Digital Audio Effects

Having learned to make basic sounds from basic waveforms and more advanced synthesis methods lets see how we can add some digital audio effects. These may be applied:

- As part of the audio creation/synthesis stage — to be subsequently filtered, (re)synthesised
- At the end of the *audio chain* — as part of the production/mastering phase.
- Effects can be applied in different orders and sometimes in a *parallel* audio chain.
- The order of applying the same effects can have drastic differences in the output audio.
- Selection of effects and the ordering is a matter for the sound you wish to create. There is no absolute rule for the ordering.

Typical Guitar (and other) Effects Pipeline

Some ordering is *standard* for some audio processing, *E.g*: Compression → Distortion → EQ → Noise Redux → Amp Sim → Modulation → Delay → Reverb

Common for guitar effects pedal.

Effect modules

COMP/EFX	DRIVE	EQ	ZNR	AMP	MODULATION	DELAY	REVERB
Compressor	FD Clean		ZNR	AMP Sim.	Chorus	Delay	Hall
Auto Wah	VX Clean				Ensemble	Tape Echo	Room
Booster	HW Clean				Flanger	Analog Delay	Spring
Tremolo	US Blues				Step	Ping Pong Delay	Arena
Phaser	BG Crunch				Pitch Shift		Tiled Room
⋮	⋮				⋮		

Effect types

Classifying Effects

Audio effects can be classified by the way do their processing:

Basic Filtering — Lowpass, Highpass filter etc,, Equaliser: see CM2202 Notes

Time Varying Filters — **Wah-wah, Phaser**

Delays — **Vibrato, Flanger, Chorus, Echo**

Modulators — **Ring modulation, Tremolo, Vibrato**

Non-linear Processing — Compression, Limiters, **Distortion**, Exciters/Enhancers

Spacial Effects — Panning, Reverb (see CM2202 Notes), Surround Sound

Some of the above **studied here**.[3]

[3] See these notes for further information. This additional information may be **useful for Coursework** but is **NOT examinable**

Time-varying Filters

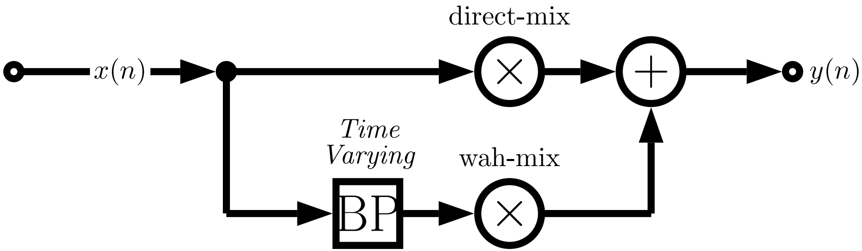
Some common effects are realised by simply time varying a filter in a couple of different ways:

Wah-wah — A bandpass filter with a 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

The signal flow for a wah-wah is as follows:



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

- 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

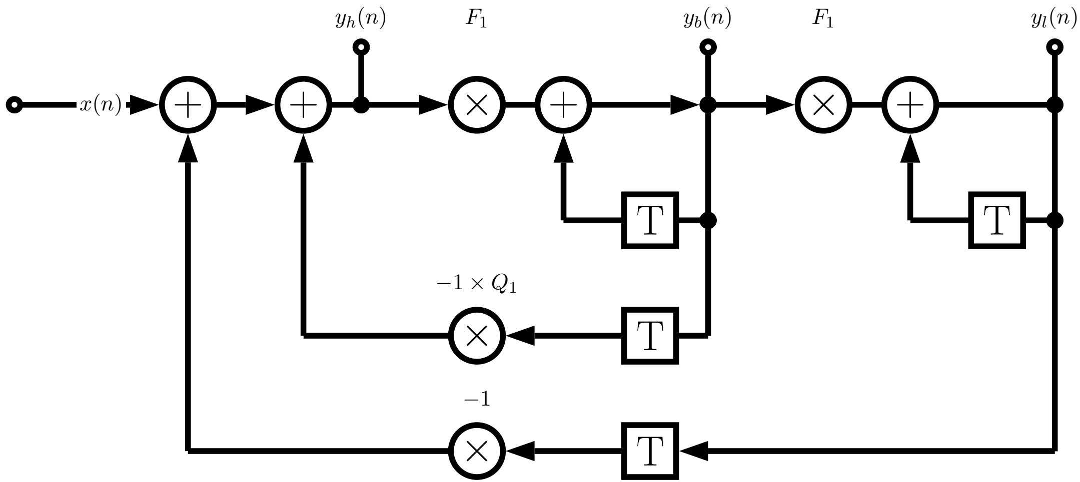
Time Varying Filter Implementation: State Variable Filter

In our audio application of 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:

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

The State Variable Filter Algorithm

The algorithm 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:

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

MATLAB Wah-wah Implementation

We simply implement the State Variable Filter with a variable frequency, f_c. The code listing is 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

infile = 'acoustic.wav';

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


% 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));
```

Back
Close

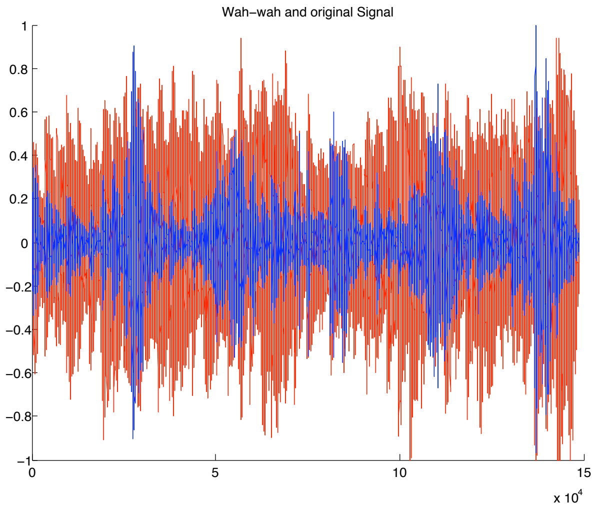
```
% difference equation coefficients
% must be recalculated each time Fc changes
F1 = 2*sin((pi*Fc(1))/Fs);
% this dictates size of the pass bands
Q1 = 2*damp;

yh=zeros(size(x));              % create empty 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
maxyb = max(abs(yb));
yb = yb/maxyb;
```

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

figure(1)
hold on
plot(x,'r');
plot(yb,'b');
title('Wah-wah and original Signal');
```


Back
Close

Wah-wah MATLAB Example (Cont.)

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



Click here to hear: original audio, wah-wah filtered audio.

Delay Based Effects

Many useful audio effects can be implemented using a delay structure:

- Sounds reflected of walls
  - In a cave or large room we here an echo and also reverberation takes place – this is a different effect — **see later**
  - If walls are closer together repeated reflections can appear as parallel boundaries and we hear a modification of sound colour instead.
- Vibrato, Flanging, Chorus and Echo are examples of delay effects

Basic Delay Structure

We build basic delay structures out of some very basic FIR and IIR filters:

- We use *FIR* and *IIR comb filters*
- Combination of FIR and IIR gives the **Universal Comb Filter**

FIR Comb Filter

This simulates a single delay:

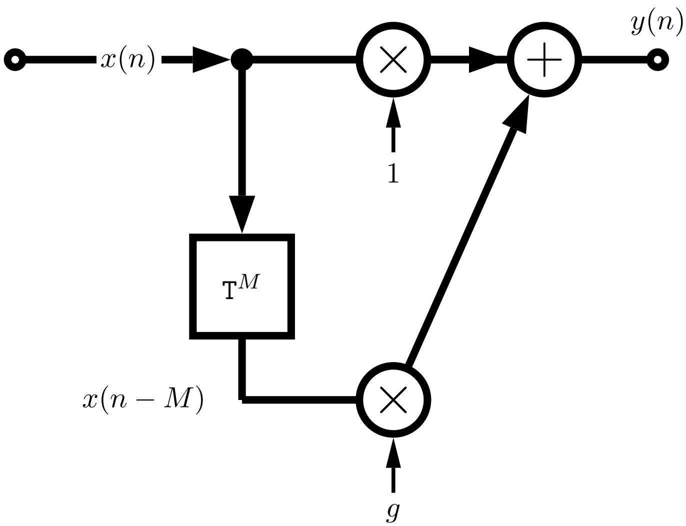
- The input signal is delayed by a given time duration, τ.
- The delayed (processed) signal is added to the input signal some amplitude gain, g
- The difference equation is simply:

$$y(n) = x(n) + gx(n - M) \quad \text{with } M = \tau / f_s$$

- The transfer function is:

$$H(z) = 1 + gz^{-M}$$

FIR Comb Filter Signal Flow Diagram



FIR Comb Filter MATLAB Code

fircomb.m:

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100

g=0.5; %Example gain

Delayline=zeros(10,1); % memory allocation for length 10

for n=1:length(x);
  y(n)=x(n)+g*Delayline(10);
  Delayline=[x(n);Delayline(1:10-1)];
end;
```

IIR Comb Filter

This simulates a single delay:

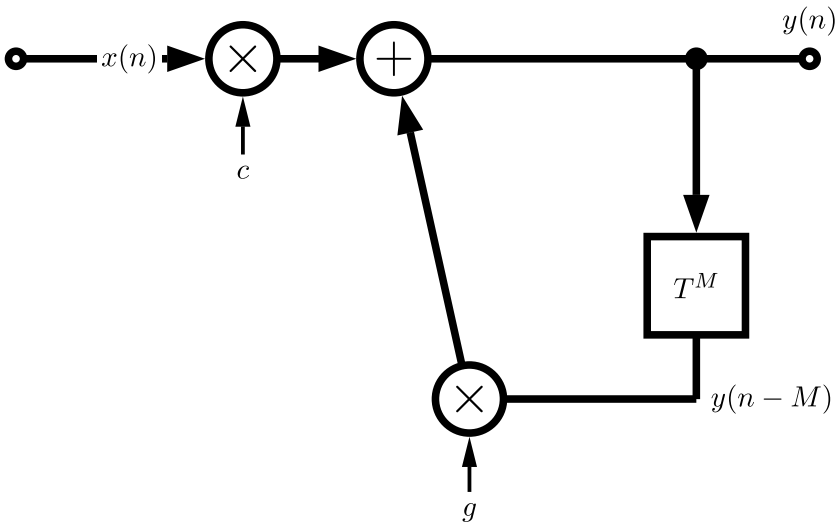
- Simulates *endless reflections* at both ends of cylinder.
- We get an endless series of responses, $y(n)$ to input, $x(n)$.
- The input signal circulates in delay line (delay time τ) that is fed back to the input..
- Each time it is fed back it is attenuated by g.
- Input sometime scaled by c to **compensate** for high amplification of the structure.
- The difference equation is simply:

$$y(n) = Cx(n) + gy(n - M) \quad \text{with} \;\; M = \tau / f_s$$

- The transfer function is:

$$H(z) = \frac{c}{1 - gz^{-M}}$$

IIR Comb Filter Signal Flow Diagram



IIR Comb Filter MATLAB Code

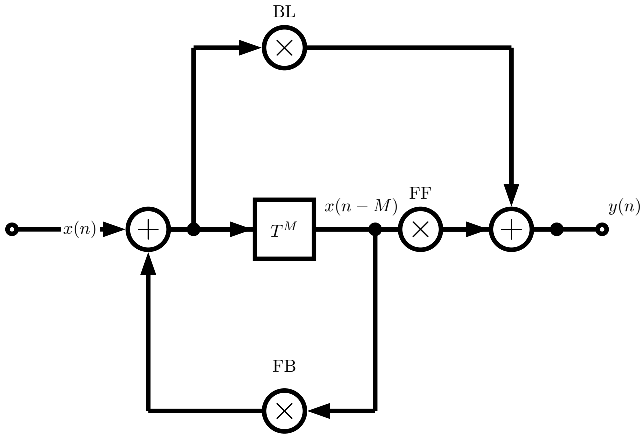
iircomb.m:

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100

g=0.5;

Delayline=zeros(10,1); % memory allocation for length 10

for n=1:length(x);
   y(n)=x(n)+g*Delayline(10);
   Delayline=[y(n);Delayline(1:10-1)];
end;
```

Universal Comb Filter

The combination of the FIR and IIR comb filters yields the **Universal Comb Filter**:

- Basically this is an **allpass filter** with an M sample delay operator and an additional multiplier, FF.



- Parameters: FF = feedforward, FB = feedbackward, BL = blend

Universal Comb Filter Parameters

Universal in that we can form any comb filter, an allpass or a delay:

	BL	FB	FF
FIR Comb	1	0	g
IIR Comb	1	g	0
Allpass	a	$-a$	1
delay	0	0	1

Universal Comb Filter MATLAB Code

unicomb.m:

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100

BL=0.5;
FB=-0.5;
FF=1;
M=10;

Delayline=zeros(M,1); % memory allocation for length 10

for n=1:length(x);
  xh=x(n)+FB*Delayline(M);
  y(n)=FF*Delayline(M)+BL*xh;
  Delayline=[xh;Delayline(1:M-1)];
end;
```

Vibrato - A Simple Delay Based Effect

- **Vibrato** — Varying the time delay periodically
- If we vary the distance between and observer and a sound source (*cf. Doppler effect*) we here a change in pitch.
- Implementation: A Delay line and a low frequency oscillator (LFO) to vary the delay.
- Only listen to the delay — no forward or backward feed.
- Typical delay time = 5–10 Ms and LFO rate 5–14Hz.

Vibrato MATLAB Code

vibrato.m function, Use vibrato_eg.m to call function:

```
function y=vibrato(x,SAMPLERATE,Modfreq,Width)

ya_alt=0;
Delay=Width; % basic delay of input sample in sec
DELAY=round(Delay*SAMPLERATE); % basic delay in # samples
WIDTH=round(Width*SAMPLERATE); % modulation width in # samples
if WIDTH>DELAY
  error('delay greater than basic delay !!!');
  return;
end;

MODFREQ=Modfreq/SAMPLERATE; % modulation frequency in # samples
LEN=length(x);         % # of samples in WAV-file
L=2+DELAY+WIDTH*2;     % length of the entire delay
Delayline=zeros(L,1); % memory allocation for delay
y=zeros(size(x));      % memory allocation for output vector
```

```
for n=1:(LEN-1)
  M=MODFREQ;
  MOD=sin(M*2*pi*n);
  ZEIGER=1+DELAY+WIDTH*MOD;
  i=floor(ZEIGER);
  frac=ZEIGER-i;
  Delayline=[x(n);Delayline(1:L-1)];
  %---Linear Interpolation-----------------------------
  y(n,1)=Delayline(i+1)*frac+Delayline(i)*(1-frac);
  %---Allpass Interpolation------------------------------
  %y(n,1)=(Delayline(i+1)+(1-frac)*Delayline(i)-(1-frac)*ya_alt);
  %ya_alt=ya(n,1);
end
```

Vibrato MATLAB Example (Cont.)

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

Click here to hear: original audio, vibrato audio.

Comb Filter Delay Effects: Flanger, Chorus, Slapback, Echo

- A few popular effects can be made with a comb filter (FIR or IIR) and some modulation
- Flanger, Chorus, Slapback, Echo same basic approach but *different sound* outputs:

Effect	Delay Range (ms)	Modulation
Resonator	$0 \ldots 20$	None
Flanger	$0 \ldots 15$	Sinusoidal ($\approx$ 1 Hz)
Chorus	$10 \ldots 25$	Random
Slapback	$25 \ldots 50$	None
Echo	> 50	None

- Slapback (or doubling) — quick repetition of the sound,
  Flanging — continuously varying LFO of delay,
  Chorus — **multiple copies** of sound delayed by small random delays

Flanger MATLAB Code

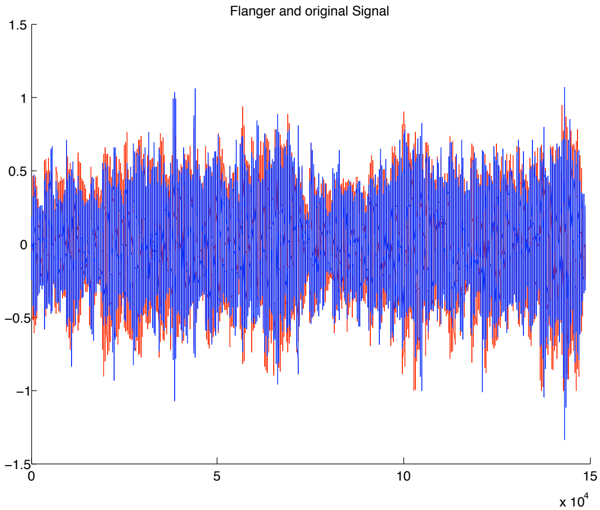
flanger.m:

```
% Creates a single FIR delay  with the delay time oscillating from
%  Either 0-3 ms or 0-15 ms at 0.1 - 5 Hz

 infile='acoustic.wav';
outfile='out_flanger.wav';

% read the sample waveform
[x,Fs,bits] = wavread(infile);

% parameters to vary the effect %
max_time_delay=0.003; % 3ms max delay in seconds
rate=1; %rate of flange in Hz

index=1:length(x);

% sin reference to create oscillating delay
sin_ref = (sin(2*pi*index*(rate/Fs)))';

%convert delay in ms to max delay in samples
max_samp_delay=round(max_time_delay*Fs);

% create empty out vector
y = zeros(length(x),1);
```

```
% to avoid referencing of negative samples
y(1:max_samp_delay)=x(1:max_samp_delay);

% set amp suggested coefficient from page 71 DAFX
amp=0.7;

% for each sample
for i = (max_samp_delay+1):length(x),
  cur_sin=abs(sin_ref(i));    %abs of current sin val 0-1
  % generate delay from 1-max_samp_delay and ensure whole number
  cur_delay=ceil(cur_sin*max_samp_delay);
  % add delayed sample
  y(i) = (amp*x(i)) + amp*(x(i-cur_delay));
end

% write output
wavwrite(y,Fs,outfile);
```

Flanger MATLAB Example (Cont.)

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



Click here to hear: original audio, flanged audio.

Modulation

Modulation is the process where parameters of a sinusoidal signal (amplitude, frequency and phase) are modified or varied by an audio signal.

We have met some example effects s that could be considered as a class of modulation already:

Amplitude Modulation — Wah-wah, Phaser

Frequency Modulation — Audio synthesis technique (**Already Studied**)

Phase Modulation — Vibrato, Chorus, Flanger

We will now introduce some Modulation effects.

Ring Modulation

Ring modulation (RM) is where the audio *modulator* signal, $x(n)$ is multiplied by a sine wave, $m(n)$, with a *carrier* frequency, f_c.

- This is very simple to implement digitally:

$$y(n) = x(n).m(n)$$

- Although audible result is easy to comprehend for simple signals things get more complicated for signals having numerous partials
- If the modulator is also a sine wave with frequency, f_x then one hears the sum and difference frequencies: $f_c + f_x$ and $f_c - f_x$, for example.
- When the input is *periodic* with at a fundamental frequency, f_0, then a spectrum with amplitude lines at frequencies $|kf_0 \pm f_c|$
- Used to create robotic speech effects on old sci-fi movies and can create some odd almost non-musical effects if not used with care. (Original speech)

MATLAB Ring Modulation

Two examples, a sine wave and an audio sample being modulated by a sine wave, ring_mod.m

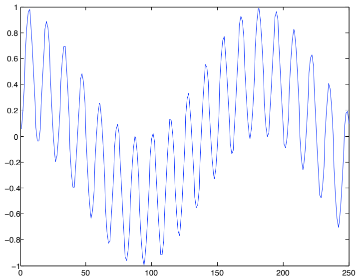
```
filename='acoustic.wav';

% read the sample waveform
[x,Fs,bits] = wavread(filename);

index = 1:length(x);

% Ring Modulate with a sine wave frequency Fc
Fc = 440;
carrier= sin(2*pi*index*(Fc/Fs))';

% Do Ring Modulation
y = x.*carrier;

% write output
wavwrite(y,Fs,bits,'out_ringmod.wav');
```

Click here to hear: original audio, ring modulated audio.

MATLAB Ring Modulation: Two sine waves

```
% Ring Modulate with a sine wave frequency Fc
Fc = 440;
carrier= sin(2*pi*index*(Fc/Fs))';

%create a modulator sine wave frequency Fx
Fx = 200;
modulator = sin(2*pi*index*(Fx/Fs))';

% Ring Modulate with sine wave, freq. Fc
y = modulator.*carrier;

% write output
wavwrite(y,Fs,bits,'twosine_ringmod.wav');
```

Output of Two sine wave ring modulation ($f_c = 440$, $f_x = 380$)



Click here to hear: Two RM sine waves ($f_c = 440$, $f_x = 200$)

Amplitude Modulation

Amplitude Modulation (AM) is defined by:

$$y(n) = (1 + \alpha m(n)).x(n)$$

- Normalise the peak amplitude of M(n) to 1.
- α is *depth of modulation*

  $\alpha = 1$ gives maximum modulation

  $\alpha = 0$ tuns off modulation
- $x(n)$ is the audio **carrier** signal
- $m(n)$ is a low-frequency oscillator **modulator**.
- When $x(n)$ and $m(n)$ both sine waves with frequencies f_c and f_x respectively we here **three** frequencies: carrier, difference and sum: f_c, $f_c - f_x$, $f_c + f_x$.

Amplitude Modulation: Tremolo

A common audio application of AM is to produce a **tremolo** effect:

- Set modulation frequency of the sine wave to below 20Hz

The MATLAB code to achieve this is tremolo1.m

```
% read the sample waveform
filename='acoustic.wav';
[x,Fs,bits] = wavread(filename);

index = 1:length(x);

Fc = 5;
alpha = 0.5;

trem=(1+ alpha*sin(2*pi*index*(Fc/Fs)))';
y = trem.*x;

% write output
wavwrite(y,Fs,bits,'out_tremolo1.wav');
```

Click here to hear: original audio, AM tremolo audio.

Tremolo via Ring Modulation

If you ring modulate with a triangular wave (or try another waveform) you can get tremolo via RM, tremolo2.m

```
% read the sample waveform
filename='acoustic.wav';
[x,Fs,bits] = wavread(filename);

% create triangular wave LFO
delta=5e-4;
minf=-0.5;
maxf=0.5;

trem=minf:delta:maxf;
while(length(trem) < length(x) )
   trem=[trem (maxf:-delta:minf)];
   trem=[trem (minf:delta:maxf)];
end

%trim trem
trem = trem(1:length(x))';

%Ring mod with triangular, trem
y= x.*trem;

% write output
wavwrite(y,Fs,bits,'out_tremolo2.wav');
```

Click here to hear: original audio, RM tremolo audio.

Non-linear Processing

Non-linear Processors are characterised by the fact that they create (intentional or unintentional) harmonic and inharmonic frequency components not present in the original signal.

Three major categories of non-linear processing:

Dynamic Processing: control of signal envelop — aim to minimise harmonic distortion Examples: Compressors, Limiters

Intentional non-linear harmonic processing: Aim to introduce strong harmonic distortion. Examples: Many electric guitar effects such as distortion

Exciters/Enhancers: add additional harmonics for subtle sound improvement.

Overdrive, Distortion and Fuzz

Distortion plays an important part in electric guitar music, especially rock music and its variants.
Distortion can be applied as an effect to other instruments including vocals.

Overdrive — Audio at a low input level is driven by higher input levels in a non-linear curve characteristic

Distortion — a wider tonal area than overdrive operating at a higher non-linear region of a curve

Fuzz — complete non-linear behaviour, harder/harsher than distortion

Overdrive

For overdrive, **Symmetrical soft clipping** of input values has to be performed. A simple three layer *non-linear soft saturation* scheme may be:

$$f(x) = \begin{cases} 2x & \text{for } 0 \leq x < 1/3 \\ \frac{3-(2-3x)^2}{3} & \text{for } 1/3 \leq x < 2/3 \\ 1 & \text{for } 2/3 \leq x \leq 1 \end{cases}$$

- In the lower third the output is liner — multiplied by 2.
- In the middle third there is a non-linear (quadratic) output response
- Above 2/3 the output is set to 1.

MATLAB Overdrive Example

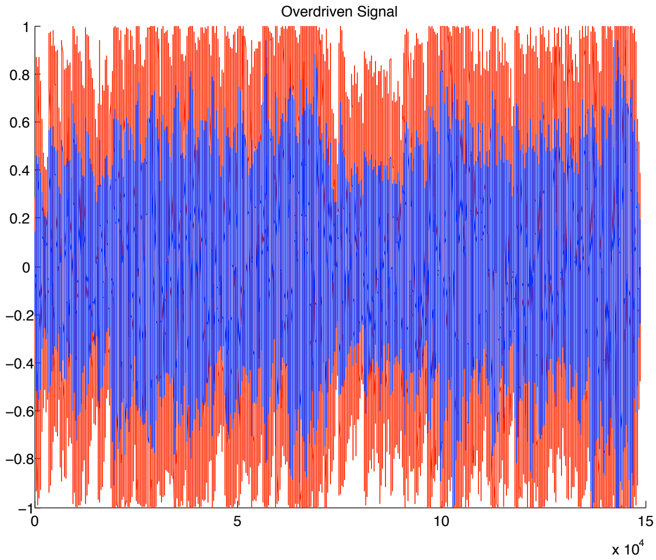
The MATLAB code to perform symmetrical soft clipping is, symclip.m:

```
function y=symclip(x)
% y=symclip(x)
% "Overdrive" simulation with symmetrical clipping
% x     - input
N=length(x);
y=zeros(1,N); % Preallocate y
th=1/3; % threshold for symmetrical soft clipping
        % by Schetzen Formula
for i=1:1:N,
   if abs(x(i))< th, y(i)=2*x(i);end;
   if abs(x(i))>=th,
     if x(i)> 0, y(i)=(3-(2-x(i)*3).^2)/3; end;
     if x(i)< 0, y(i)=-(3-(2-abs(x(i))*3).^2)/3; end;
   end;
   if abs(x(i))>2*th,
     if x(i)> 0, y(i)=1;end;
     if x(i)< 0, y(i)=-1;end;
   end;
end;
```

MATLAB Overdrive Example (Cont.)

An **overdriven signal** looks like this , overdrive_eg.m:

```
% read the sample waveform
filename='acoustic.wav';
[x,Fs,bits] = wavread(filename);

% call symmetrical soft clipping
% function
y = symclip(x);

% write output
wavwrite(y,Fs,bits,...
        'out_overdrive.wav');

figure(1);
hold on
plot(y,'r');
plot(x,'b');
title('Overdriven Signal');
```



Click here to hear: original audio, overdriven audio.

Distortion/Fuzz

A non-linear function commonly used to simulate distortion/fuzz is given by:

$$f(x) = \frac{x}{|x|}(1 - e^{\alpha x^2/|x|})$$

- This a non-linear exponential function:
- The gain, α, controls level of distortion/fuzz.
- Common to mix part of the distorted signal with original signal for output.

MATLAB Fuzz Example

The MATLAB code to perform symmetrical soft clipping is, fuzzexp.m:

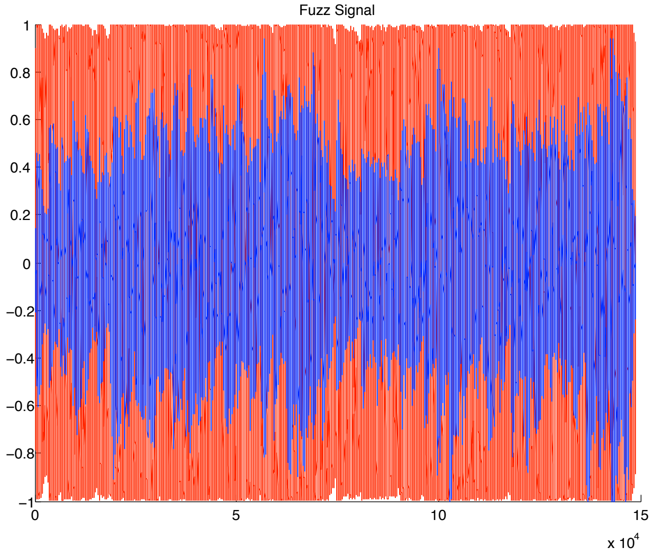
```
function y=fuzzexp(x, gain, mix)
% y=fuzzexp(x, gain, mix)
% Distortion based on an exponential function
% x    - input
% gain - amount of distortion, >0->
% mix  - mix of original and distorted sound, 1=only distorted
q=x*gain/max(abs(x));
z=sign(-q).*(1-exp(sign(-q).*q));
y=mix*z*max(abs(x))/max(abs(z))+(1-mix)*x;
y=y*max(abs(x))/max(abs(y));
```

Note: function allows to `mix` input and fuzz signals at output

MATLAB Fuzz Example (Cont.)

An **fuzzed up signal** looks like this , fuzz_eg.m:

```
filename='acoustic.wav';

% read the sample waveform
[x,Fs,bits] = wavread(filename);

% Call fuzzexp
gain = 11; % Spinal Tap it
mix = 1; % Hear only fuzz
y = fuzzexp(x,gain,mix);

% write output
wavwrite(y,Fs,bits,'out_fuzz.wav');
```



Click here to hear: original audio, Fuzz audio.