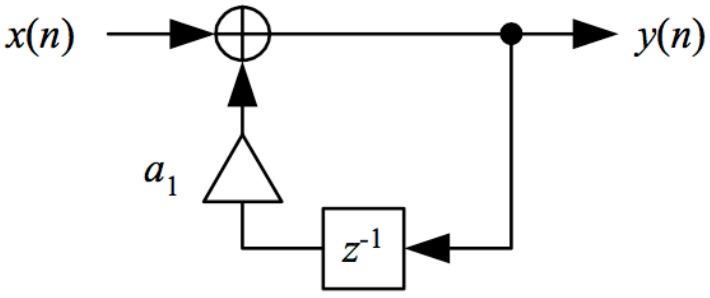
- Pole-Zero plot = a pole–zero plot is a graphical representation transfer function
- Attenuate = Zeros
- Amplify = Poles





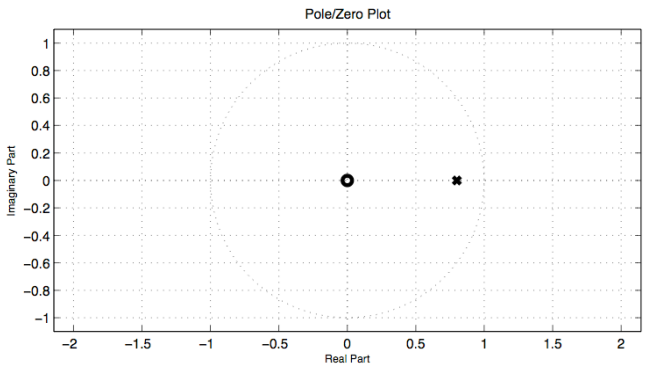
Example

- ex: **PZ plot** of First Order IIR Filter
  - $y(n) = x(n) + a_1 y(n-1)$
  - $Y(z)/X(z) = 1/(1-a_1 z^{-1})$



"Leaky integrator"

(when $0 < a_1 < 1$)



Example

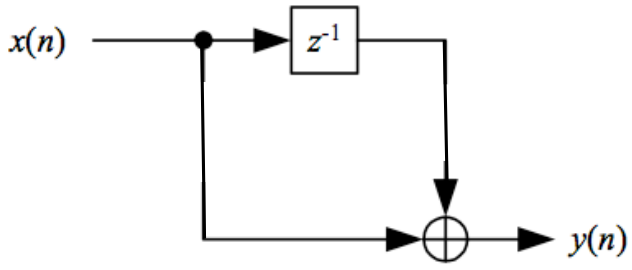
- First Order FIR Filter

$y(n) = x(n) + x(n-1)$

$Y(z) = X(z) + X(z)(z^{-1})$

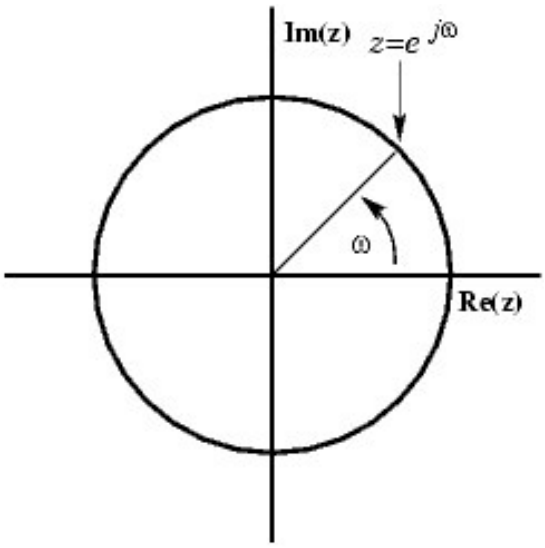
$Y(z)/X(z) = (1+z^{-1})$

$H(z) = (1+z^{-1})$



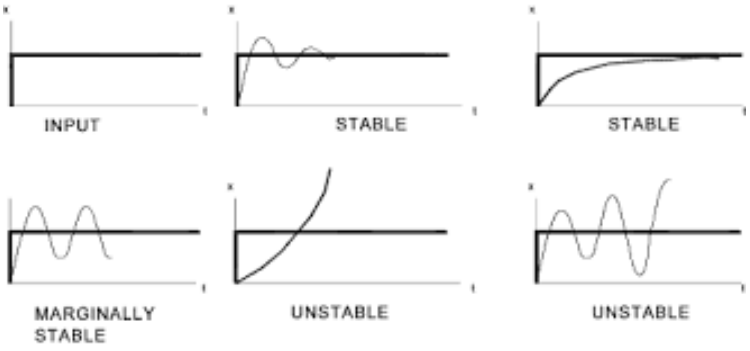
Z-Transform

- important in identifying system stability
- determines frequency response of a system
- complex number $z = re^{jw}$
- *w = angular* frequency



Poles and Zeros

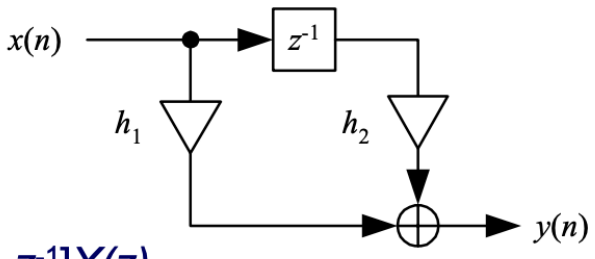
- Poles and Zeros determine how the system acts
  - what it does to the input
- Poles determines stability of the system



Example

First-order FIR Filter

- Add the input signal with its delayed and scaled version
  - Real weights h_1 and h_2 for the input and the delayed input signal
- Difference equation:
  $y(n) = h_1 x(n) + h_2 x(n-1)$
- Impulse response: $[h_1\ h_2]$
- Z transform:
  $Y(z) = h_1 X(z) + h_2 X(z) z^{-1} = [h_1 + h_2 z^{-1}] X(z)$
- Transfer function: $H(z) = h_1 + h_2 z^{-1}$
- Frequency response (set $z = e^{j\omega}$)
  $H(e^{j\omega}) = Y(e^{j\omega}) / X(e^{j\omega}) = h_1 + h_2 e^{-j\omega}$

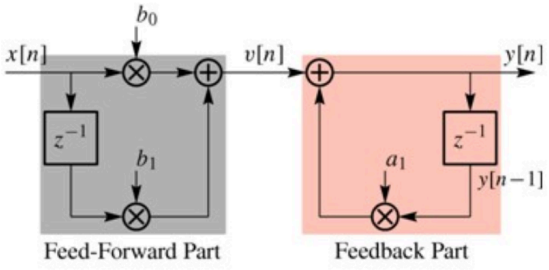


Filter Implementation

$$y[n] = \sum_{l=1}^{N} a_l y[n-l] + \sum_{k=0}^{M} b_k x[n-k]$$

feedback term — FIR part

recursive filter



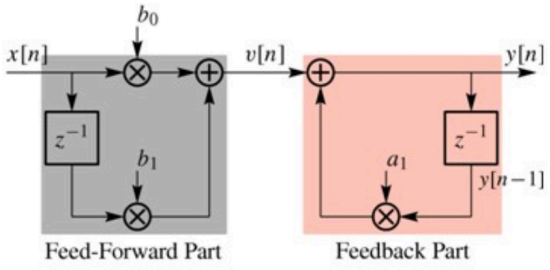
$$y[n] = a_1 y[n-1] + b_0 x[n] + b_1 x[n-1]$$

The first order case : $N = M = 1$,

```
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.  The filter is a "Direct Form II Transposed"
   implementation of the standard difference equation:

   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)
```

B = [*b0 b1*] A = [*1 -a1*]

Ex: Filter Implementation



$$y[n] = a_1 y[n-1] + b_0 x[n] + b_1 x[n-1]$$

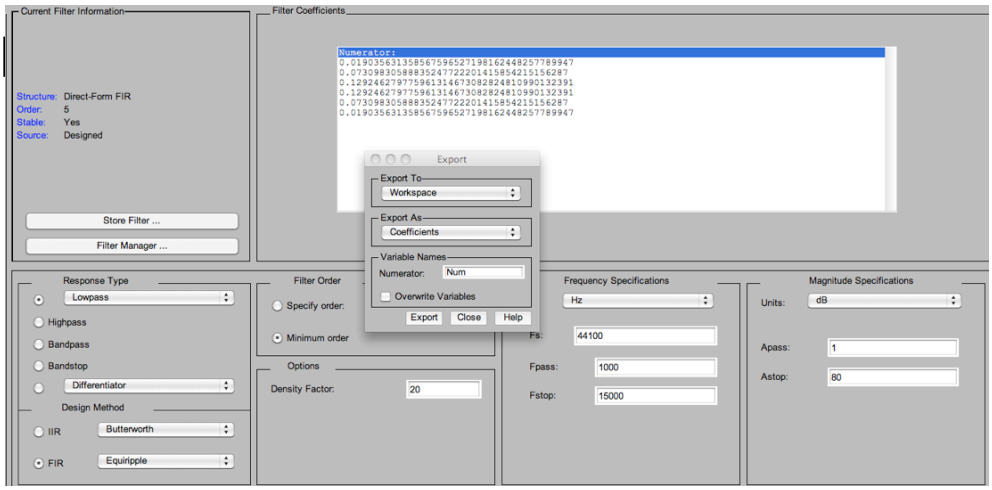
$$\rightarrow \quad Y(z) = a_1 z^{-1} Y(z) + b_0 X(z) + b_1 z^{-1} X(z)$$

$$\rightarrow \quad (1 - a_1 z^{-1}) Y(z) = (b_0 + b_1 z^{-1}) X(z)$$

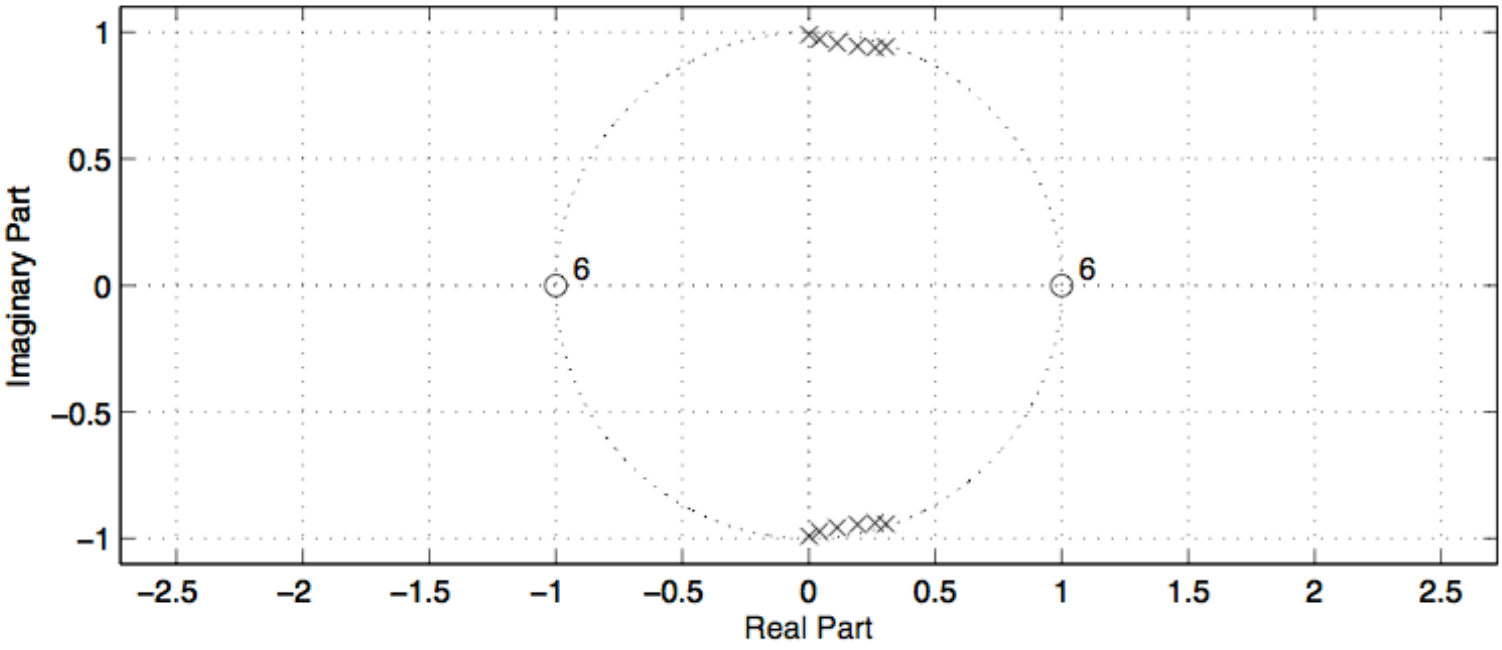
$$\rightarrow \quad H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 - a_1 z^{-1}} \equiv \frac{B(z)}{A(z)} \quad \begin{matrix} \rightarrow \text{ FIR part} \\ \rightarrow \text{ feedback part} \end{matrix}$$

Filter Design

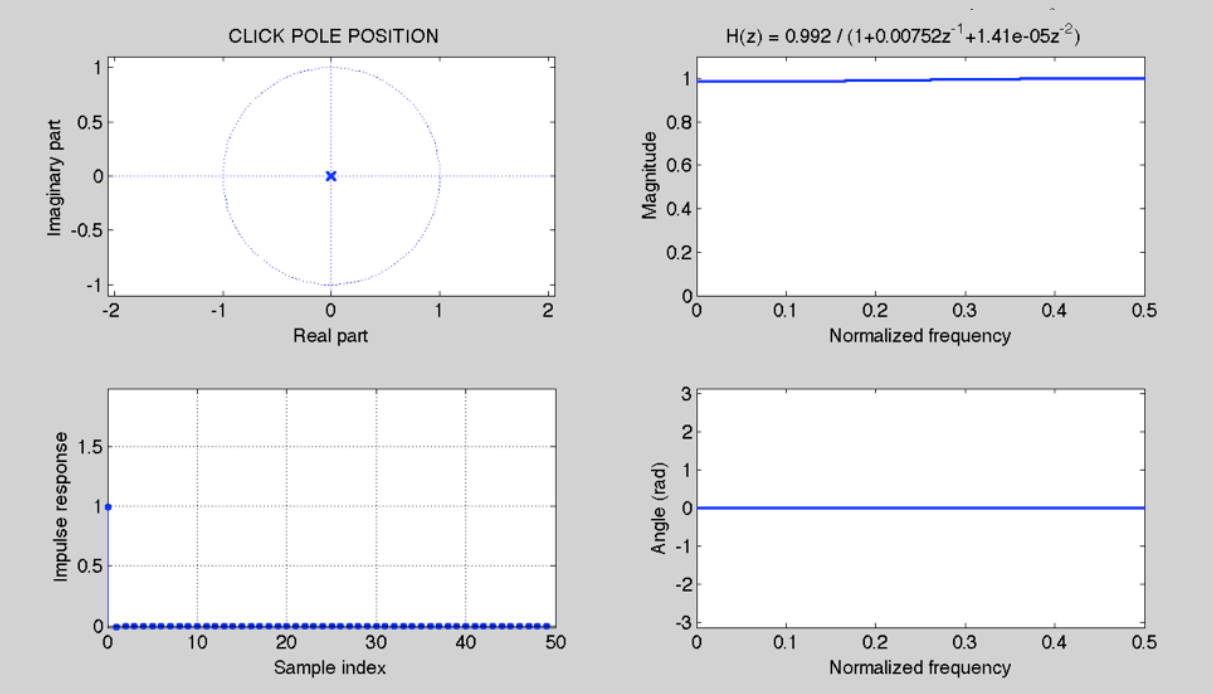
- determining coefficients
  - desired frequency response
  - Z - domain representation
  - o



Examples



Examples

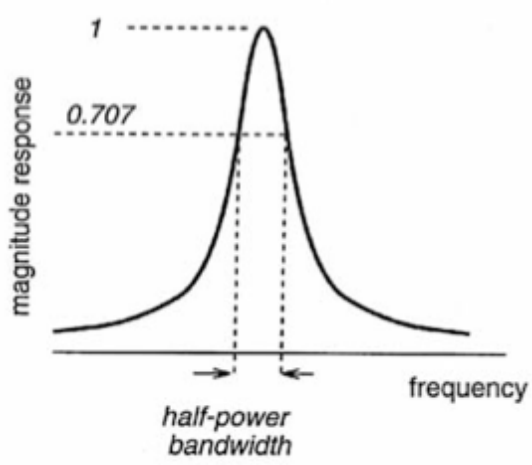

CLICK POLE POSITION
Imaginary part
Real part
H(z) = 0.992 / (1+0.00752z^-1+1.41e-05z^-2)
Magnitude
Normalized frequency
Impulse response
Sample index
Angle (rad)
Normalized frequency

Resonant Filter

- A bump in the magnitude response
- Properties:
  - Sharpness described by *bandwidth (B)*
    - *Difference of frequency points where the magnitude response is 3 dB below the peak*
  - *Center frequency* & *peak value (gain)*
  - *Q* factor (Quality factor)

    = center frequency/bandwidth

    - Large Q indicates sharp resonance

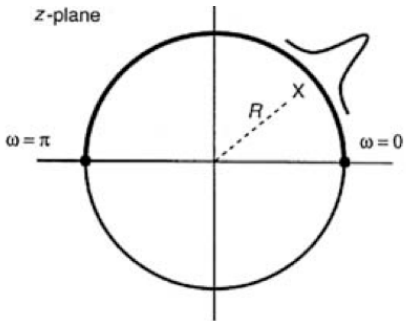


Resonant Filter

- Resonance is caused by a pole close to the unit circle
- Distance can be deduced from bandwidth:
    - $R \approx 1 - B/2$ where $B = 2\pi f$ and f is normalized frequency
- Example:

    If fs = 44.1 kHz and bandwidth = 20 Hz,

    $R \approx 1 - \pi(20/44100) = 0.998575$



Resonant Filter

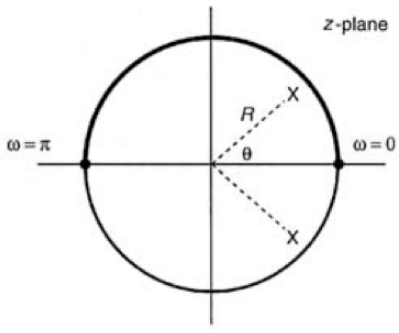
- Resonators are important in audio DSP
  - Parametric equalizers, subtractive synthesis, formant synthesis, ...
- Difference equation

$$y(n) = A_0 x(n) + a_1 y(n\text{-}1) - a_2 y(n\text{-}2)$$

$$A_0 = (1 + R^2)\sin(\theta)$$

$$a_1 = 2R\cos(\theta);\ a_2 = R^2;\ R = 1 - B/2$$

θ = normalized (by fs) frequency

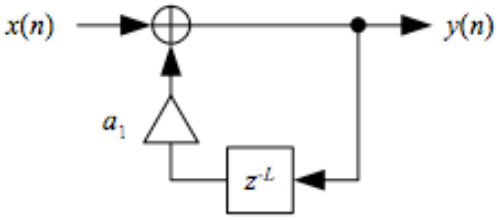


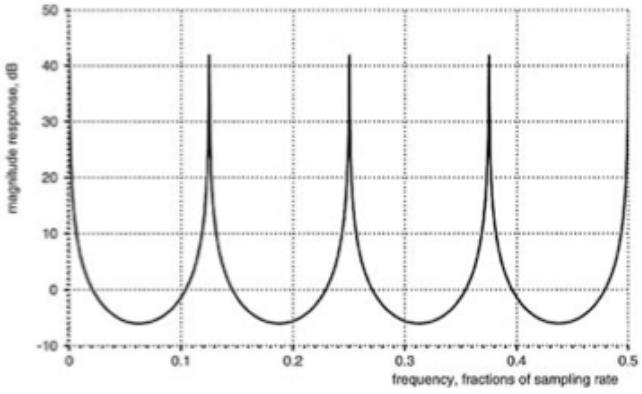
How to design a digital resonator

- Choose bandwidth *B (ex:50 hz)* and peak frequency f_c
- Calculate desired pole radius $R = 1 - B/2 = 1-((2\pi*50/44100)/2)$
- Calculate cos(θ):
  - $\cos(\theta) =(2R/(1+R^2))\cos(2*pi* f_c/f_s)$
- Calculate gain factor A_0 that gives a max gain of 0 dB:
  - $A_0 = (1 + R^2)\sin(2*pi* f_c/f_s)$
- Implement the filter:
  - $y(n) = A_0x(n) + 2R\cos(\theta)y(n-1) - R^2y(n-2)$

Comb Filter

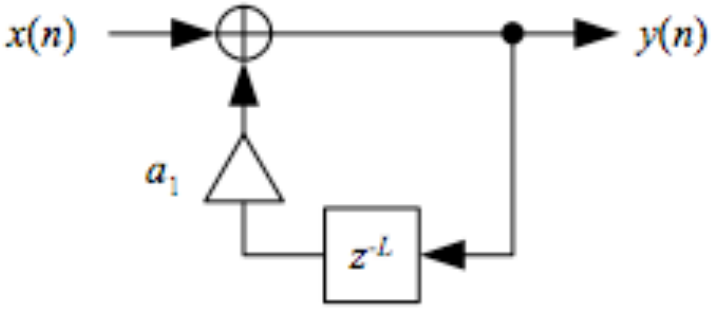
- The name stems from the shape of the magnitude response
- Useful for sound synthesis
- a comb filter adds a delayed version of a signal to itself

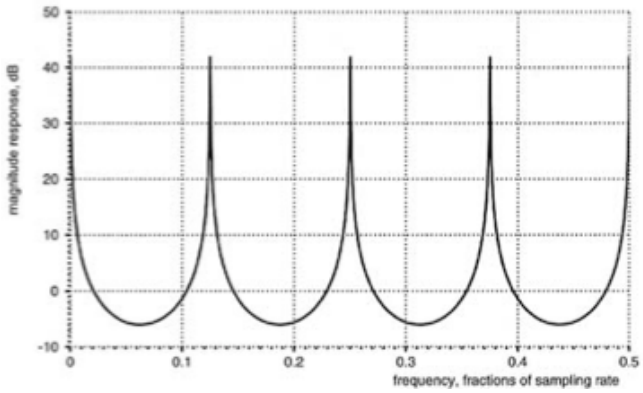




Comb Filter

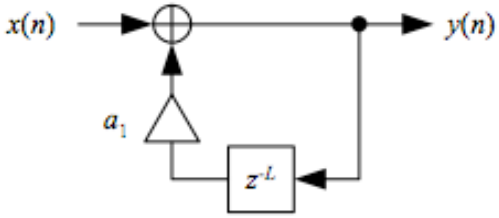
- Obtained from a one-pole filter by replacing z^{-1} with z^{-L}
- Number of spikes = $L/2$



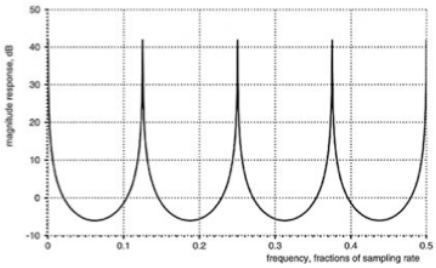


Comb Filter

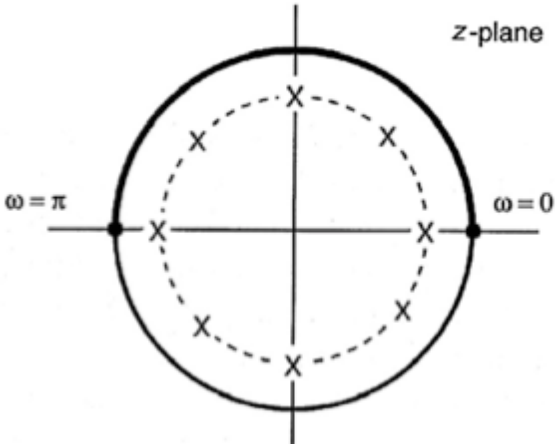
Time domain



Frequency domain



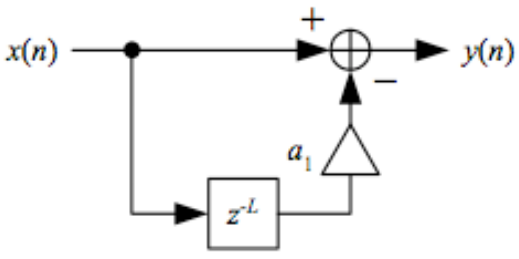
Z domain

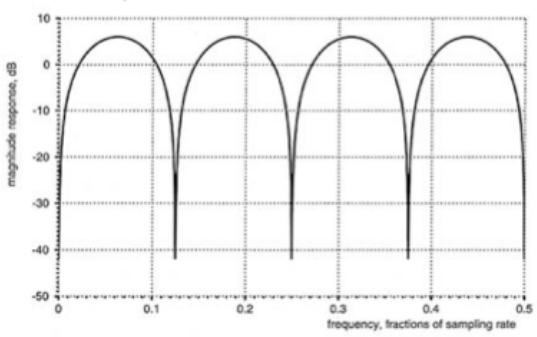


ex : L = 8 unit delays

Inverse Comb Filte

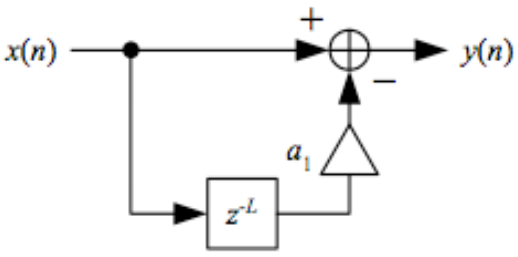
- FIR filter
- Obtained by inverting the comb filter transfer function
  - Feedback path becomes feedforward path
  - Poles become zeros



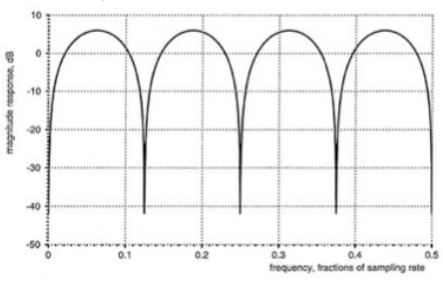


Inverse Comb Filte

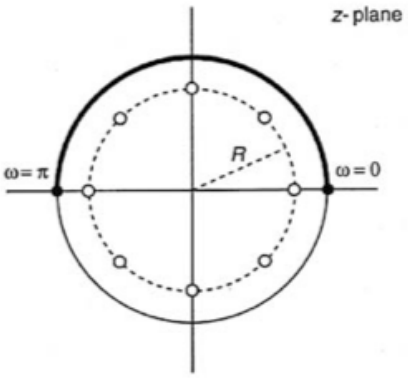
Time domain



Frequency domain



Z domain



ex : L = 8 unit delays