Contents

1 Week 6 Module 4 Part 2: Introduction to Filtering

- First strategy of filter design: Imitation
  - uuImpulse truncation

– Window Method
– Frequency Sampling

Trying to replicate the structure of either the impulse response or the frequency response of ideal filters.

1.1 Filter Design Part 1 (FIR Filter)

- An ideal filter is not realizable in practice because the impulse response is a two-sided infinite support sequence.

1.1.1 Reference

- The Scientist and Engineers Guide to DSP: Recurscive Filter

1.1.2 Impulse truncation

Impulse Truncation

1. Pick ω_c
2. Compute ideal impulse response h[n] (analytically)
3. truncate h[n] to a finite-support $\hat{h}[n]$
4. $\hat{h}[n]$ defines an FIR filter

FIR approximation of lenght M = 2N+1

$$\hat{h}[n] = \begin{cases} \frac{\omega_c}{\pi}\, sinc(\frac{\omega_c}{\pi} n) & |n| \leq N \\ 0 & \text{otherwise} \end{cases}$$

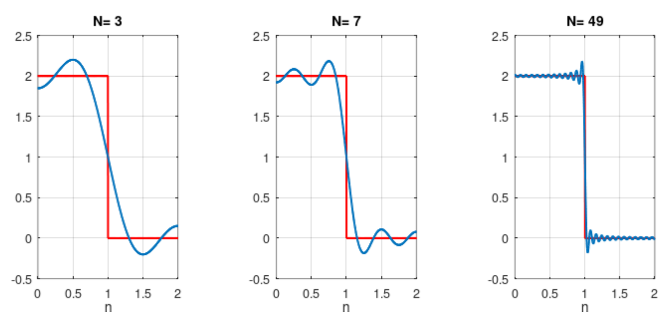
- **Why approximation by truncation could be a good idea** A justification of this method is the computation of the mean square error:

$$\begin{aligned}
MSE &= \frac{1}{2\pi}\int_{-\pi}^{\pi} |H(e^{j\omega}) - \hat{H}(e^{j\omega})|^2 d\omega \\
&= ||H(e^{j\omega}) - \hat{H}(e^{j\omega})||^2 \\
&= ||h[n] - \hat{h}[n]||^2 \\
&= \sum_{n=-\infty}^{\infty} |h[n] - \hat{h}[n]|^2
\end{aligned}$$

The means square error MSE is minimized by symmetric impulse truncation around zero

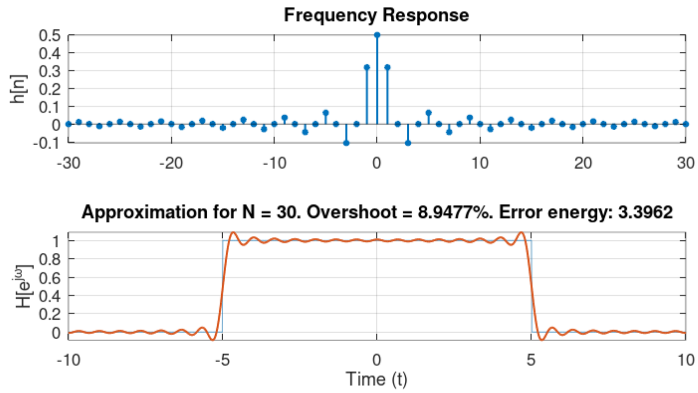
- **Why approximation by truncation is not such a good idea** The maximum error around the cutoff frequency is around 9% of the height of the jump regardless of N. This is known as the Gibbs Phenomenon.

1. The Gibbs Phenomenon



References:

- Matlab Answers



1.1.3 Window method

The impulse truncation can be interpreted as the product of the ideal filter response and a rectangular window of N points.

From the modulation theorem, the DTFT of the product f two signals is equivalent to the convolution of their DTFTs. Hence, the choice of window influences the quality of the approximation results.

Window Method

The window method is just a generalization of the impulse truncation method where we use a different window shape.

For example, by using a triangular window, we reduce the Gibbs error at the price of a longer transition.

1. The modulation theorem revisited. We can consider the approximated filter as

$$\hat{h}[n] = h[n]\,w[n]$$

with the indicator function w[n]

$$w[n] = \begin{cases} 1 & |n| \leq N \\ 0 & \text{otherwise} \end{cases}$$

The question is how can we express the Fourier Transform $\hat{H}(e^{j\omega}) =?$ of the filter as the product of two sequences? For that, we have to study the modulation theorem.

- Convolution Theorem states that the Fourier Transform of the convolution of two sequences is the product in the frequency domain of the Fourier Transforms.

$$DTFT\{(x * y)[n]\} = X(e^{j\omega})\, Y(e^{j\omega})$$

- Modulation Theorem The modulation theorem states that the Fourier Transform of the product of two sequences is the convolution in the frequency domain of the Fourier Transform

$$DTFT\{(x[n]\ y)[n]\} = (X * Y)(e^{j\omega})$$

- Convolution in the Frequency Domain

in $\mathbb{C}^{\infty}$ the space of infinite support signals, the convolution can be defined in terms of the inner product of the two sequences.

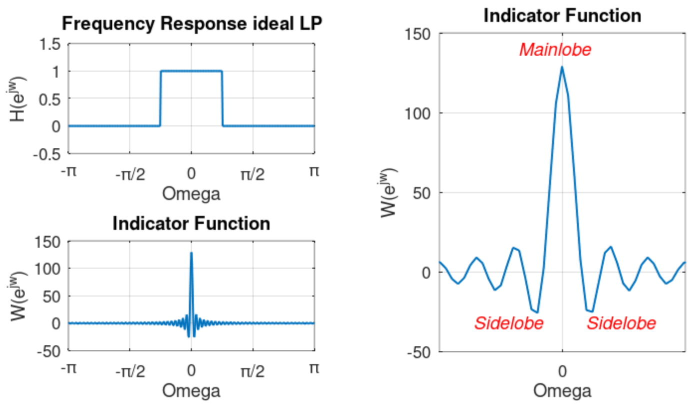
$$\begin{aligned}(x * y)[n] &= \Big\langle x^*[k], y[n-k] \Big\rangle \\ &= \sum_{n=-\infty}^{\infty} x[k] y[n-k]\end{aligned}$$

We can adapt the same strategie in $\mathbb{L}\Big(\big[-\pi, \pi\big]\Big)$, which is the space where the DTFT life's. So we find the convolution of two Fourier Transforms as the inner product of the first Fourier Transform conjugated and the second Fourier Transform frequence reversed and delayed by ω

$$\begin{aligned}(X * Y)(e^{j\,\omega}) &= \Big\langle X^*(e^{j\,\sigma}), Y(e^{j\,\omega-\sigma}) \Big\rangle \\ &= \frac{1}{2\pi}\int_{-\pi}^{pi} X^*(e^{j\,\sigma})\; Y(e^{j\,\omega-\sigma})\; d\sigma\end{aligned}$$

If we apply the definition of the inner product for $L2([-\pi,\, \pi])$ we get that the convolution between two Fourier Transforms.

2. Mainlobe and Sidelobes



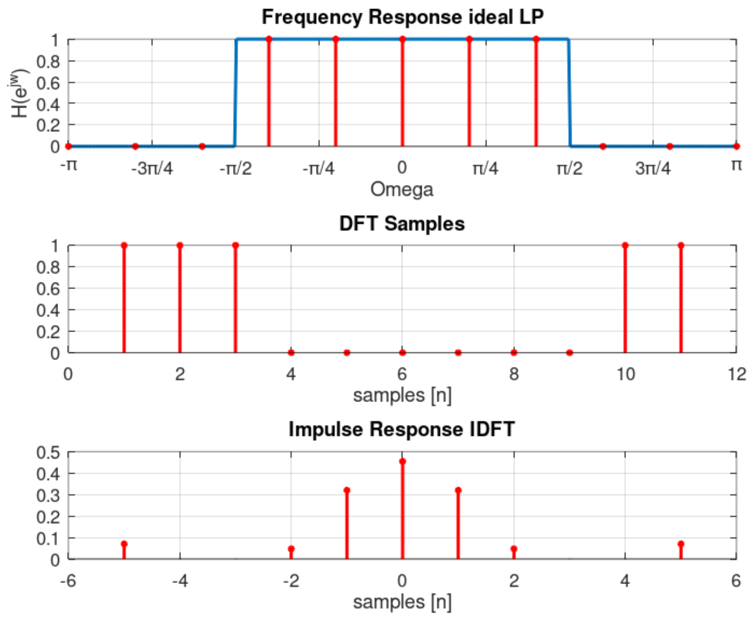
We want:

- narrow mainlobe $\Rightarrow$ to have sharp transition
- small sidelobe $\Rightarrow$ gibbs error is small
- short window $\Rightarrow$ FIR is efficient

1.1.4 Frequency sampling

Frequency Sampling

1. Draw desired frequency response $H(e^{j\omega})$
2. take M values at $\omega_k = \dfrac{2\pi}{M} \cdot k$
3. compute IDFT of values
4. use result as M-tap impulse response $\hat{h}[n]$



- **Why Frequency Sampling is not such a good idea:**
  - frequency response is DTFT of finite-support, whose DFT we know
  - frequency response is interpolation of frequency samples
  - interpolator is transform N-tap rectangular window (no escape from the indicator function)
  - again no control over main- and sidelobe

Summery Imitation

> These methods to approximate ideal filters are certainly very useful when we want to derive a quick and dirty prototype, and we don't have time to use more sophisticated filter design methods

1.2 Signal of the Day: Camera Resolution and space exploration

1.2.1 Rosettta Mission: Spacecraft

- Reaching Comet 67P. 10 years to get momentum to get its orbit.

- Resolution of taken pictures:

Resolution	at Distance	Year	
1km/pixel	86'000km	28. June 2014	
	12'000km	14. July 2014	
100m/pixel	5'500km	20. July 2014	
5.3m/pixel	285km	3. August 2014	
11cm/pixel	6km	14. February 2015	most detailed pictures of a planet

Is it necessary to send a probe for 10years into space to get high resolution pictures?

1.2.2 Image Formation

$$\begin{aligned} i(x,y) &= s(x,y) * h(x,y), \text{ i: image that is formed,} \\ &= s(x,y) * t(x,y) * p(x,y) \end{aligned}$$

- i: image that is formed on the retina or camera
- s: light sources (source image)
- h: transfer function of the light
- t: medium through the light is traveling
- p: point spread function (PSF), lenses and focal distance

The major enemy to image quality of telescope on earth are the atmospheric disturbances.

The pinhole camera A certain pixel density is required to distinguish light sources on the image plane. We might be tempted to say the maximum achievable resolution is only depend on the **resolution** of the sensor at the back of the camera. In reality the resolution is limited by pixel density resolution is limited by diffraction.

Diffraction (Beugung) The image of an original point light source will appear as a diffraction pattern. The diffraction pattern through a small circular aperture is called **Airy disk**.

Rayleigh's criterion Minimum angle θ between light point sources that guarantees resolution

$$\theta = 1.22\frac{\lambda}{D}$$

- λ : wave length of the light that hits the camera
- D : Diameter of the aperture

1.2.3 Seeing the Lunar Excursion Module (LEM)

- size of LEM $\approx$ 5m
- distance to the Moon $\approx$
- $Rightarrow\ \theta$ subtended by the LEM is $\approx 0.003arcsec$
- Hubble's aperture: 2.4m
- visible spectrum $\lambda \approx 550nm$
- Rayleigh's criterion: $\theta \approx 0.1arcsec$

  $\Rightarrow$ to see the LEM, Hubble should have an aperture of 80m!!!!

1.2.4 Rayleigh's criterion, Spatial Resolution

$$\delta x = 1.22\ f\ \frac{f}{D} = \theta \cdot f$$

If the pixel separation on the camera sensor is not less than δx our camera will be resolution limited rather than diffraction limited.

- f: foco length
- f/D: f-number

pixel density takes into account the size of the sensor.

1.2.5 What about mega pixels?

How many mega pixels one need on an commercial camera. This actually depends on the size of the sensor and on the optics:

- f-number of all trades: f/8
- spatial Rayleigh's criterion: $\delta x \approx 4\mu m$

- max pixel area $16 \cdot 10^{-5}$

  $\Rightarrow$ to opperate at the diffraction limit we need $62'500 pixels/mm^2$

  Highend camera usually have one of the following sensors:

- APS-C sensor (329mm^2): 20 MP $\Rightarrow$ the camera is operating at the defraction limit

- 35-mm sensor (864mm^2): 54 MP $\Rightarrow$ the camera is operating at the defraction limit

1.3 Realizable Filters

1.3.1 The Z-Transform

1. References . Signals and Systems for Dummies: Z-Transform

2. Z-Transform maps a discrete-time sequence x[n] onto a function of $\sum_{n=-\infty}^{\infty} x[n]\, z^{-n}$.

   $$x[n] = \sum_{n=-\infty}^{\infty} x[n]\, z^{-n} \tag{1}$$

   The z-Transform is an extension of the DTFT to the whole complex plane and is equal to the DTFT for $z = e^{j\omega}$.

   $$X(z)|_{z=e^{j\omega}} = DTFT\{x[n]\} \tag{2}$$

   Key properties of the z-Transform are:

   - linearity: $\mathcal{Z}\{\alpha x[n] + \beta y[n]\} = \alpha X(z) + \beta Y(z)$
   - time shift: $\mathcal{Z}\{x[n-N]\} = z^{-N} X(z)$

   Applying the z-transform to CCDE's

$$\sum_{k=0}^{N-1} a_k y[n-k] = \sum_{k=0}^{M-1} b_k x[n-k]$$
$$Y(z)\sum_{k=0}^{N-1} a_k z^{-k} = X(z)\sum_{k=0}^{M-1} b_k z^{-k}$$
$$Y(z) = H(z)X(z)$$

- **M input values**
- **N output values**

3. Constant Difference Equation A constant coefficent difference equation (CCDE) expresses the input-, output relationship of an LTI system as a linear combination of output samples equal to a linear combination of input samples

$$\boxed{\sum_{k=0}^{N-1} a_k y[n-k]} = \boxed{\sum_{k=0}^{M-1} b_k x[n-k]}$$

In the z-domain, a Constant Coefficent Difference Equation CCDE is represented as a ration $H(z)$ of two polynomials of z^{-1}.

$$H(z) = \frac{\sum\limits_{k=0}^{M-1} b_k z^{-k}}{\sum\limits_{k=0}^{N-1} a_k z^{-k}} \tag{3}$$

4. Frequency Response The frequency response of a filter is equal to this transfer function evaluated at $z =^{j\omega}$.

$$H(j\omega) = H(z)|_{Z=e^{j\omega}} = \frac{\sum\limits_{k=0}^{M-1} b_k z^{-k}}{\sum\limits_{k=0}^{N-1} a_k z^{-k}} \tag{4}$$

1.3.2 Z-Transform of the leaky integrator

$$
\begin{aligned}
y[n] &= (1-\lambda)x[n] + \lambda y[n-1] \\
Y(z) &= (1-\lambda)X(z) + \lambda z^{-1}Y(z) \\
Y(z) - \lambda z^{-1}Y(z) &= (1-\lambda)X(z) \\
Y(z)\big(1-\lambda z^{-1}\big) &= (1-\lambda)X(z) \\
Y(z) &= H(z)X(z) \\
H(z) &= \frac{Y(z)}{X(z)} = \frac{1-\lambda}{1-\lambda z^{-1}} \\
H(e^{j\omega}) &= \frac{1-\lambda}{1-\lambda e^{-j\omega}}
\end{aligned}
$$

1. LTI Systems

   An LTI system can be represented as the convolution $y[n] = x[n] * h[n]$. From the convolution property of the Z-transform, it follows that the z-transform of y[n] is:

   $$Y(z) = H(z)\ X(z) \tag{5}$$

1.3.3 Region of convergence

Conditions for convergences

- The zeros/poles are the roots of the numerator/denominator of the rational transfer function
- the region of convergence is only determined by the magnitude of the poles
- the z-transform of a causal LTI system extends outwards from the largest magnitude pole

BIBO-Stable

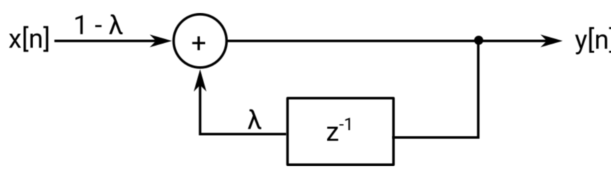
An LTI system is stable if its region of convergence includes the unit circle

1.4 Filter Design Part 2

- many signal processing problems can be solved using simple filters
- we have seen simple lowpass filters already (Moving Average, Leaky Integrator)
- simplel (low order) transfer functions allow for intuitive design and tuning

1.4.1 Intuitive IIR Designs

1. Leaky Integrator
   (a) Filter Structure



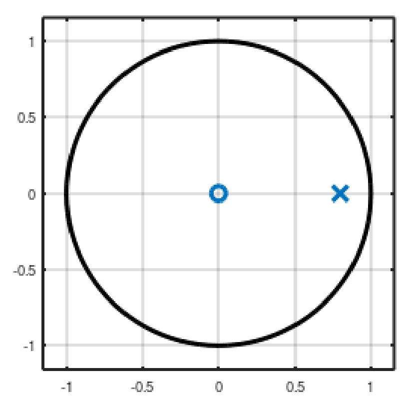
   (b) Transfer Function

$$H(z) = \frac{1-\lambda}{1-\lambda z^{-1}}$$

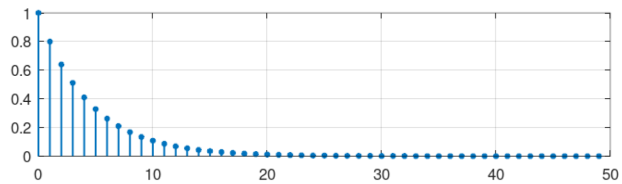
   (c) CCDE

$$y[n] = (1-\lambda)\, x[n] + \lambda\, y[n-1]$$

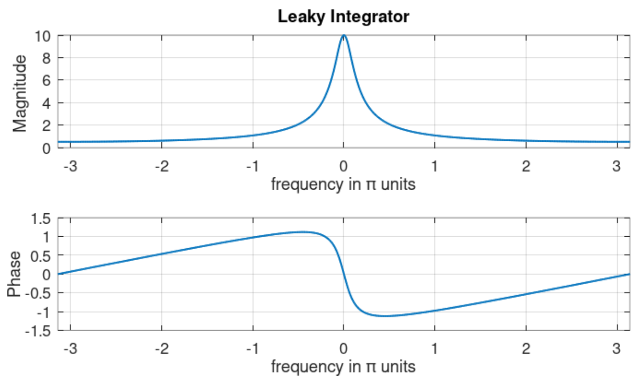
   (d) Pole-Zero Plot



   (e) Impulse response



(f) Frequency Response



2. Resonator

- a resonator is a narrow bandbass filter
- used to detect presence of a given frequency
- useful in communication systems and telephone (DTMF)
- Idea: shift passband of the Leaky Integrator

(a) Transfer Function

$$H(z) = \frac{G_0}{(1 - pz^{-1})(1 - p^* z^{-1})}$$

$$p = \lambda e^{j\omega_0}$$

$$H(z) = \frac{G_0}{1 - 2\mathcal{R}pz^{-1} + |p|^2 z^{-2}}$$

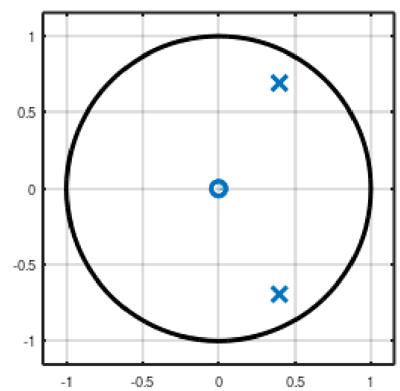
$$H(z) = \frac{G_0}{1 - 2\lambda\omega_0 z^{-1} + |\lambda|^2 z^{-2}}$$

The coeffience to be used in the CCDE

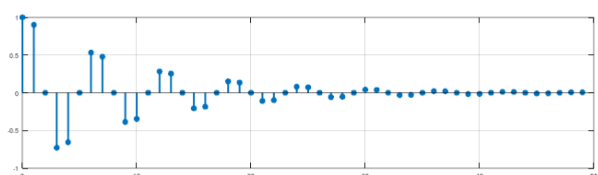
$$a_1 = 2\lambda cos\omega_0$$
$$a_2 = -|\lambda|^2$$

(b) Pole-Zero Plot

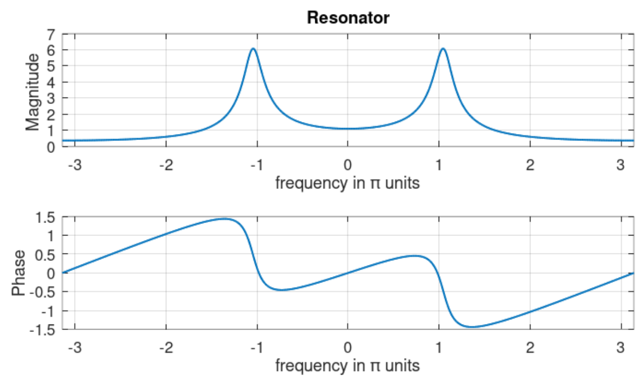
- Move the pole of the leaky integrator radially around the circle of radius lambda to shift the passband at the frequency that we are interested in, i.e. ω_0. interested in selecting. Since we want a real filter, we also have to create a complex conjugate pole at an angle that is $-\omega_0$.



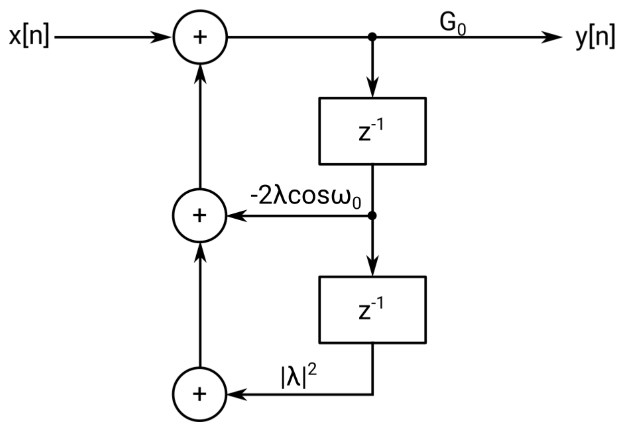
(c) Impulse response



(d) Frequency Response



(e) Filter Structure



3. DC Removal

- a DC-balances signal has zero sum: $\lim_{N\to\infty}\sum_{n=-N}^{N} x[n] = 0$ i.e. there is no Direct Current component
- its DTFT value at zero is zero for an $\omega = 0$
- we want to remove the DC bias from a non zero-centered signal
- we want to kill the frequency component at $\omega = 0$

(a) Transfer Function

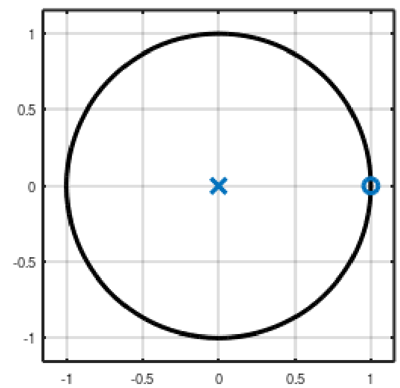
$$H(z) = 1 - Z^{-1}$$

(b) CCD

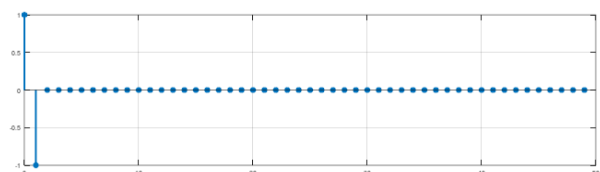
$$y[n] = x[n] - x[n-1]$$

(c) Pole-Zero Plot

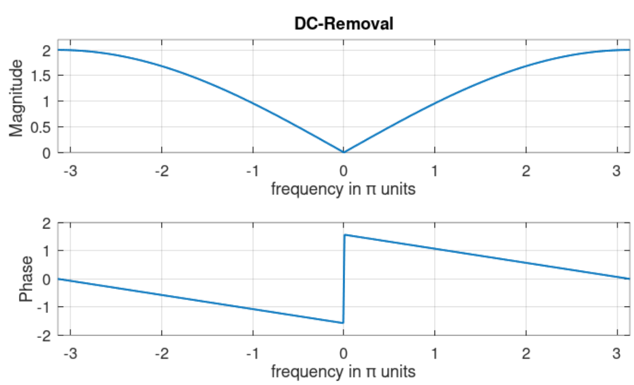
- Simply place a zero at \$ z = 1\$



(d) Impulse response



(e) Frequency response



This is not an acceptable characteristic because it introduces a very big attenuation over almost the entety of the frequency support.

4. DC Removal Improved - DC-Notch Filter

(a) Transfer Function

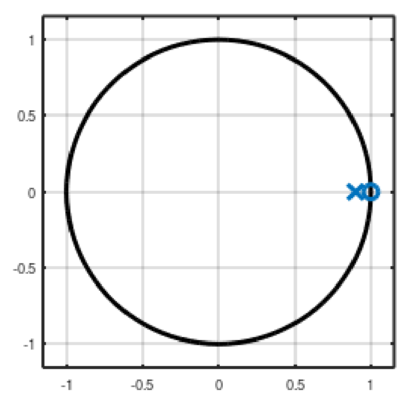
$$H(z) = \frac{1 - z^{-1}}{1 - \lambda z^{-1}}$$

(b) CCDE

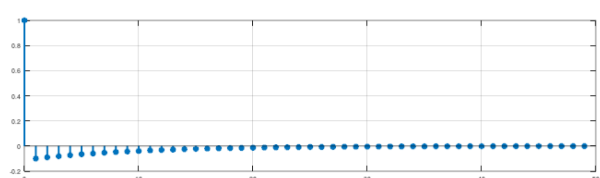
$$y[n] = \lambda y[n-1] + x[n] - x[n-1]$$

(c) Pole-Zero Plot

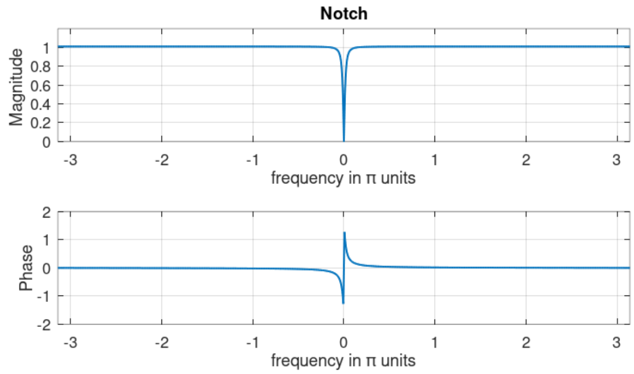
- and if we remember the circus tent method, we know that we can push up the z-transform by putting a pole in the vicinity of the 0. So we try and do that and we combine therefore, the effect of a 0 and 1 with the effect of a pole close to one, and inside the unit circle, for obvious reasons of stability.



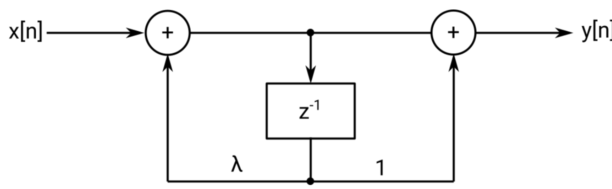
(d) Impulse response



(e) Frequency Response



(f) Filter Structure



5. Hum Removal

- The hum removal filter is to the dc notch what the resonator is to the leaky integrator
- similar to DC removal but want to remove a specific nonzero frequency
- very usful for musicaians amplifiers for electronic guitars pick up the hum from the electronic mains (50Hz in Europe and 60Hz in North America)
- we need to tune the hum removal according the country

(a) Transfer Function

$$
\begin{aligned}
H(z) &= \frac{(1-e^{j\omega_0}z^{-1})(1-e^{-j\omega_0}z^{-1})}{(1-\lambda e^{j\omega_0}z^{-1})(1-\lambda e^{-j\omega_0}z^{-1})} \\
p &= e^{j\omega_0} \\
q &= \lambda e^{j\omega_0} \\
&= \frac{(1-pz^{-1})(1-p*z^{-1})}{(1-qz^{-1})(1-q*z^{-1})} \\
H(z) &= \frac{1-2\mathcal{R}pz^{-1}+|p|^2z^{-2}}{1-2\mathcal{R}qz^{-1}+|q|^2z^{-2}} \\
&= \frac{1-2\omega_0 z^{-1}+z^{-2}}{1-2\lambda\omega_0 z^{-1}+|\lambda|^2z^{-2}}
\end{aligned}
$$

The coeffience to be used in the CCDE

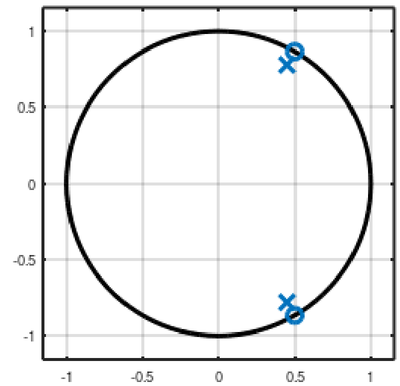
$$
\begin{aligned}
a_1 &= -2\lambda cos\omega_0 \\
a_2 &= |\lambda|^2 \\
b_1 &= -2\omega_0 \\
b_2 &= 1
\end{aligned}
$$

(b) CCDE

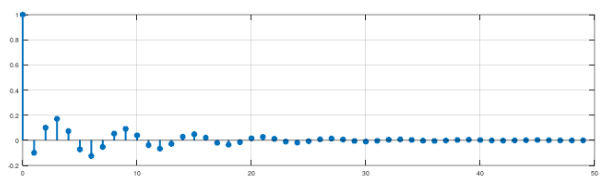
$$y[n] = 2\lambda\ cos\omega_0\ y[n{-}1]{+}|\lambda|^2\ y[n{-}2]{+}x[n]{-}2\ cos\omega_0\ x[n{-}1]{+}\ x[n{-}2]$$

(c) Pole-Zero Plot

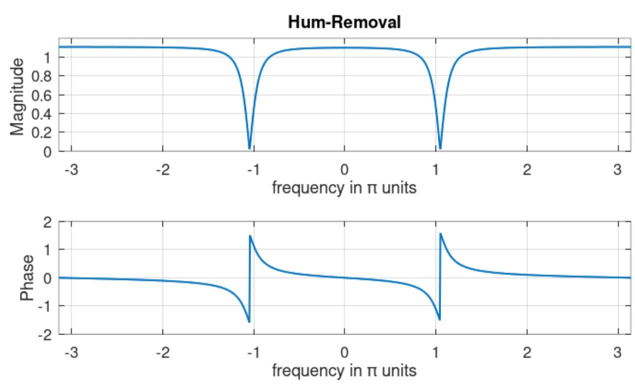
- and if we remember the circus tent method, we know that we can push up the z-transform by putting a pole in the vicinity of the 0. So we try and do that and we combine therefore, the effect of a 0 and 1 with the effect of a pole close to one, and inside the unit circle, for obvious reasons of stability.



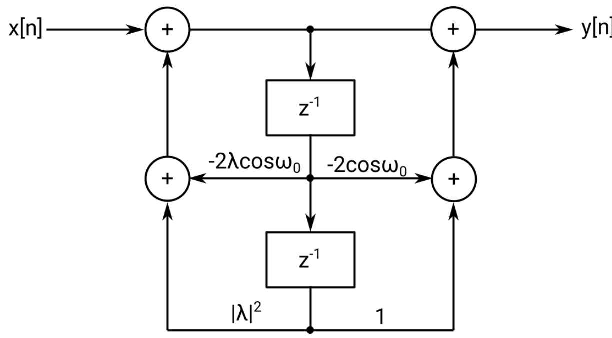
(d) Impulse response



(e) Frequency Response



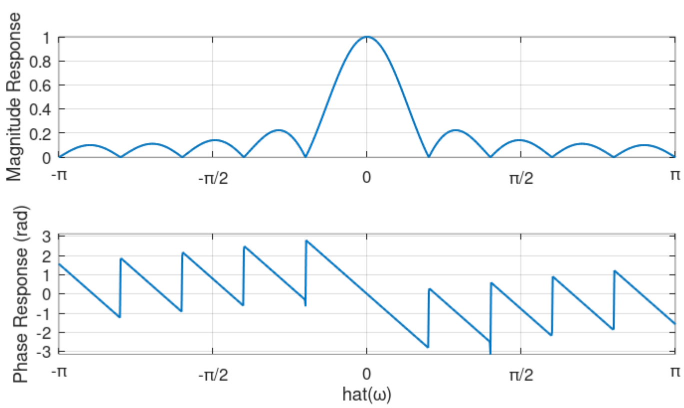
(f) Filter Structure



1.4.2 Matlab

Dirichlet The Dirichlet or periodic sync function can be used to analyze Moving Average Filters $D_M(j\omega) = diric(\omega, M) = \dfrac{sin(\frac{\omega}{2}M)}{sin(\frac{\omega}{2}M)}$

Freqz The frequency response can be plotted most easily using freqz() function.



1.5 Filter Design Part 3

1.5.1 Filter Specification

1.5.2 IIR Design

Filterdesign was established art long before digital processing appeared

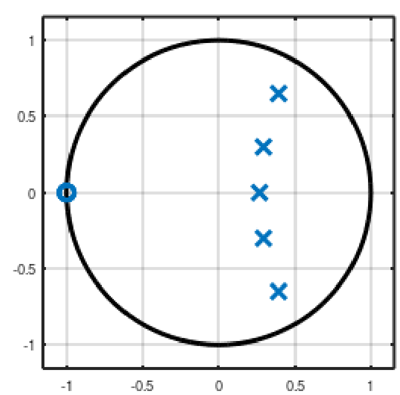
- AFD: Analog Filter Design
- lots of nice analog filters exist

- methods exist to ”translate” the analog design into a rational transfer function
  - **impulse invariance transformation**, preserves the shape of the impulse response
  - finite difference approximation, converts a differential equation into a ccde
  - step invariance, preserves the shape of the step response
  - matched-z transformation, matches the pole-zero representation
  - **bilinear transformation**, preserves the system function representation
- most numerical packages (Matlab, etc.) provide ready-made routines
- design involves specifying some parameters and testing that the specs are fulfilled

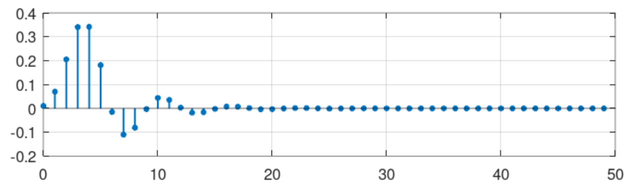
1. Butterworth lowpass

Magnitude response	**Design Parameters**	**Test values**
maximally flat	order N	width of transition band
monotonic over $[0, \pi]$	cutoff frequency	passband error

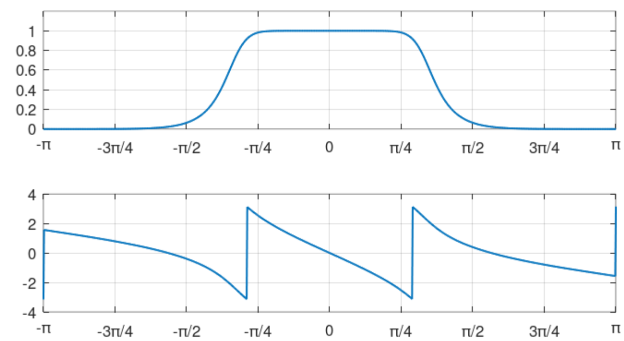
   (a) Pole-Zero Plot



   (b) Impulse Response



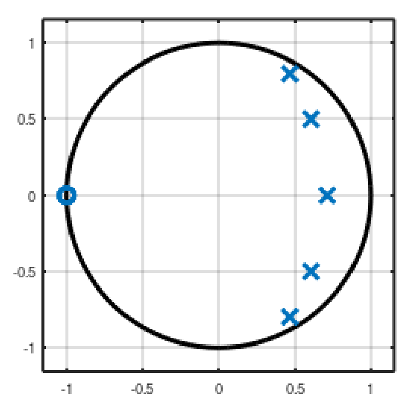
(c) Frequency Response



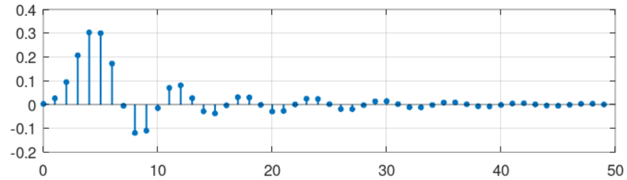
2. Chebyshev lowpass

Magnitude response	Design Parameters	Test values
equiripple in passband	order N	width of transition band
monotonic in stopband	passband max error	stopband error
	cutoff frequency	

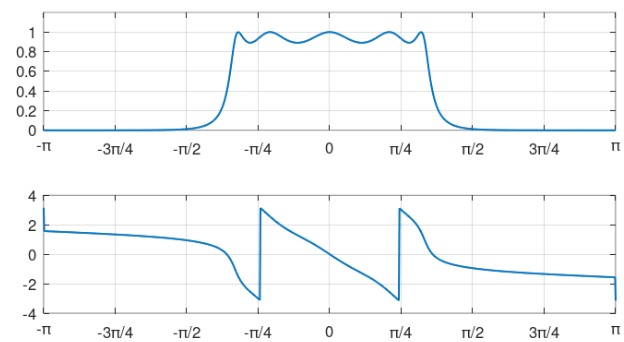
(a) Pole-Zero Plot



(b) Impulse Response



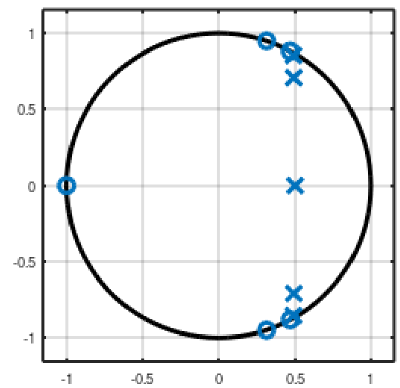
(c) Frequency Response



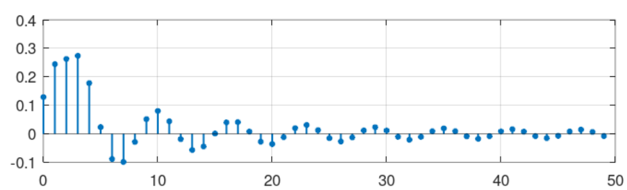
3. Elliptic lowpass

Magnitude response	Design Parameters	Test values
equiripple in passband	order N	width of transition band
equiripple in stopband	cutoff frequnecy	
	passband max error	
	stopband min attenuation	

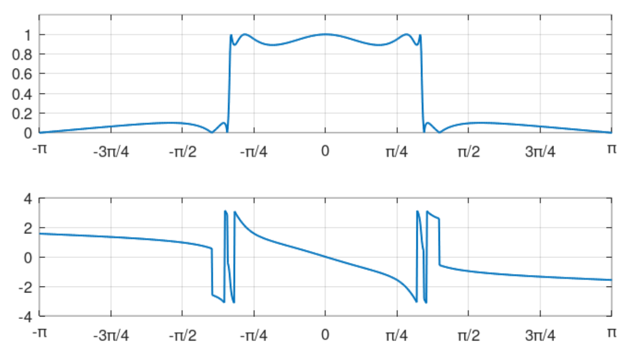
(a) Pole-Zero Plot



(b) Impulse Response



(c) Frequency Response



1.5.3 FIR Design

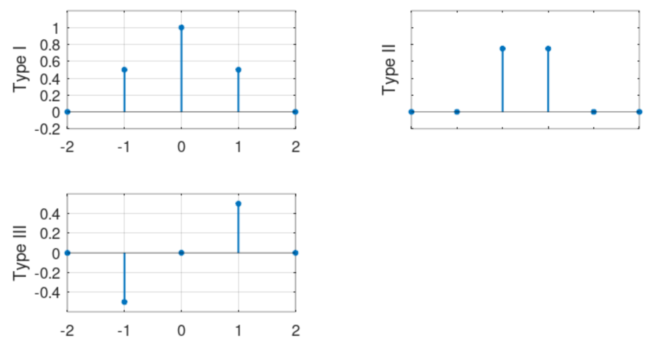
1. Optimal minmax design FIR filters are `digital` signal processing ”exclusivity”. In the 70s Parks and McClellan developed an algorithm to design optimal FIR filters:

    - linear phase

- equiripple error in passband and stopband

algorithm proceeds by **minimizing** the maximum error in passpand and stopband

(a) Linear Phase Linear phase derives from a symmetric or antisymmetric impulse respones



Type I-Filters Odd length impulse response, and are symmetric

Type II-Filters Even length impulse response, and are symmetric

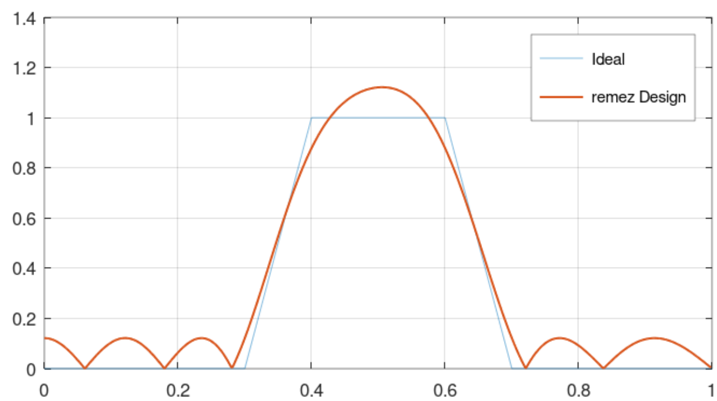
Type III-Filters Odd length impulse response, and are antisymmetric

Type IV-Filters Even length impulse response, and are antisymmetric

Type-II and Type-IV Filters are symmetric and antisymmetric filters, respectively, both of which have an even number of taps. That means that the center symmetry of these filters fall in between samples. And so they both introduce a non integer linear phase factor, of one half sample.

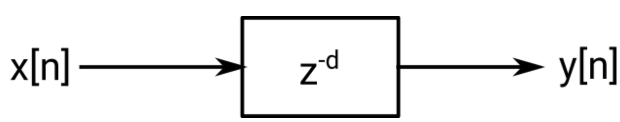
1.5.4 The Park McMellon Design Algorithm

Magnitude response	Design Parameters	Test values
equiripple in passband and stopband	order N	passband max error
	passband edge ω_p	stopband max error
	stopband edge ω_s	
	ratio of passband to stopband error $\frac{\delta_p}{delta_s}$	



1.6 ONGOING Notes and Supplementary Materials

1.6.1 The Fractional Delay Filter (FDF)



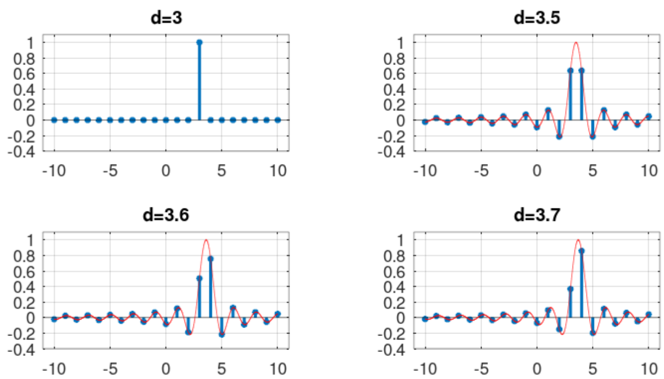
The transfer function of a simple delay z^{-d} is:

$$H(e^{j\omega}) = e^{-j\omega d},\, d \in \mathbb{Z}$$

what happens if, in $H(e^{j\omega}$ we use a non-integer $d \in \mathbb{R}$?

1. Impulse Response

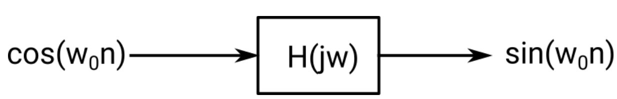
$$\begin{aligned}
h[n] &= IDFT\left\{e^{j\omega d}\right\} \\
&= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega d} e^{j\omega n} d\omega \\
&= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega(n-d)} d\omega \\
&= \frac{1}{\pi(n-d)} \frac{e^{j\pi(n-d)} - e^{-j\pi(n-d)}}{2j} \\
&= \frac{sin\pi(n-d)}{\pi(n-d)} \\
&= sinc(n-d)
\end{aligned}$$



For now suffice it to say that we can actually interpolate in discrete time and find intermediate values of a discrete time sequence using just discrete times filters like the fractional delay

1.6.2 ONGOING The Hilbert Filter

- Demodulator



can we build such a thing?

1.6.3 Implementing of Digital Filters

1. Leaky Integrator in C

```
#include <stdio.h>
double leaky(double x) {
    static const double lambda = 0.9;
    static double y = 0;   // 1x memory cell
    // plus initialization

    // algorithm: 2x multiplication, 1x addition
    y = lambda * y + (1-lambda) *x;
    return y;
}

int main() {
    int n;
    for(n = 0; n <20; n++)
    {
        //call with delta signl
        printf("%.4f ", leaky(n==0 ? 1.0 : 0.0));
        if(!((n+1)%10)) printf("\n");
    }
}
```

0.1	0.09	0.081	0.0729	0.0656	0.059	0.0531	0.0478	0.043	0.0387
0.0349	0.0314	0.0282	0.0254	0.0229	0.0206	0.0185	0.0167	0.015	0.0135

- we need a ”memory cell” to store previous state
- we need to initialize the storage before first use
- we need 2 multiplications and one addition per output sampel

2. Moving Average in C

```
#include <stdio.h>
double ma(double x) {
    static const int M = 5;
    static double z[M]; // Mx memory cells
```

```
    static int ix = -1;

    int n;
    double avg = 0;

    if(ix == -1) {       // initalize storage
        for(n=0; n<M; n++)
            z[n] = 0;
        ix = 0;
    }

    z[ix] = x;
    ix = (ix + 1) % M;  // circular buffer

    for(n=0; n<M; n++)  // Mx additions
        avg += z[n];

    return avg / M;     // 1x division
}

int main() {
    int n;
    for (n = 0; n<20; n++)
    {
      // call with delta signl
        printf("%.4f ", ma(n==0 ? 1.0 : 0.0));
        if(!((n+1)%10)) printf("\n");
    }
}
```

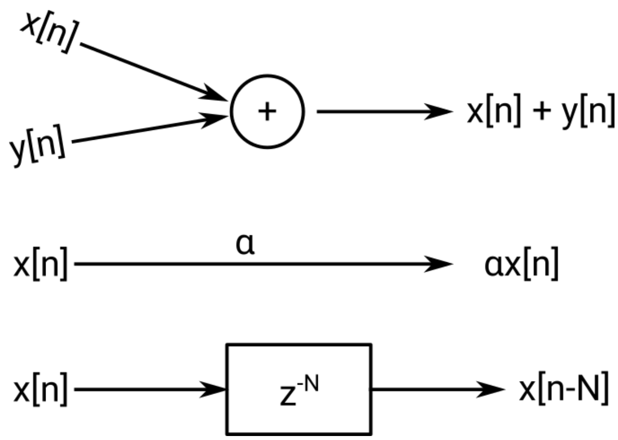
0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- we need M memory cells to store previous input values
- we need to initialize the storage before first use
- we need 1 division and M additions per output sample

3. Programming Abstraction

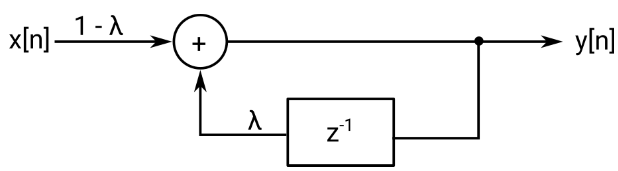
With this three building blocks we can describe and Constant Coefficient Equation.

(a) Building Blocks



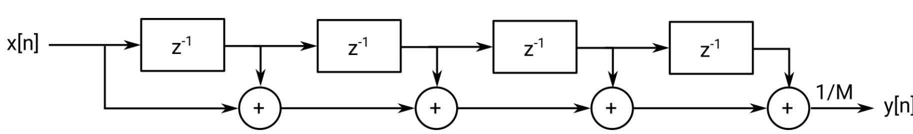
(b) Leaky Integrator

$$y[n] = \lambda y[n-1] + (1-\lambda)x[n]$$



(c) Moving Average

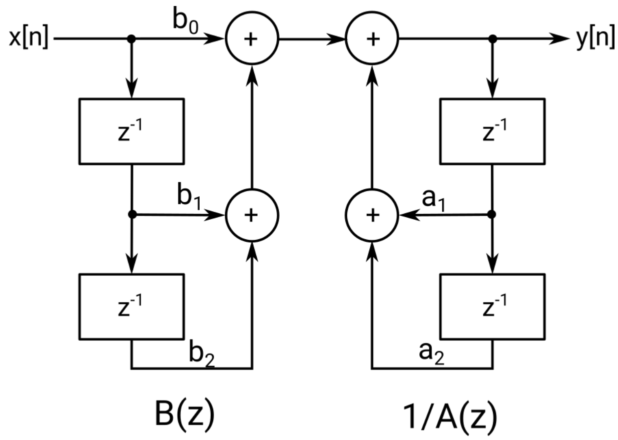
$$y[n] = \frac{1}{M}\sum_{k=0}^{M-1} x[n-k]$$



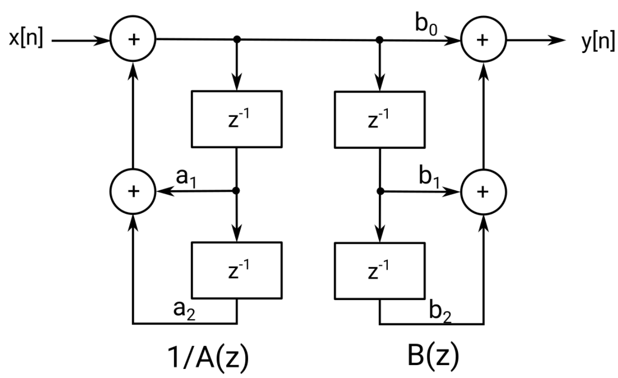
(d) The second-order section

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} = \frac{B(z)}{A(z)}$$

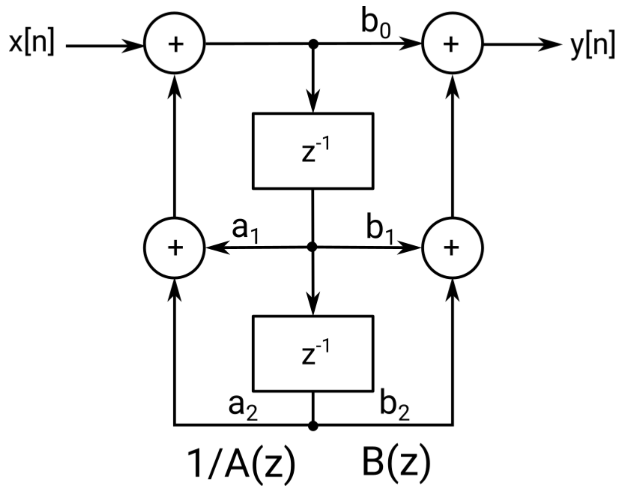
(e) Second-order section, direct form I



(f) Second-order section, inverted direct form I Because the convolution is commutative, numerator and denominator may be swapped.



(g) Second-order section, direct form II Since the content of the delay cells are exactly the same for all time, so we can lumb the delay cells together.



1.6.4 TODO Real-Time Processing

1. I/O and DMA Everything works in synch with a system clock of period T_s

   - record a value $x_i[n]$
   - process the value in a casual filter
   - play the output $x_o[n]$

   > Everything needs to happen in at most T_s seconds!

   Buffering:

   - interrupt for each sample would be too much overhead
   - soundcard consumes samples in buffers
   - soundcard notifies when buffer used up
   - CPU can fill a buffer in less time than soundcard can empty it

   Double Buffering

   - Delay $d = T_s \times \frac{L}{2}$ L: Lenght of the Buffer
   - If CPU doesn't fill the buffer fast enough: **underflow**

Multiple I/O Processing

- Delay: $d = T_s \times L$
- usually start out process first

2. Implementation Framework Low Level

   - study soundcard data sheet
   - write code to program soundcard via writes to IO Ports
   - write an interrupt handler
   - write the code to handle the data

   High Level

   - choose a good API
   - write a callback function to handle the data

3. Callback Prototype

```
int Callback( const void *input,     // pointer to the input buffer
              void *output,          // pointer to the output buffer
              unsigned long samples, // length of samples
    );
{
    float* pIn = (float*)input;      // Convert the generic buffers
    float* pOut = (float*)output;    // to the right data type
    for (int n=0; n < samples; n++)  // Calls process for each sample in input t
        *pOut++ = Process(*pIn++);   // and store the result into the output buf
}
```

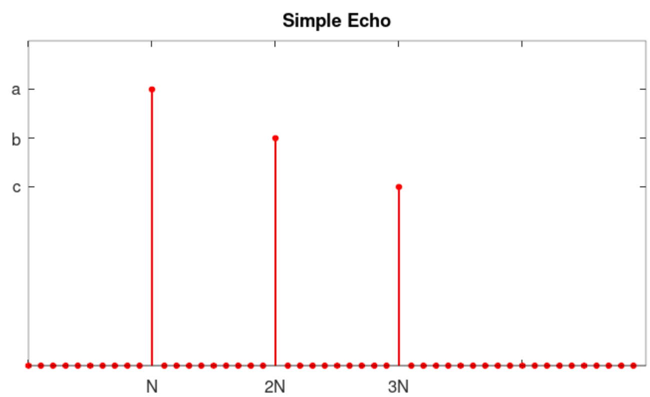
4. Processing Gateway

```
                                 // 10 sec @ 24kHz
enum {BUF_LEN = 0x10000};        // Buffer length, power of 2
enum {BUF_MASK = BUF_LEN -1};    // Circular buffer mask
float m_pY[BUF_LEN];             // 2 Buffers
float m_pX[BUF_LEN];
int m_Ix;                        // 2 indixes into the buffers
int m_Iy;
float Process(float Sample)
```

```
{
    m_PX[m_Ix] = Sample;          // store the sample into input buffer
    float y = Effect();           // call Effect()
    m_pY[m_Iy] = y;               // store the output into output buffer
    m_Ix = (m_Ix + 1) & BUF_MAS;  // Update indices with the ciruclar strategy
    m_IY = (m_Iy + 1) & BUF_MAS;
    return y;                     // return current output sample
}
```

5. Effect Implementing the echo effect as a reflection of the original signal, scaled with a factor at subsequent points in time:

$$y[n] = \frac{ax[n] + bx[n-N] + cx[n-2N]}{a+b+c}$$




```
float Echo() {
    static float a = 0.85;              // the three scaling factors
    static float b = 0.6f;
    static float c = 0.45f;
    static float norm = 1.0f/(a+b+c);   // the normalisation factor
    static int N = (int)(0.3 * m_SR);   // delay between reflextion

    return norm * ( a * m_pX[m_Ix]
                  + b * m_pX[(m_Ix + BUF_LEN -N)   & BUF_MASK]
                  + c * m_pX[(m_Ix + BUF_LEN -2*N) & BUF_MASK]); }
```

1.6.5 TODO Derevereration and echo cancellation