Filtering

Filtering in a broad sense is selecting portion(s) of data for some processing.

In many multimedia contexts this involves the removal of data from a signal — This is essential in almost all aspects of lossy multimedia data representations.

We will look at filtering in the frequency space very soon, but first we consider filtering via impulse responses.

We will look at:

IIR Systems : Infinite impulse response systems

FIR Systems : Finite impulse response systems

Infinite Impulse Response (IIR) Systems

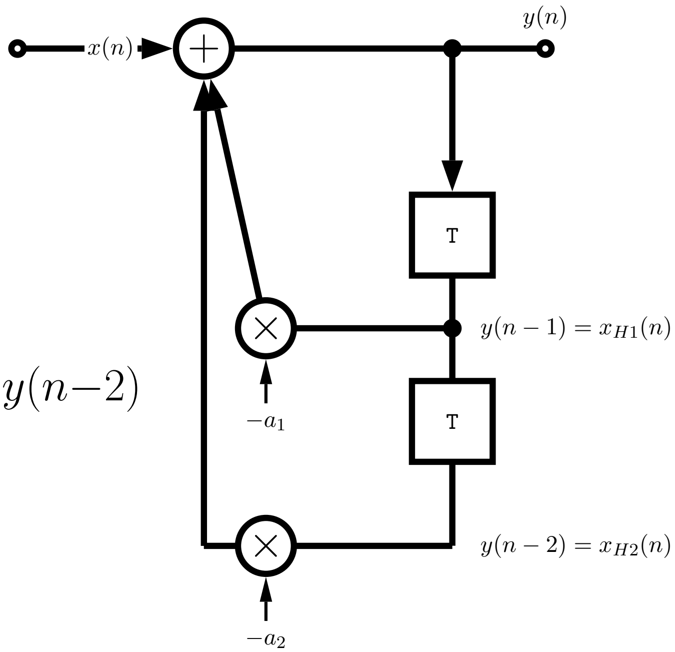
If $h(n)$ is an infinite impulse response function then the digital system is called and IIR system.

Example:

- The algorithm is represented by the difference equation:

$$y(n) = x(n) - a_1 y(n-1) - a_2 y(n-2)$$

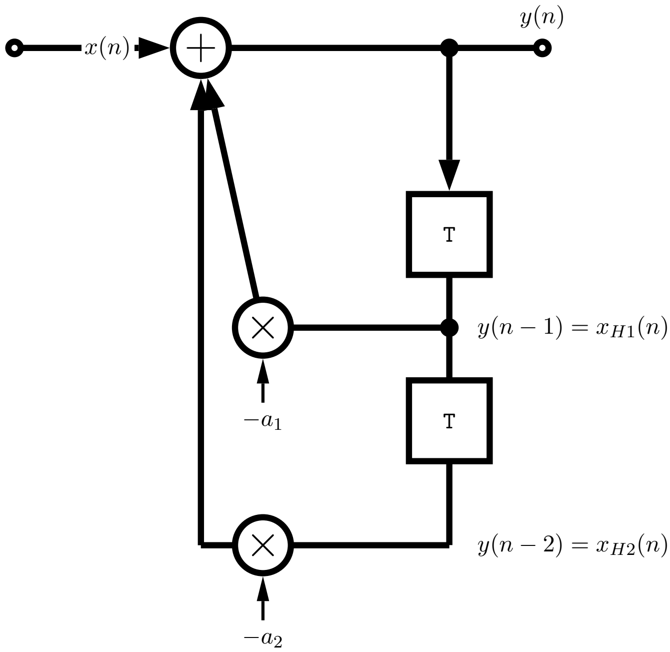
- This produces the opposite signal flow graph



Infinite Impulse Response (IIR)Systems Explained

The following happens:

- The output signal $y(n)$ is *fed back* through a series of delays
- Each delay is weighted
- Fed back weighted delay summed and passed to new output.
- Such a feedback system is called a **recursive system**



Z-transform of IIR

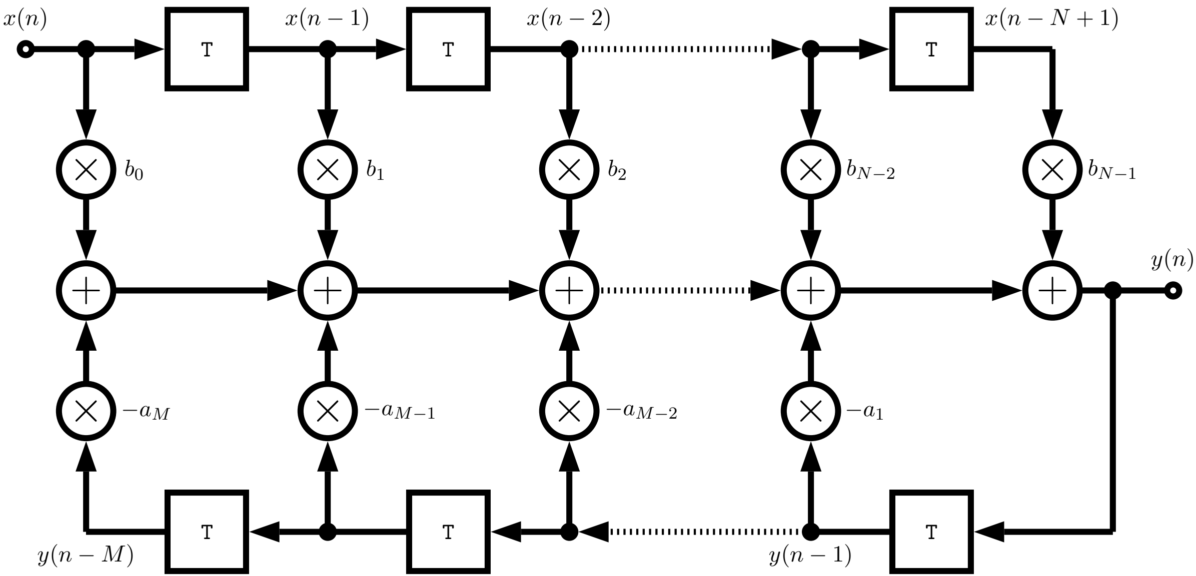
If we apply the Z-transform we get:

$$
\begin{aligned}
Y(z) &= X(z) - a_1 z^{-1} Y(z) - a_2 z^{-2} Y(z) \\
X(z) &= Y(z)(1 + a_1 z^{-1} + a_2 z^{-2})
\end{aligned}
$$

Solving for $Y(z)/X(z)$ gives $H(z)$ our transfer function:

$$
H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}}
$$

A Complete IIR System

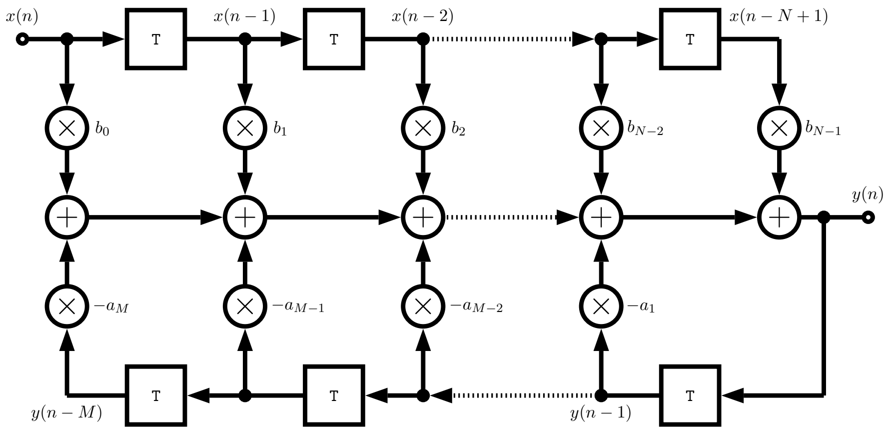


Here we extend:

The **input** delay line up to $N-1$ elements and

The **output** delay line by M elements.

Complete IIR System Algorithm



We can represent the IIR system algorithm by the difference equation:

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

Complete IIR system Transfer Function

The Z-transform of the difference equation is:

$$Y(z) = -\sum_{k=1}^{M} a_k\, z^{-k} Y(z) + \sum_{k=0}^{N-1} b_k\, z^{-k} X(z)$$

and the resulting **transfer function** is:

$$H(z) = \frac{\sum_{k=0}^{N-1} b_k\, z^{-k}}{1 + \sum_{k=1}^{M} a_k\, z^{-k}}$$

Filtering with IIR

We have **two** **filter banks** defined by vectors: $A = \{a_k\}$, $B = \{b_k\}$.

These can be applied in a *sample-by-sample* algorithm:

- MATLAB provides a generic `filter(B,A,X)` function which filters the data in vector `X` with the filter described by vectors `A` and `B` to create the filtered data `Y`.

  The filter is of the standard difference equation form:

$$\begin{aligned} a(1)*y(n) &= b(1)*x(n)+b(2)*x(n-1)+\ldots+b(nb+1)*x(n-nb) \\ &\quad -a(2)*y(n-1)-\ldots-a(na+1)*y(n-na) \end{aligned}$$

- Filter banks can be created manually or MATLAB can provide some predefined filters — **more later, see tutorials**
- See also `help filter`, online MATLAB docs and tutorials on filters.

Filtering with IIR: Simple Example

The MATLAB file IIRdemo.m sets up the filter banks as follows:

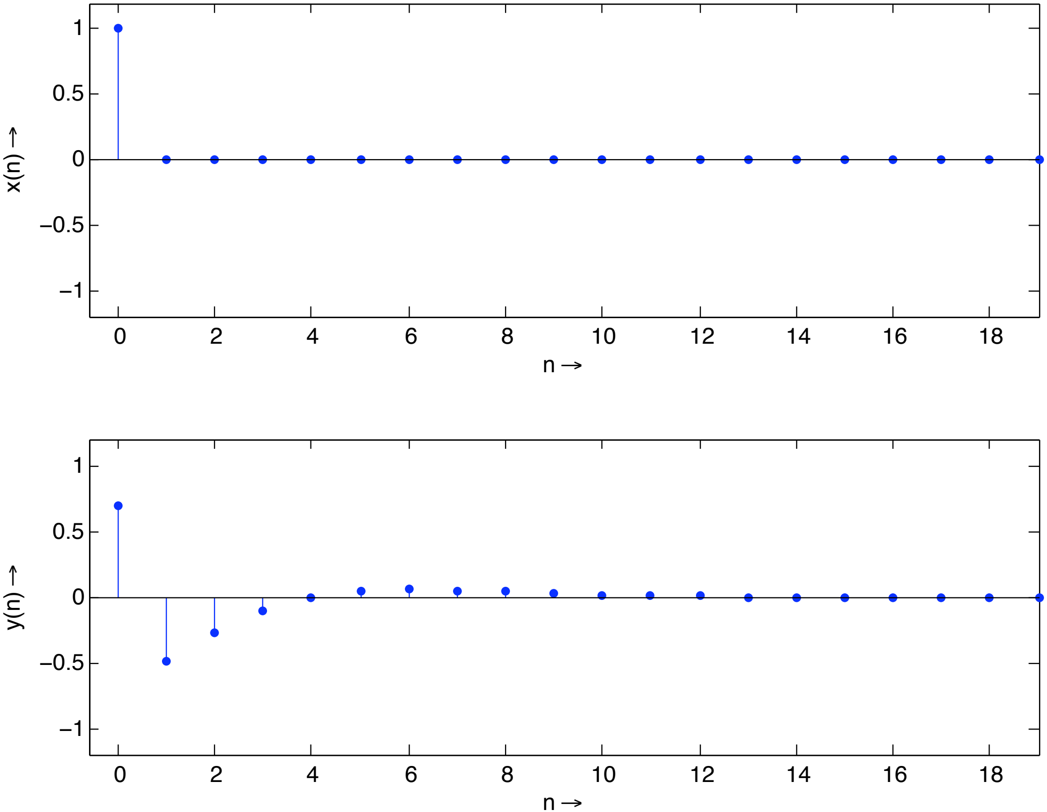
```
fg=4000;
fa=48000;
k=tan(pi*fg/fa);

b(1)=1/(1+sqrt(2)*k+k^2);
b(2)=-2/(1+sqrt(2)*k+k^2);
b(3)=1/(1+sqrt(2)*k+k^2);
a(1)=1;
a(2)=2*(k^2-1)/(1+sqrt(2)*k+k^2);
a(3)=(1-sqrt(2)*k+k^2)/(1+sqrt(2)*k+k^2);
```

and then applies the difference equation:

```
for n=1:N
y(n)=b(1)*x(n) + b(2)*xh1 + b(3)*xh2 - a(2)*yh1 - a(3)*yh2;
xh2=xh1;xh1=x(n);
yh2=yh1;yh1=y(n);
end;
```

Filtering with IIR: Simple Example Output

This produces the following output:



MATLAB filters

Matlab `filter()` function implements an IIR (or an FIR no A components).

Type `help filter`:

```
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)

    If a(1) is not equal to 1, FILTER normalizes the filter
    coefficients by a(1).

    FILTER always operates along the first non-singleton dimension,
    namely dimension 1 for column vectors and non-trivial matrices,
    and dimension 2 for row vectors.
```

Using `filter()` in Practice

We have **two** **filter banks** defined by vectors: $A = \{a_k\}$, $B = \{b_k\}$. We have to specify some values for them.

- We can do this by hand — we could design our own filters
- MATLAB provides standard functions to set up A and B for many common filters.

Using MATLAB to make filters

MATLAB provides a few built-in functions to create ready made filter parameter A and B:

E.g: `butter`, `buttord`, `besself`, `cheby1`, `cheby2`, `ellip`, `freqz`, `filter`.

For our purposes the Butterworth filter will create suitable filters, `help butter`:

```
BUTTER Butterworth digital and analog filter design.
    [B,A] = BUTTER(N,Wn) designs an Nth order lowpass digital
    Butterworth filter and returns the filter coefficients in length
    N+1 vectors B (numerator) and A (denominator). The coefficients
    are listed in descending powers of z. The cutoff frequency
    Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to
    half the sample rate.

    If Wn is a two-element vector, Wn = [W1 W2], BUTTER returns an
    order 2N bandpass filter with passband  W1 < W < W2.
    [B,A] = BUTTER(N,Wn,'high') designs a highpass filter.
    [B,A] = BUTTER(N,Wn,'low') designs a lowpass filter.
    [B,A] = BUTTER(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].
```

Using MATLAB to make filters

help buttord:

```
BUTTORD Butterworth filter order selection.
 [N, Wn] = BUTTORD(Wp, Ws, Rp, Rs) returns the order N of the lowest
 order digital Butterworth filter that loses no more than Rp dB in
 the passband and has at least Rs dB of attenuation in the stopband.
 Wp and Ws are the passband and stopband edge frequencies, normalized
 from 0 to 1 (where 1 corresponds to pi radians/sample). For example,
        Lowpass:    Wp = .1,       Ws = .2
        Highpass:   Wp = .2,       Ws = .1
        Bandpass:   Wp = [.2 .7],  Ws = [.1 .8]
        Bandstop:   Wp = [.1 .8],  Ws = [.2 .7]
 BUTTORD also returns Wn, the Butterworth natural frequency (or,
 the "3 dB frequency") to use with BUTTER to achieve the
 specifications.
```

Using MATLAB Filter Example: Subtractive Synthesis Lecture Example

The example for studying subtractive synthesis, subtract_synth.m, uses the `butter` and `filter` MATLAB functions:

```
% simple low pas filter example of subtractive synthesis
Fs = 22050;
y = synth(440,2,0.9,22050,'saw');

% play sawtooth e.g. waveform
doit = input('\nPlay Raw Sawtooth? Y/[N]:\n\n', 's');
if doit == 'y',
  figure(1)
plot(y(1:440));
playsound(y,Fs);
end

%make lowpass filter and filter y
[B, A] = butter(1,0.04, 'low');
yf = filter(B,A,y);

[B, A] = butter(4,0.04, 'low');
yf2 = filter(B,A,y);
```

```
% play filtererd sawtooths
doit = ...
    input('\nPlay Low Pass Filtered (Low order) ? Y/[N]:\n\n', 's');
if doit == 'y',
figure(2)
plot(yf(1:440));
playsound(yf,Fs);
end

doit = ...
  input('\nPlay Low Pass Filtered (Higher order)? Y/[N]:\n\n', 's');
if doit == 'y',
    figure(3)
plot(yf2(1:440));
playsound(yf2,Fs);
end

%plot figures
doit = input('\Plot All Figures? Y/[N]:\n\n', 's');
if doit == 'y',
figure(4)
plot(y(1:440));
hold on
plot(yf(1:440),'r+');
plot(yf2(1:440),'g-');
end
```

synth.m

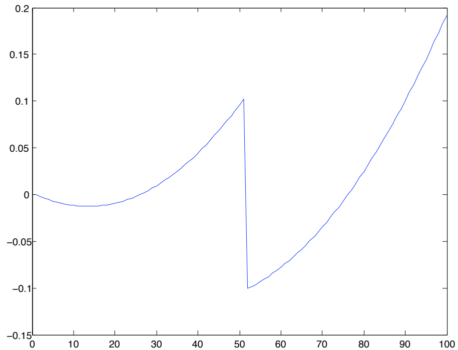
The supporting function, synth.m, generates waveforms as we have seen earlier in this tutorial:

```
function y=synth(freq,dur,amp,Fs,type)
% y=synth(freq,dur,amp,Fs,type)
%
% Synthesize a single note
%
% Inputs:
%  freq - frequency in Hz
%  dur - duration in seconds
%  amp - Amplitude in range [0,1]
%  Fs -  sampling frequency in Hz
%  type - string to select synthesis type
%         current options: 'fm', 'sine', or 'saw'

if nargin<5
  error('Five arguments required for synth()');
end

N = floor(dur*Fs);
n=0:N-1;
if (strcmp(type,'sine'))
  y = amp.*sin(2*pi*n*freq/Fs);
```

```
elseif (strcmp(type,'saw'))

  T = (1/freq)*Fs;      % period in fractional samples
  ramp = (0:(N-1))/T;
  y = ramp-fix(ramp);
  y = amp.*y;
  y = y - mean(y);

elseif (strcmp(type,'fm'))

  t = 0:(1/Fs):dur;
  envel = interp1([0 dur/6 dur/3 dur/5 dur], [0 1 .75 .6 0], 0:(1/Fs):dur);
  I_env = 5.*envel;
  y = envel.*sin(2.*pi.*freq.*t + I_env.*sin(2.*pi.*freq.*t));

else
  error('Unknown synthesis type');
end

% smooth edges w/ 10ms ramp
if (dur > .02)
  L = 2*fix(.01*Fs)+1;  % L odd
  ramp = bartlett(L)';  % odd length
  L = ceil(L/2);
  y(1:L) = y(1:L) .* ramp(1:L);
  y(end-L+1:end) = y(end-L+1:end) .* ramp(end-L+1:end);
end
```

synth.m (Cont.)

Note the *sawtooth* waveform generated here has a non-linear up slope:



This is created with:

```
ramp = (0:(N-1))/T;
y = ramp-fix(ramp);
```

`fix` rounds the elements of X to the nearest integers towards zero.

This form of sawtooth sounds slightly less harsh and is more suitable for audio synthesis purposes.

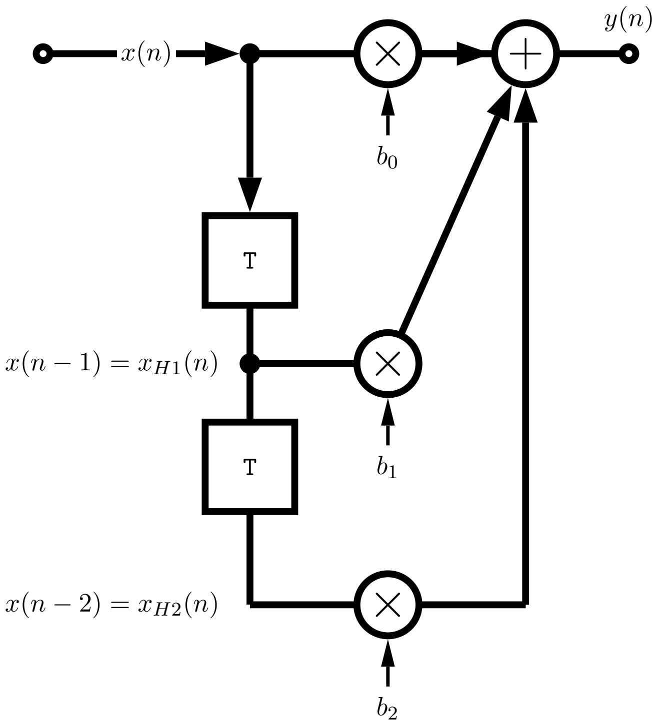
Finite Impulse Response (FIR) Systems

FIR system's are slightly simpler — there is no feedback loop.

A simple FIR system can be described as follows:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2)$$

- The input is fed through delay elements
- Weighted sum of delays give $y(n)$



Simple FIR Transfer Function

Applying the Z-transform we get:

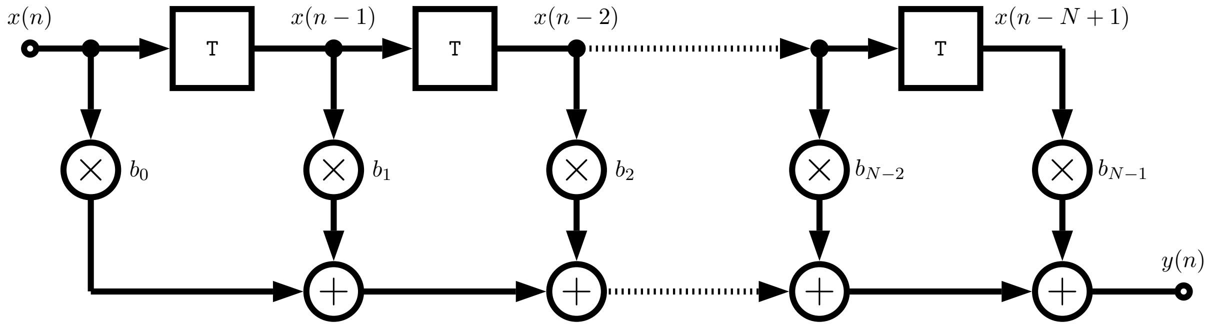
$$Y(z) = b_0 X(z) + b_1 z^{-1} X(z) + b_2 z^{-2} X(z)$$

and we get the transfer function:

$$H(z) = \frac{Y(z)}{X(z)} = b_0 + b_1 z^{-1} + b_2 z^{-2}$$

A Complete FIR System

To develop a more complete FIR system we need to add $N-1$ feed forward delays:



We can describe this with the algorithm:

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

Complete FIR System
Impulse Response and Transfer Function

The FIR system has the finite impulse response given by:

$$h(n) = \sum_{k=0}^{N-1} b_k \, \delta(n-k)$$

This means that each impulse of $h(n)$ is a weighted shifted unit impulse.

We can derive the transfer function as:

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

Signal
Flow Graphs: More on Construction

RECAP

We use a simple *equation* relation to describe the algorithm.

We will need to consider *three* basic components:

- Delay
- Multiplication
- Summation

Hints for Constructing Signal Flow Graphs

Apart from the three basic building blocks of *Delay, Addition and Multiplication* there are two other tools that we can exploit:

- Feedback loops — merged back with *Delay, Addition and/or Multiplication*.

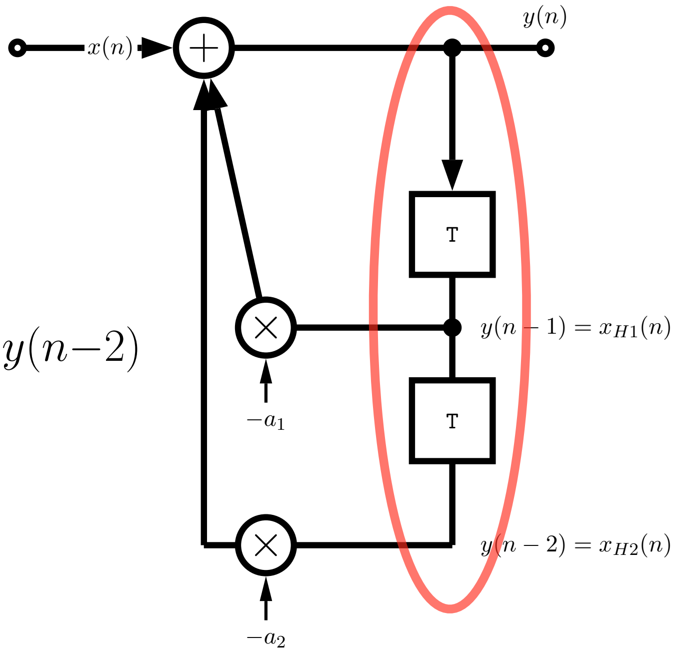
  Frequently (In many of our examples) we tap the output $y(n)$ and then delay *etc.* this.

  – $y(n-1)$ *etc.* then appears in the equation (right hand side), $y(n)$ on left hand side.

- Subproblem — break problem into smaller Signal flow graph components. **Useful for larger problems**

Simple Feedback Loop Example

(Simple IIR Filter)



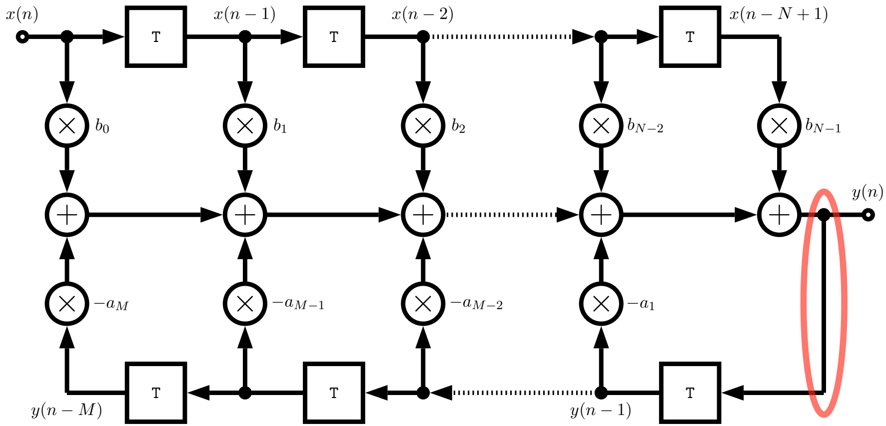
- The algorithm is represented by the difference equation:

$$y(n) = x(n) - a_1 y(n-1) - a_2 y(n-2)$$

- This produces the opposite signal flow graph

More Complex Feedback Loop Example

(General IIR Filter)



We can represent the IIR system algorithm by the difference equation:

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

Signal Flow Graph Problem Decomposition

(Shelving Filter)

$$
\begin{aligned}
y_1(n) &= a_{B/C}x(n) + x(n-1) - a_{B/C}y_1(n-1) \\
y(n) &= \frac{H_0}{2}(x(n) \pm y_1(n)) + x(n)
\end{aligned}
$$

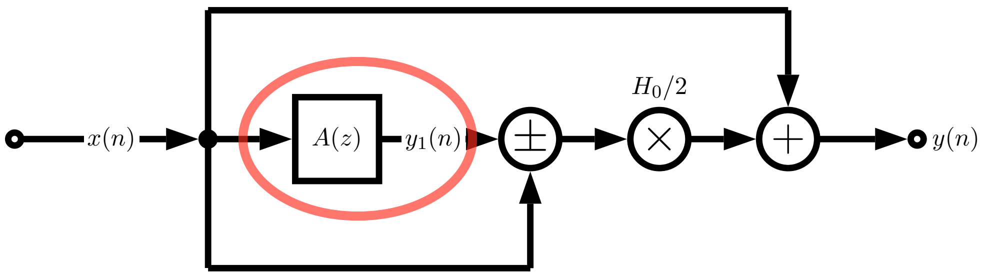
The gain, G, in dB can be adjusted accordingly:

$$H_0 = V_0 - 1 \;\text{ where } V_0 = 10^{G/20}$$

and the cut-off frequency for **boost**, a_B, or **cut**, a_C are given by:

$$
\begin{aligned}
a_B &= \frac{tan(2\pi f_c/f_s) - 1}{tan(2\pi f_c/f_s) + 1} \\
a_C &= \frac{tan(2\pi f_c/f_s) - V_0}{tan(2\pi f_c/f_s) - V_0}
\end{aligned}
$$

Shelving Filters Signal Flow Graph



where $A(z)$ is given by: