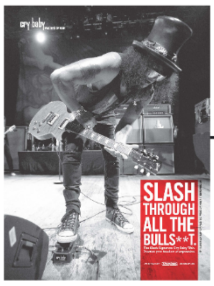
WAH WAH

INTRODUCTION

The Wah Wah effect has been used in popular music since the 1960s. The most notable of uses is in a music performance context in the form of a solid state pedal for electric guitar. Operated with the foot, guitarist are able to manipulate the effect's parameters whilst leaving their hands free to play their instrument. In their sturdy form-factor, wah wah pedals are not just limited to use by guitar players; any amplified-acoustic or electric instruments as can be run through a wah wah pedal and played in real time.

The crux of the wah wah effect is modulation of a passband (illustrated in FIG2). Performers are able to create a musical effect by emphasising and accenting notes by controlling the depth and rate of this modulation along with what they play on their instrument. This is achieved by 'sweeping' a bandpass filter with a narrow Q up and down the frequency spectrum, usually between a defined upper and lower frequency.



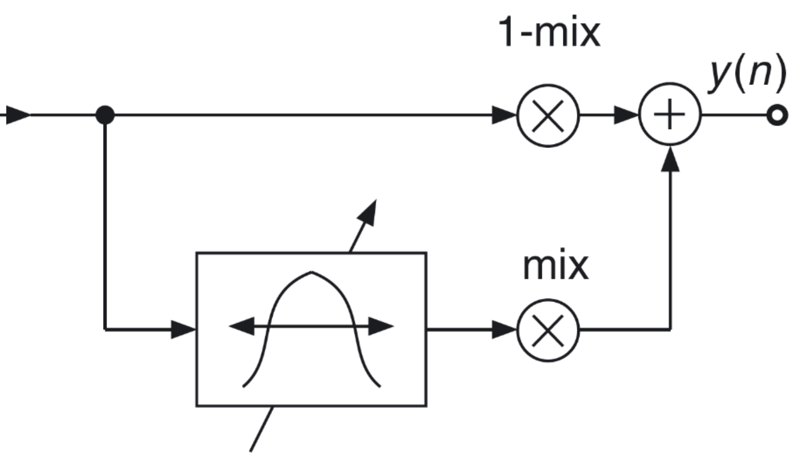


FIG 2. Diagram representation of the basic wah wah effect

In the basic block diagram above, the guitar input (often band limited) is filtered by the passband and summed back with the original dry signal. This same effect is what is achieved by mouthing the phoneme 'wah' 'wah' wah' except that, in this case, all that is heard is the 'wet' signal; completely filtered by the mouth's narrowing and widening aperture acting like a passband filter.

FORMULAE AND CODE

The wah wah effect can be implemented in Matlab in the following steps.
After accepting an audio input, the user is prompted to enter values for the filters coefficients & the entire frequency is set to match the sampling frequency depending on
the width of the passband set by the value returned to the variable ‘`width`’.

```
infile = 'Lincoln.wav';
% Read in Audio File
[ audio_in, fs] = audioread(infile);

%Prompt for and store 'Damping', 'Min_Cutoff' and 'Max_Cutoff' and 'Width'
Variables
damping = input('\n What Damping Factor would you like to
use?\n(Recommended value 0.05)\n>');
width = input('\n What Rate would you like to use? (Hz)\n(Recommended value
1000)\n>');
min_cutoff = input('\n What minimum Cut-Off Frequency would you like to
use? (Hz)\n(Recommended value 250)\n>');
max_cutoff = input('\n What maximum Cut-Off Frequency would you like to
use? (Hz)\n(Recommended value 5000)\n>');
```

```
centre_freq = width/fs;
```

To make this sweep linear and its change in direction instantaneous, a triangle wave is used for modulation. Along with some additional filtering coefficients, this can be implemented in code in the following manner:

```
cutoff_freq=min_cutoff:center_freq:max_cutoff;
while(length(cutoff_freq) < length(audio_in) )
cutoff_freq = [ cutoff_freq (max_cutoff:-center_freq:min_cutoff) ];
cutoff_freq = [ cutoff_freq (min_cutoff:center_freq:max_cutoff) ];
end
```

The modulated signal must be the combination of two equally long files so the modulation signal and audio input are adjusted in this line:

```
cutoff_freq = cutoff_freq(1:length(audio_in));
```

The Filtering coefficients F1 and Q1 are used to control the centre frequency and Q:

```
F1 = 2*sin((pi*cutoff_freq(1))/fs);
Q1 = 2*damping;
```

The state variable filter derived by H. Chamberlin for the bandpass.

```
% Create and Zero Vectors to Match Length of Audio Input File
highpass=zeros(size(audio_in));
bandpass=zeros(size(audio_in));
lowpass=zeros(size(audio_in));
```

To avoid multiplying by zero errors, the calculation is commenced from the first sample:

```
highpass(1) = audio_in(1);
bandpass(1) = F1*highpass(1);
lowpass(1) = F1*bandpass(1);
```

Followed by the differential equation:

```
for n=2:length(audio_in),
highpass(n) = audio_in(n) - lowpass(n-1) - Q1*bandpass(n-1);
bandpass(n) = F1*highpass(n) + bandpass(n-1);
lowpass(n) = F1*bandpass(n) + lowpass(n-1);
F1 = 2*sin((pi*cutoff_freq(n))/fs);
end
```

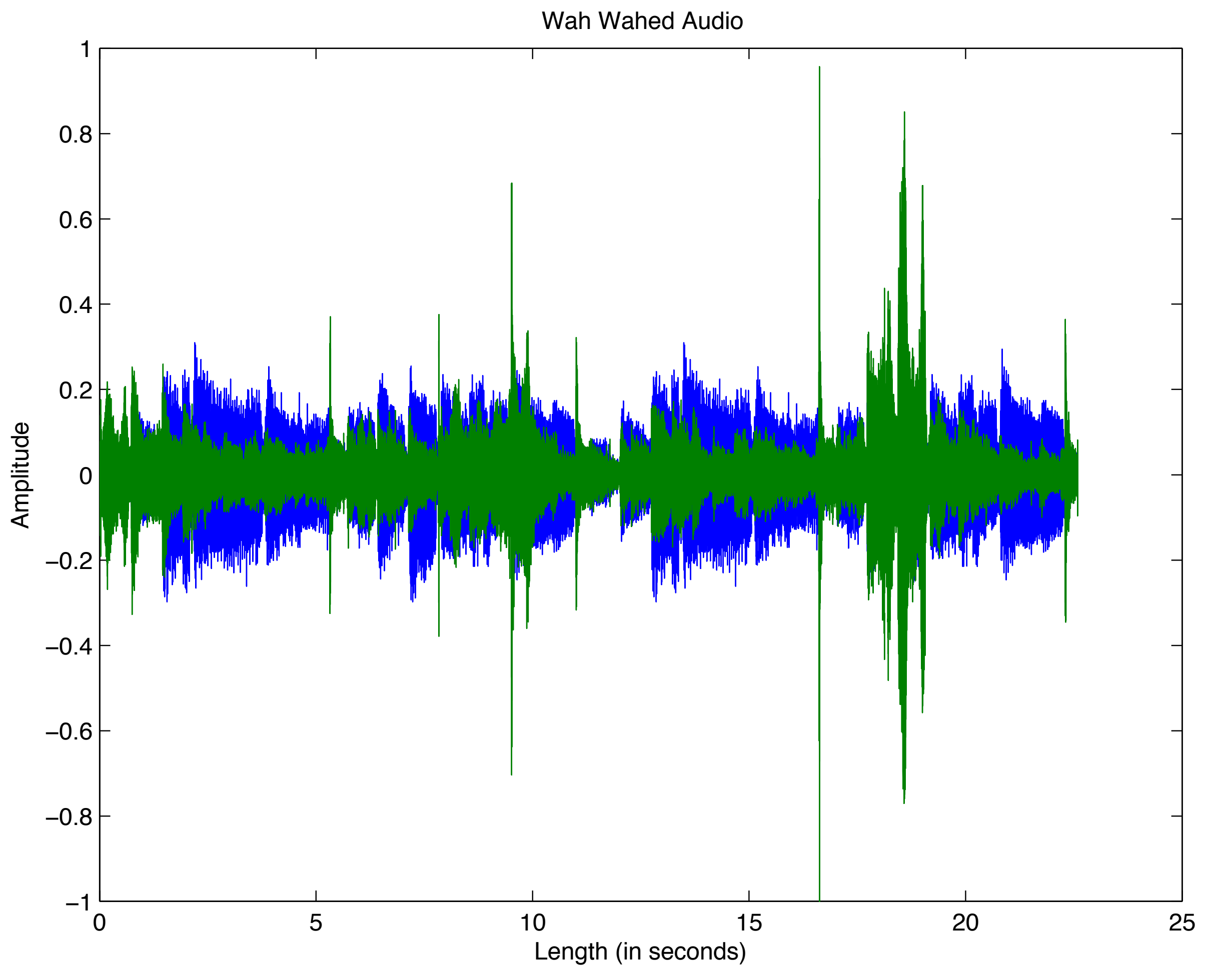
Audio is normalised and played back

```
normed = bandpass./max(max(abs(bandpass)));
audiowrite('wah wahed.wav', normed, fs);
sound (normed, fs);
```

The audio is plotted

```
%%
%Plots
t1=0:1/fs:(length(audio_in)-1)/fs;

figure(1);
plot(t1,audio_in);
title('Original Audio');
ylabel('Amplitude');
xlabel('Length (in seconds)');

hold all;

t2=0:1/fs:(length(normed)-1)/fs;

figure(1);
plot(t2,normed);
title('Wah Wahed Audio');
ylabel('Amplitude');
xlabel('Length (in seconds)');

n=length(audio_in)-1;
f1=0:fs/n:fs;
audio_infft=abs(fft(audio_in));

figure(2);
plot(f1,audio_infft);
xlabel('Frequency in Hz');
ylabel('Magnitude');
title('The Audio Input FFT');

hold all;

n=length(normed)-1;
f1=0:fs/n:fs;
normedfft=abs(fft(audio_in));

figure(2);
plot(f1,normedfft);
xlabel('Frequency in Hz');
ylabel('Magnitude');
title('Wah Wahed FFT');
```

The Plot below of the original signal blue and the filtered signal green.



The intention of the original audio effect was to utilise real-time web cam input by calling a modified version of the webcam.m function written by Jeromy Anglim (www.matlabtips.com) however, due to constraints this was abandoned.

The code block for calling the function is as follows:

```
%%
% %Run Webcam Script for input Minimum and Maximum Cut-off frequencies!!
% disp ('Running Web Cam for Wah Wah Input\n close the window when you have
finished');
% run webcam.m;
%
% load OldValues.mat
%
% %Set Min & Max Cut Off frequencies to match sample rate??
% min_cutoff = 0:1/fs:(length(audio_in)-1)/fs;
% max_cutoff = 0:1/fs:(length(audio_in)-1)/fs;

%%
```

References:

Zolzer, Udo, 2002, DAFX: Digital Audio Effects, John Wiley & Sons

Chamberlin, Hal, 1980, Musical Applications of Microprocessors, Hayden Book Co.