CM2208: Scientific Computing
2. Digital Signal Processing
2.3. Filters and Their Applications

Prof. David Marshall

School of Computer Science & Informatics

Filtering

Filtering

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

Filtering Examples:

- 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.
  - **JPEG Image** Compression
  - **MPEG Video** Compression
  - **MPEG Audio** Compression
- In **Digital Audio** we may wish to determine a range of frequencies we wish the enhance or diminish to equalise the signal, *e.g.*:
  - **Tone** — Treble and Bass — **Controls** (**Example coming soon**)
  - **Graphic Equaliser**

How can we filter a Digital Signal

Two Ways to Filter

- Temporal Domain — *E.g.* Sampled (PCM) Audio
- Frequency Domain — Analyse frequency components in signal

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

Temporal Domain Filters

We will look at:

IIR Systems : Infinite impulse response systems

FIR Systems : Finite impulse response systems

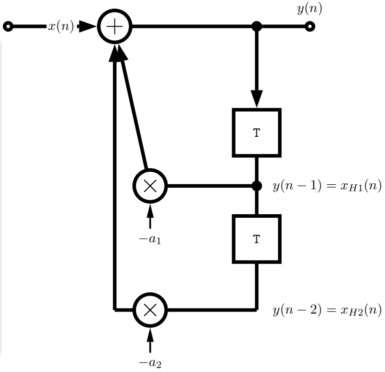
Infinite Impulse Response (IIR) Systems

Simple Example 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**

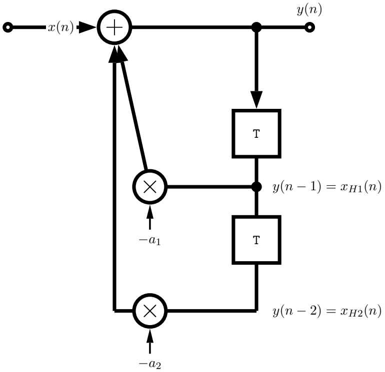


Infinite Impulse Response (IIR)Systems Explained

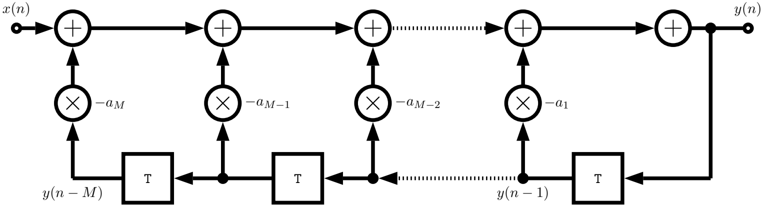
IIR Filter Explained

The following happens:

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



A Complete IIR System



Complete IIR Algorithm

Here we extend:

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

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

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

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

Finite Impulse Response (FIR) Systems

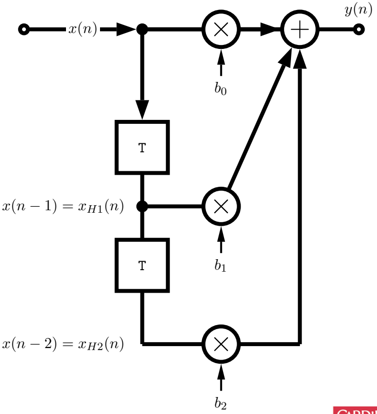
FIR system's are slightly simpler — there is **no feedback loop**.

Simple Example FIR Filter

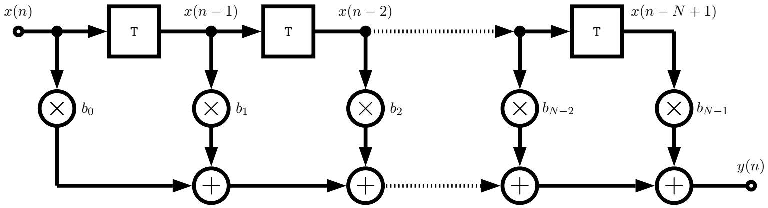
A simple FIR system can be described as follows:

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

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



A Complete FIR System



FIR Algorithm

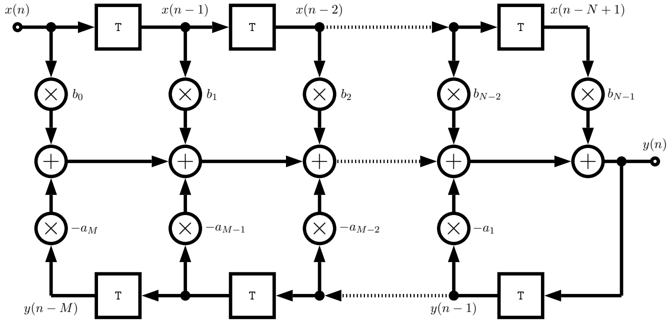
To develop a more complete FIR system we need to add $N-1$ **feed forward delays**

We can describe this with the algorithm:

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

A Complete IIR/FIR System

Combine **IIR** and **FIR** into **one system**:



Complete IIR/FIR System Algorithm

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

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

Filtering with IIR/FIR

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:

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

- If a(1) is **not equal** to **1**, filter **normalizes** the filter coefficients by a(1). If **a(1) equals 0**, `filter()` **returns** an **error**

Creating Filters

How do I create Filter banks `A` and `B`

- Filter banks can be created manually — Hand Created: **See next slide** and **Equalisation** example later in slides
- MATLAB can provide some predefined filters — **a few slides on, see lab classes**
  - Many standard filters provided by MATLAB
- See also `help filter`, online MATLAB `docs` and lab classes.

Filtering with IIR/FIR: Simple Example

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

IIRdemo.m

```
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);
```

Apply this filter

How to apply the (previous) difference equation:

- By hand

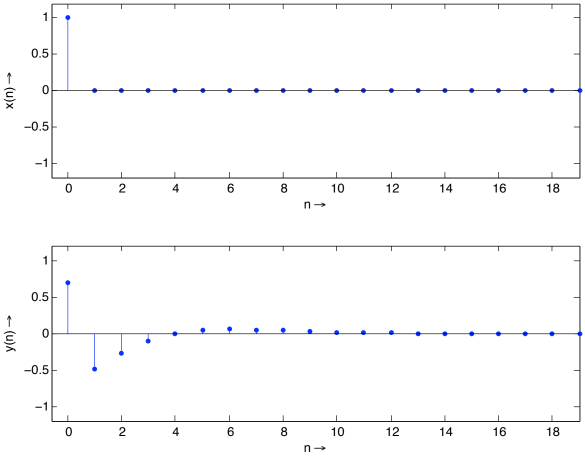
IIRdemo.m Cont.

```
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;
```

- Use MATLAB `filter()` function — **see next but one slide**
  - Far more **preferable**: general — **any length filter**

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 MATLAB to make filters for `filter()` (1)

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

Some common MATLAB Filter Bank Creation Functions

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

See `help` or `doc` appropriate function.

Using MATLAB to make filters for `filter()`(2)

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].
```

Note: We will study the **Butterworth** filter in **more detail** later

Two Examples of Filtering

Application of Filtering

There are numerous examples of Filtering in DSP:

- Noise Removal
- Signal Analysis
- Audio Synthesis
- Audio Effects
- Many more ....

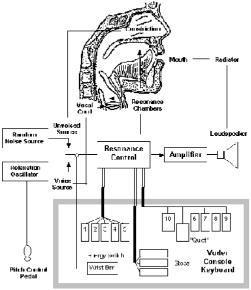
Two Examples

- **Subtractive Synthesis**
- **Equalisation** — Tone control

Subtractive Synthesis

Basic Idea: **Subtractive synthesis** is a method of **subtracting overtones** from a **sound** via by the application of a **filter**.

- First Example: Vocoder — talking robot (1939).
- Popularised with Moog Synthesisers 1960-1970s



Subtractive Synthesis: A Human Example

Human Filtering — cf. Wah-Wah Effect

We can model how humans make utterances as subtractive synthesis: (e.g. Vocoder)

Oscillator — the vocal cords act as the sound source and

Filter — the mouth and throat modify the sound.

- **A sweeping filter — vary (modulate) the filter frequency**

- Make a “ooh” and “aah” sound — same pitch
- By gradually changing from “ooh” to “aah” and back again – simulate the sweeping filter effect
- Effect widely used in electronic music/synthesis
- Basis of the **wah-wah guitar effect**, so named for obvious reasons.

Subtractive Synthesis: One More Human Example

Making Aeroplane, Wind and Ocean Wave Noises

Make a "ssh" sound — white noise

- Now "synthesise" a "jet plane landing" sound
- Vary mouth shape to filter the white noise into pink noise by removing the higher frequencies.
- The same technique (filtered white noise) can be used to electronically synthesise the sound of ocean waves and wind,
- Used in early drum machines to create snare drum and other percussion sounds.

Using MATLAB Filter. Example 1: Subtractive Synthesis Lecture (1)

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

subtract_synth.m

```
% simple low pass 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);
```

Example 1: Subtractive Synthesis Lecture (2)

subtract_synth.m Cont.

```
% 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 (1)

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

synth.m

```
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);
```

synth.m (2)

synth.m Cont.

```
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
```

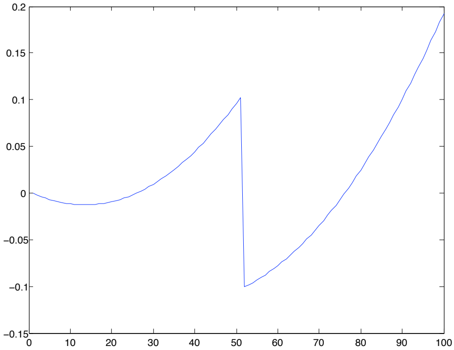
synth.m Explained

Note: the *'sawtooth'* waveform has a non-linear upslope:

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.



Basic Digital Audio Filtering Effects: Equalisers

Filters

Filters by definition **remove/attenuate** audio from the spectrum above or below some cut-off frequency.

- For many audio applications this a little too restrictive

Equalisers

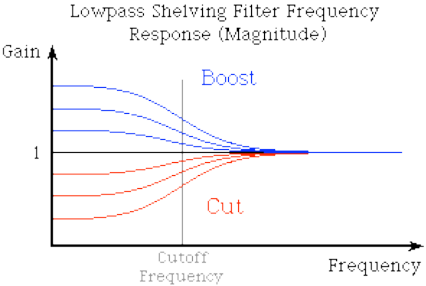
Equalisers, by contrast, **enhance/diminish** certain frequency bands whilst leaving others **unchanged**:

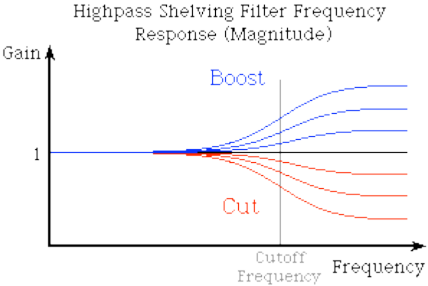
- Built using a series of *shelving* and *peak* filters
- First or second-order filters usually employed.

Shelving and Peak Filters (1)

Two Special Classes of Filters:

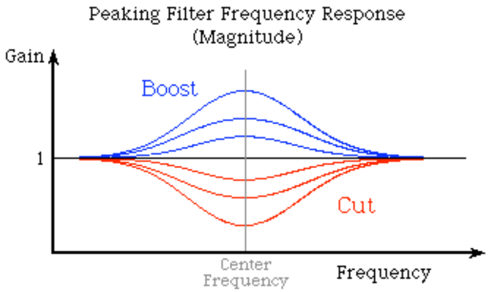
Shelving Filter/Equaliser — Boost or cut the low or high frequency bands with a cut-off frequency, F_c and gain G





Shelving and Peak Filters (2)

Peak Filter/Equaliser — Boost or cut mid-frequency bands with a cut-off frequency, F_c, a bandwidth, f_b and gain G



How can we make a Peak Filter from Shelving Filter (or Two)?

Shelving Filters (1)

First-order Shelving Filter

A **first-order** **shelving filter** may be described by following algorithm/difference equation:

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

where

- **Lowpass Filter**/**Highpass Filter** $= +/-$
- **B** =**Boost**, **C** =**Cut**

Shelving Filters (2)

Tuning Parameter

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

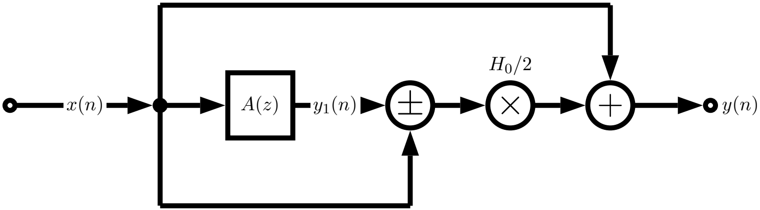
$$\mathbf{H_0 = V_0 - 1} \quad \text{where } \mathbf{V_0 = 10^{G/20}}$$

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

$$a_B = \frac{tan(2\pi f_c/f_s) - 1}{tan(2\pi f_c/f_s) + 1}$$

$$a_C = \frac{tan(2\pi f_c/f_s) - V_0}{tan(2\pi f_c/f_s) - V_0}$$

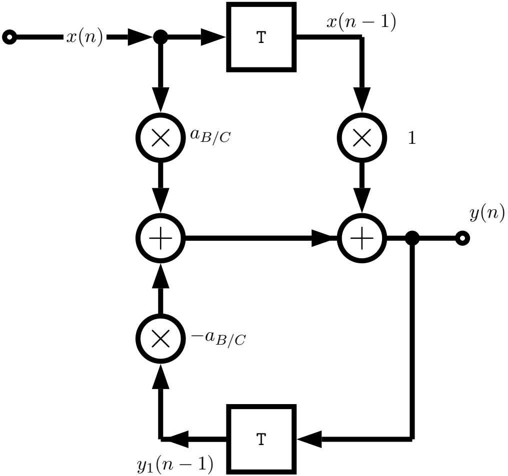
Shelving Filters Signal Flow Graph (1)



where $A(z)$ is an **Allpass Filter**.

Shelving Filters Signal Flow Graph (2)

$A(z)$ is given by:



Peak Filters

Second-order Peak Filter

A **second-order** **peak filter** may be described by following algorithm/difference equation:

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

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

Peak Filters (2)

Tuning Parameters

The center/cut-off frequency, d, is given by:

$$d = -cos(2\pi f_c/f_s)$$

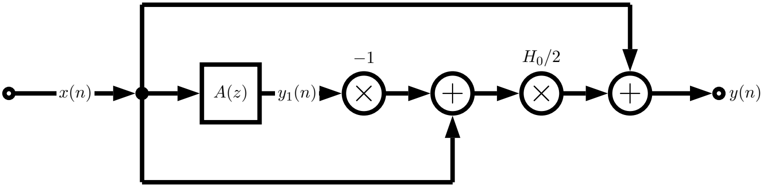
The H_0 by relation to the gain, G, as before:

$$H_0 = V_0 - 1 \quad \textbf{where} \quad V_0 = 10^{G/20}$$

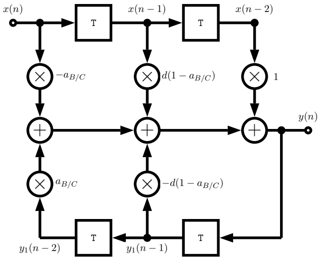
and the bandwidth, f_b is given by the limits for **boost**, a_B, or **cut**, a_C are given by:

$$tan(2\pi f_b/f_s) - 1$$

Peak Filters Signal Flow Graph



where $A(z)$ is given by:



Shelving Filter EQ MATLAB Example (1)

The following function, shelving.m performs a shelving filter:

shelving.m

```
function [b, a]  = shelving(G, fc, fs, Q, type)
%
% Derive coefficients for a shelving filter with a given amplitude
% and cutoff frequency.  All coefficients are calculated as
% described in Zolzer's DAFX book (p. 50 -55).
%
% Usage:      [B,A] = shelving(G, Fc, Fs, Q, type);
%
%             G is the logrithmic gain (in dB)
%             FC is the center frequency
%             Fs is the sampling rate
%             Q adjusts the slope be replacing the sqrt(2) term
%             type is a character string defining filter type
%             Choices are: 'Base_Shelf' or 'Treble_Shelf'


%Error Check
if((strcmp(type,'Base_Shelf') ~= 1) && ...
        (strcmp(type,'Treble_Shelf') ~= 1))
    error(['Unsupported Filter Type: ' type]);
end

K = tan((pi * fc)/fs);
V0 = 10^(G/20);
root2 = 1/Q; %sqrt(2)
```

Shelving Filter EQ MATLAB Example (2)

shelving.m cont.

```
%Invert gain if a cut
 if (V0 < 1)
     V0 = 1/V0;
 end

%%%%%%%%%%%%%%%%%%%%
%    BASE BOOST
%%%%%%%%%%%%%%%%%%%%
 if(( G > 0 ) & (strcmp(type,'Base_Shelf')))

     b0 = (1 + sqrt(V0)*root2*K + V0*K^2) / (1 + root2*K + K^2);
     b1 = (2 * (V0*K^2 - 1) ) / (1 + root2*K + K^2);
     b2 = (1 - sqrt(V0)*root2*K + V0*K^2) / (1 + root2*K + K^2);
     a1 =  (2 * (K^2 - 1) ) / (1 + root2*K + K^2);
     a2 =  (1 - root2*K + K^2) / (1 + root2*K + K^2);
```

Shelving Filter EQ MATLAB Example (3)

shelving.m cont.

```
%%%%%%%%%%%%%%%%%%%%
%     BASE CUT
%%%%%%%%%%%%%%%%%%%%
elseif (( G < 0 ) & (strcmp(type,'Base_Shelf')))

    b0 = (1 + root2*K + K^2) / (1 + root2*sqrt(V0)*K + V0*K^2);
    b1 =  (2 * (K^2 - 1) ) / (1 + root2*sqrt(V0)*K + V0*K^2);
    b2 =  (1 - root2*K + K^2) / (1 + root2*sqrt(V0)*K + V0*K^2);
    a1 =  (2 * (V0*K^2 - 1) ) / (1 + root2*sqrt(V0)*K + V0*K^2);
    a2 = (1 - root2*sqrt(V0)*K + V0*K^2) / ...
              (1 + root2*sqrt(V0)*K + V0*K^2);

%%%%%%%%%%%%%%%%%%%%
%    TREBLE BOOST
%%%%%%%%%%%%%%%%%%%%
elseif (( G > 0 ) & (strcmp(type,'Treble_Shelf')))

    b0 = (V0 + root2*sqrt(V0)*K + K^2) / (1 + root2*K + K^2);
    b1 =  (2 * (K^2 - V0) ) / (1 + root2*K + K^2);
    b2 = (V0 - root2*sqrt(V0)*K + K^2) / (1 + root2*K + K^2);
    a1 =  (2 * (K^2 - 1) ) / (1 + root2*K + K^2);
    a2 =  (1 - root2*K + K^2) / (1 + root2*K + K^2);
```

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

Shelving Filter EQ MATLAB Example (4)

shelving.m cont.

```
%%%%%%%%%%%%%%%%%%%%
%    TREBLE CUT
%%%%%%%%%%%%%%%%%%%%

elseif (( G < 0 ) & (strcmp(type,'Treble_Shelf')))

    b0 =   (1 + root2*K + K^2) / (V0 + root2*sqrt(V0)*K + K^2);
    b1 =    (2 * (K^2 - 1) ) / (V0 + root2*sqrt(V0)*K + K^2);
    b2 =    (1 - root2*K + K^2) / (V0 + root2*sqrt(V0)*K + K^2);
    a1 =    (2 * ((K^2)/V0 - 1) ) / (1 + root2/sqrt(V0)*K ...
                + (K^2)/V0);
    a2 = (1 - root2/sqrt(V0)*K + (K^2)/V0) / ....
             (1 + root2/sqrt(V0)*K + (K^2)/V0);

%%%%%%%%%%%%%%%%%%%%
%    All-Pass
%%%%%%%%%%%%%%%%%%%%
else
    b0 = V0;
    b1 = 0; b2 = 0; a1 = 0; a2 = 0;
end

%return values
a = [  1, a1, a2];
b = [ b0, b1, b2];
```

Shelving Filter EQ MATLAB Example (5)

The following script shelving_eg.m illustrates how we use the shelving filter function to filter:

shelving_eg.m

```
infile = 'acoustic.wav';

% read in wav sample
[ x, Fs, N ] = wavread(infile);

%set Parameters for Shelving Filter
% Change these to experiment with filter

G = 4; fcb = 300; Q = 3; type = 'Base_Shelf';

[b a] = shelving(G, fcb, Fs, Q, type);
yb = filter(b,a, x);

% write output wav files
wavwrite(yb, Fs, N, 'out_bassshelf.wav');

% plot the original and equalised waveforms
figure(1), hold on;
plot(yb,'b');
plot(x,'r');
title('Bass Shelf Filter Equalised Signal');
```

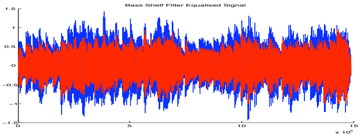
Shelving Filter EQ MATLAB Example (6)

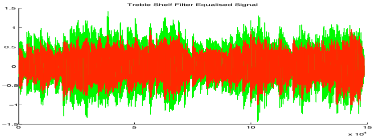
shelving_eg.m cont.

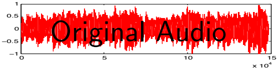
```
%Do treble shelf filter
fct = 600; type = 'Treble_Shelf';

[b a] = shelving(G, fct, Fs, Q, type);
yt = filter(b,a, x);

% write output wav files
wavwrite(yt, Fs, N, 'out_treblehelf.wav');

figure(1), hold on;
plot(yb,'g');
plot(x,'r');
title('Treble Shelf Filter Equalised Signal');
```

Shelving Filter EQ MATLAB Example Output

The output from the above code is (red plot is original audio):







Click on above images or here to hear: original audio, bass shelf filtered audio, treble shelf filtered audio.

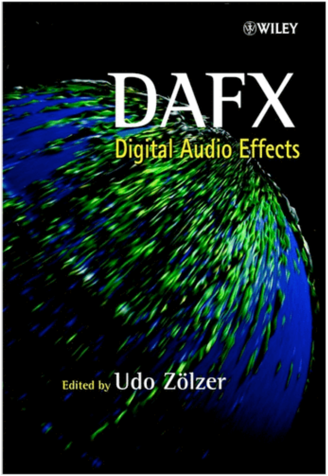
Further Reading

DAFX: Digital Audio Effects
Udo Zolzer
John Wiley and Sons Ltd , 2002
(ISBN-13: 978-0471490784)

Excellent coverage of audio signal processing effects and synthesis plus a lot more

All MATLAB examples

Copies in library



Additional Examples, MATLAB Code and Reading

Audio Synthesis — CM3106 Module Notes

Audio Effects — CM3106 Module Notes

Also (Past) CM0268 Module Notes

See **Lab Class** Exercises