Time-varying Filters

Time-varying Filter Effects

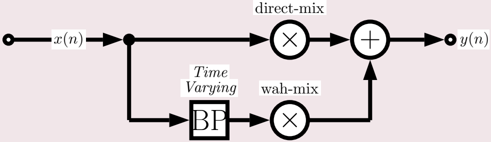
Some common effects are realised by simply time varying a filter in a couple of different ways:

Wah-wah: A bandpass filter with a (**modulated**) time varying centre (resonant) frequency and a small bandwidth. Filtered signal mixed with direct signal.

Phasing: A notch filter, that can be realised as set of cascading IIR filters, again mixed with direct signal.

Wah-wah Example

Wah-wah, Signal flow diagram:



where **BP** is a **time-varying frequency bandpass filter**.

Wah-wah Variations

- A *phaser* is similarly implemented with a **notch filter replacing** the **bandpass filter**.
- A variation is the ***M*-fold wah-wah** filter where M tap delay bandpass filters spread over the entire spectrum change their centre frequencies simultaneously.
  - A **bell effect** can be achieved with around a **hundred *M* tap delays** and **narrow bandwidth filters**

Time Varying Filter Implementation: State Variable Filter

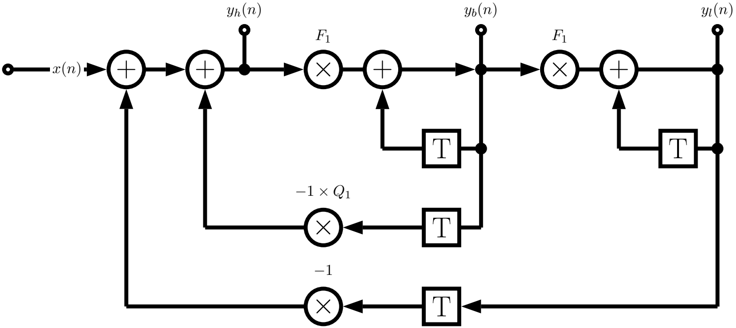
The Practical State Variable Filter

In time varying filters we now want **independent** control over the **cut-off frequency** and **damping factor** of a filter.

(Borrowed from analog electronics) We can implement a **State Variable Filter** to solve this problem.

- One further advantage is that we can **simultaneously** get **lowpass**, **bandpass** and **highpass** filter output.

The State Variable Filter



where:

$$
\begin{aligned}
x(n) &= \textbf{input signal} \\
y_l(n) &= \textbf{lowpass signal} \\
y_b(n) &= \textbf{bandpass signal} \\
y_h(n) &= \textbf{highpass signal}
\end{aligned}
$$

The State Variable Filter Algorithm

State Variable Filter difference equations are given by:

$$
\begin{aligned}
y_l(n) &= F_1 y_b(n) + y_l(n-1) \\
y_b(n) &= F_1 y_h(n) + y_b(n-1) \\
y_h(n) &= x(n) - y_l(n-1) - Q_1 y_b(n-1)
\end{aligned}
$$

with **tuning coefficients** F_1 and Q_1 related to the cut-off frequency, f_c, and damping, d:

$$F_1 = 2\sin(\pi f_c / f_s), \qquad \textbf{and} \quad Q_1 = 2d$$

MATLAB Wah-wah Implementation

Making a Wah-wah

We simply implement the State Variable Filter with a Sinusoid **Modulated** (**variable**) **frequency**, f_c.

wah_wah.m:

```
% wah_wah.m   state variable band pass
%
% BP filter with narrow pass band, Fc oscillates up and
% down the spectrum
% Difference equation taken from DAFX chapter 2
%
% Changing this from a BP to a BR/BS (notch instead of a bandpass)
%  converts this effect to a phaser
%
% yl(n) = F1*yb(n) + yl(n-1)
% yb(n) = F1*yh(n) + yb(n-1)
% yh(n) = x(n) - yl(n-1) - Q1*yb(n-1)
%
% vary Fc from 500 to 5000 Hz
```

Wah-wah Implementation

wah_wah.m (Cont.):

```
infile = 'acoustic.wav';

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

%%%%%%% EFFECT COEFFICIENTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% damping factor
% lower the damping factor the smaller the pass band
damp = 0.05;

% min and max centre cutoff frequency of variable bandpass filter
minf=500;
maxf=3000;

% wah frequency, how many Hz per second are cycled through
Fw = 2000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Wah-wah Implementation

wah_wah.m (Cont.):

```
% change in centre frequency per sample (Hz)
delta = Fw/Fs;

% create triangle wave of centre frequency values
Fc=minf:delta:maxf;
while(length(Fc) < length(x) )
    Fc= [ Fc (maxf:-delta:minf) ];
    Fc= [ Fc (minf:delta:maxf) ];
end

% trim tri wave to size of input
Fc = Fc(1:length(x));

% difference equation coefficients
% must be recalculated each time Fc changes
F1 = 2*sin((pi*Fc(1))/Fs);
% this dictates size of the pass bands
Q1 = 2*damp;
```

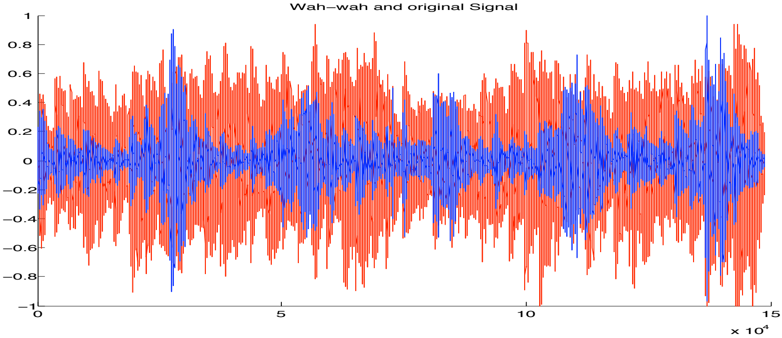
Wah-wah Implementation

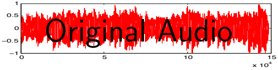
wah_wah.m (Cont.):

```
yh=zeros(size(x));          % create emptly out vectors
yb=zeros(size(x));
yl=zeros(size(x));

% first sample, to avoid referencing of negative signals
yh(1) = x(1);
yb(1) = F1*yh(1);
yl(1) = F1*yb(1);

% apply difference equation to the sample
for n=2:length(x),
    yh(n) = x(n) - yl(n-1) - Q1*yb(n-1);
    yb(n) = F1*yh(n) + yb(n-1);
    yl(n) = F1*yb(n) + yl(n-1);
    F1 = 2*sin((pi*Fc(n))/Fs);
end

% normalise  and  Output .........
```

Wah-wah MATLAB Example (Cont.)

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





Click on images or here to hear: original audio, wah-wah audio.