Subtractive Synthesis

- Additive synthesis involves building the sound by summing desired frequency components.
- In subtractive synthesis, new signals/sounds are created by removing or subtracting spectral components from a source.
- Any sound can be used as a source for subtractive synthesis (such as the sounds we’ve been synthesizing), but it is also very common to use a broadband source such as noise or a pulse.

Filters

- Any medium through which a signal passes may be regarded as a filter.
- Typically however, we view a filter as something which modifies the signal in some way. Examples include:
  - stereo speakers
  - our vocal tract
  - our musical instruments
- A digital filter is one that operates on digital signals (such as the ones we’ve been looking at). It is a formula for going from one digital signal to another.

![Figure 1: A black box filter.](image)

Describing a Filter

The characteristic of a filter is described by its frequency response which consists of:

1. The Amplitude Response (or magnitude frequency response):
   - Describes the gain of a filter at every frequency: a positive gain boosts the signal while a negative gain attenuates the signal.
   - Determined by the ratio of the peak output amplitude to the peak input amplitude at a given frequency.
2. The Phase Response:
   - Describes the delay of a filter at every frequency.
   - Determined by subtracting the input phase from the output phase at a particular frequency.

A frequency response describes how the amplitude and phase of a sound’s spectral components is changed by the filter.
Determining the Frequency Response

- The Frequency Response may be determined using the following two approaches:

1. **Sinewave Analysis**: checking the behaviour of the filter at every possible frequency between 0 and $f_s/2$ Hz.

2. **Obtaining the Impulse Response**: use an input signal that contains all of those frequencies, and then we only have to do the “checking” operation once.

   - An input signal with the broadest possible spectrum is a unit impulse, a signal which equals 1 at time zero and 0 elsewhere:
     \[
     \delta(n) = \begin{cases} 
     1, & n = 0 \\
     0, & n \neq 0 
     \end{cases}
     \]

   - The output of the filter in response to the impulse is called an *impulse response*.

   - The spectrum of the impulse response gives the frequency response of the filter from DC to the Nyquist limit.

Linear Time Invariant (LTI) Filters

- Any LTI system may be completely characterized by its impulse response. That is, its output can be calculated in terms of the input and the impulse response.

- A filter is linear if

  1. the amplitude of the output is proportional to the input amplitude (scaling property),
     \[
     H\{gx_1(\cdot)\} = gH\{x_1(\cdot)\}
     \]
  2. the output due to a sum of input signals is equal to the sum of outputs due to each signal alone (superposition property).
     \[
     H\{gx_1(\cdot) + gx_2(\cdot)\} = gH\{x_1(\cdot)\} + gH\{x_2(\cdot)\}
     \]

- Time-invariant means that the filter behaviour does not change over time (it is not time dependent).

- An LTI filter can only attenuate or boost the amplitude and/or modify the phase of the input spectral components. It does not introduce frequency components to the input signal.

Four Simple Filter Types

- There are four (4) basic types of filters:

  1. low-pass
  2. high-pass
  3. band-pass
  4. band-reject

![figure](image.png)

Specifying the Filter

- A low-pass filter permits frequencies below a cutoff frequency $f_c$ to pass with little change while attenuating or rejecting frequencies above $f_c$.

- Conversely, a high-pass filter permits frequencies above $f_c$ to pass while attenuating those frequencies which fall below.

- A filter which passes frequencies within a certain frequency band (the pass band) is called a band-pass filter.

- A filter which attenuates frequencies with a certain frequency band (the stop band) is called a band-reject (or notch) filter.
An Ideal Lowpass Filter

- A lowpass filter is one that allows low frequencies to pass, while attenuating anything above a specified cutoff frequency \( f_c \).
- The amplitude response of an ideal low-pass filter is shown below.

![Amplitude Response for an ideal low-pass filter.]

- Such an ideal amplitude response is not realizable in digital signals. How close we come depends on the complexity of the filter.

A Simple Lowpass Filter

- The simplest low-pass filter (and therefore furthest from ideal) is given by the difference equation
  \[ y(n) = x(n) + x(n-1). \]
- The corresponding system diagram is

![System Diagram for a simple low-pass filter.]

where \( z^{-1} \) means “delay by 1 sample”.
- Recall that if a signal can be expressed in the form
  \[ x(t) = s(t - t_1), \]
  \( x(t) \) is a time-shifted version of \( s(t) \), where
  - a positive \( t_1 \) is a time delay (right shift on the time axis),
  - a negative \( t_1 \) is a time advance (left shift on the time axis).
- This filter is really just a running averager (2-point averager) with a factor of two. That is, it takes the average of two adjacent samples.

Characteristics of a Filter

- No digital filter is ideal and will always have a smooth (rather than a clearly defined) transition between the pass and stop bands at the cutoff frequency \( f_c \).
- The cutoff frequency is typically defined as the frequency at which the power transmitted by the filter drops to one-half (by -3 dB) of the maximum power transmitted in the passband.
- A pass-band filter is characterized by its quality factor \( Q \) which is inversely proportional to its bandwidth:
  \[ Q = \frac{f_c}{\text{BW}}, \]
  where \( f_c \) is the center frequency of the band.
  - A high quality factor (a high \( Q \)) denotes a filter with a narrow bandwidth and a low \( Q \) denotes a filter with a wide bandwidth.

Intuitive Analysis of LP Filter

- The running average of an input signal with little or no variation from sample to sample is very close to the input signal.
- The running average of an input signal with significant variation from sample to sample will be quite different than the input signal.

So why low-pass?

- Consider an input \( x_1(n) \) to this filter at the lowest possible frequency 0 Hz (DC). This signal doesn’t alternate and has a single amplitude value over time:
  \[ x_1(n) = [A, A, A, ...], \]
  yielding an output of
  \[ y(n) = x_1(n) + x_1(n-1) = 2A, \text{ for } n > 0. \]
- The filter therefore applies a gain of 2 at the lowest possible frequency.
Why is this a Low-Pass Filter?

• Consider a second input \( x_2(n) \) to this filter at the highest possible frequency, \( f_s/2 \) (the Nyquist limit). Such a signal is say to be critically sampled and will swing ever more drastically from sample to sample,

\[
x_2(n) = [A, -A, A, ...],
\]

for a cosine of amplitude \( A \), yielding an output of

\[
y(n) = x_2(n) + x_2(n - 1) = A - A = 0, \quad \text{for } n > 0.
\]

• This filter therefore, in addition to boosting low frequencies as shown in the previous slide, attenuates (stopping) components at higher frequencies.

What about at \( f_s/4 \)?

• A cosine with frequency \( f = f_s/4 \) and amplitude \( A \) would have sample values

\[
x_2(n) = [A, 0, -A, 0, A, ...],
\]

yielding an output of

\[
y(n) = x_2(n) + x_2(n - 1)
\]

\[
= [A, 0, -A, 0, A, ...] + [0, A, 0, -A, 0, A, ...]
\]

\[
\]

• Though this may not appear to be sinusoidal, it is in fact a cosine with amplitude of \( \sqrt{2} \approx 1.4 \), with a phase delay \( \pi/4 \).

Simple Low-Pass Impulse Response

• Recall that we may find the frequency response of the filter by checking the behaviour of the filter at every possible frequency between 0 and \( f_s/2 \) Hz, or by obtaining its impulse response.

• If the input is a unit impulse:

\[
\delta(n) = [1, 0, 0, 0, ...],
\]

then the output is

\[
h(n) = \delta(n) + \delta(n - 1)
\]

\[
= [1, 0, 0, 0, ...] + [0, 1, 0, 0, 0, ...]
\]

\[
= [1, 1, 0, 0, 0, ...].
\]

• Notice how the impulse response is equal to the coefficients of the filter (we’ll see this again later).

• Exercise: Take the DFT of impulse response \( h(n) \) to obtain frequency response \( H(\omega) \).

Matlab Low-pass Filter Implementation

• One possible implementation in Matlab (though we’ll see a better one later) for the filter

\[
y(n) = x(n) + x(n - 1), \quad n = 1, 2, 3, ..., N
\]

\[
N = 1024; \quad \text{% signal length}
\]

\[
x = [1 \text{ zeros}(1, N-1)]; \quad \text{% impulse}
\]

\[
y = \text{zeros}(1, N); \quad \text{% output buffer}
\]

\[
y(1) = x(1); \quad \text{% impulse response}
\]

for \( n=2:N 
\]

\[
y(n) = x(n)+x(n-1); \quad \text{% impulse response end}
\]

\[
\%	ext{ plot spectrum}
\]

\[
Y = \text{fft}(y); \quad \text{% frequency response}
\]

\[
Y = \text{abs}(Y); \quad \text{% amplitude response}
\]

\[
f \text{m} = (0:N/2)/N; \quad \text{% normalized frequency axis}
\]

\[
\text{subplot}(211); \text{plot}(f \text{m}, Y); \text{ grid}; \text{set(gca, 'XLim', [0 0.5]);}
\]

\[
\text{title('Amplitude Response \( y(n) = x(n) + x(n-1) \'),'Ylim', [0 0.5]);}
\]

\[
x \text{label('Frequency (normalized)'),'Ylabel','Magnitude (linear)');}
\]

\[
\text{subplot}(212); \text{plot}(f \text{m}, 20\text{log10}(Y)); \text{ grid;}
\]

\[
x \text{label('Frequency (normalized)'));}
\]
Other simple non-recursive filters

- Test the filter
  \[ y(n) = x(n) - x(n - 1) \]
  with DC and the Nyquist limit. What conclusions do you draw?
- It can be similarly verified that the filter
  \[ y(n) = x(n) + x(n - 2) \]
  passes both DC and the Nyquist limit. An input signal at a frequency equal to a quarter of the sampling rate,
  \[ x(n) = [A, 0, -A, 0, A, 0, -A, \ldots] \]
  however produces no output. We may therefore assume this is a band-reject, or notch filter, with a notch at \( f_s/4 \).
- The filter
  \[ y(n) = x(n) - x(n - 2) \]
  rejects both DC and Nyquist limit frequencies yet boosts frequencies at \( f_s/4 \). What kind of filter is it?

Plots of simple filters

Generalized FIR

- It should be clear by now that several different nonrecursive filters can be made by changing the delay (and thus the order) of the filter, and the filter coefficients.
- The general equation for an FIR (Finite Impulse Response) filter is given by
  \[ y(n) = \sum_{k=0}^{M} b_k x(n - k) \]
  where \( M \) is the order of the filter.
- It should also be evident that a filter can be defined simply on a set of coefficients. For example if
  \[ b_k = [1, 3, 3, 1] \]
  we have a third order filter (\( M = 3 \)) which expands into the difference equation
  \[ y(n) = x(n) + 3x(n - 1) + 3x(n - 2) + x(n - 3); \]
- When the input to the FIR filter is a unit impulse sequence, the output is the unit impulse response.
A Better Matlab Implementation

• Previously, we implemented an FIR filter in Matlab using a “for loop”. This was done only to give insight into how to implement the filter. Loops are very slow in Matlab and therefore should be avoided if possible.
• Matlab has a function called filter which will do the filtering operation as implemented by the loop in the previous implementation.

Matlab’s filter function

• The Matlab implementation for the filter is most easily accomplished using the filter function
  \[ y = \text{filter}(B, A, x). \]
• The filter function takes three (3) arguments:
  1. feedforward coefficients \( B \),
  2. feedback coefficients \( A \) (set to 1 for an FIR filter),
  3. and the input signal \( x \).
• If the filter doesn’t have feedback coefficients, as is the case with an FIR filter, we must still set \( A = 1 \).
• An \( N \)th order filter will have a delay of \( N \). Our simple filter \( y(n) = x(n) + x(n-1) \) is, therefore, a first order filter and has a length of two (2), where the values for each of the coefficients is set to \( B = [1, 1] \).

Sinewave Proof (to do in class)

• In class examples. Create the following filter in Matlab:
  \[
  B = [1 1];
  A = 1;
  y = \text{filter}(B, A, x);
  \]
  – Input a sinusoid at DC.
  – Input a sinusoid with frequency \( f_s/2 \).
  – Input a frequency with frequency \( f_s/4 \);

Increasing the Filter Order

• Let’s return now to the simple low-pass filter \( y(n) = x(n) + x(n-1) \) which is just a two point running average (with a factor of 2).
• If we increase the number of samples averaged, i.e. increase the filter order,
  \[ y(n) = x(n) + x(n-1) + x(n-2), \]
the waveform will be smoothed (with a more gentle slope to zero), which corresponds to a lowered cutoff frequency.

![Figure 7: A Cascade of Simple Lowpass filters.](image-url)
Coefficients as Impulse Response

• If we look at the contents of both the impulse $x$ and the impulse response $y$ we may see that the latter is merely the coefficients of our FIR (finite impulse response) filter.

  $\begin{align*}
  \text{ans} &= 1 0 0 0 0 0 0 0 0 0 \\
  \text{ans} &= 1 1 0 0 0 0 0 0 0 0
  \end{align*}$

Matlab LPF Cascade Implementation

  x = [1 zeros(1, N-1)];
  B = [1 1];
  A = [1];
  \%
  Cascade of filters
  y1 = filter(B, A, x);
  y2 = filter(B, A, y1);
  y3 = filter(B, A, y2);
  y4 = filter(B, A, y3);
  y5 = filter(B, A, y4);
  \%
  or equivalently
  \%
  y = filter(B,A, filter(B,A, filter(B,A, filter(B,A, x))));

FIR Coefficients and Impulse response

• Let the input to the cascade be a unit impulse response.

• Each time we use the previous output $y$ as the input to the same filter we have essentially created a new impulse response representing the characteristic of a new filter.

• Each time, the new output has a finite number of non-zero components (which is why this type of filter is called a finite impulse response FIR filter).

• Notice how each time we cascade another first order filter, the output impulse response increases by one more non-zero element.

• Each filter in the cascade is producing an output impulse response that is equivalent to the FIR coefficients of a new FIR filter.

Cascade Connection

• The order of the simple lowpass filter can be increased by placing several in a cascade (series connection).

$\begin{align*}
  x &= (1, 0, 0, \ldots) \\
  y_1 &= x(n) + x(n-1) \\
  y_2 &= x(n) + x(n-1) \\
  y_3 &= x(n) + x(n-1) \\
  y_4 &= x(n) + x(n-1) \\
  y_5 &= x(n) + x(n-1)
  \end{align*}$

Figure 8: A cascade of simple low-pass filters.
Transforming between low-pass and high-pass filters

- Given the coefficients of a low-pass filter this filter may be converted to a high-pass filter by multiplying every odd coefficient by -1 while leaving the even coefficients untouched (where coefficient indexing begins from zero).

- As an example, let’s take the filter resulting from a cascade of 5 simple low-pass filters, with feedforward coefficients given by $B = 1, 5, 10, 10, 5, 1$ and the feedback coefficient vector is $A = 1$.

- The Matlab implementation is given by

```matlab
N = 1024;
A = 1;
Blp = [1 5 10 10 5 1];
Bhp = [1 -5 10 -10 5 -1];
x = [1 zeros(1, N-1)];
ylp = filter(Blp, A, x);
yhp = filter(Bhp, A, x);
```

FIR Approximation to the Ideal Lowpass

- The coefficients of an $N$th order non-recursive approximation to an ideal low-pass filter with a cutoff frequency of $f_c$ are given by

$$b_k = \frac{\sin(2\pi(k - \frac{1}{2}N)(f_c/f_s))}{\pi(k - \frac{1}{2}N)}\left(0.54 + 0.46\cos\left(\frac{\pi(k - \frac{1}{2}N)}{N}\right)\right)$$

- Again, when implementing in Matlab, care must be taken for the possibility of a divide by 0.

```matlab
fs = 44100;
N = 128;
fc = fs/4;

k = [0:N-1];
kern = pi*(k-N/2);
kern(find(kern==0)) = eps;
b = sin(2.*kern.*(fc/fs))./ ... 
kern.*(0.54+0.46.*cos(kern./N));
```

Figure 9: Transforming a low-pass filter to a high-pass.

Figure 10: Approximate ideal low-pass FIR (non-recursive) filter with order $N = 128$ and a cut-off frequency of $f_s/4$. 
Recursive Filters

• Using FIR filters often requires significant computation and coefficients to reproduce a desired frequency response.

• It is often possible to reduce the number of feedforward coefficients needed to obtain a frequency response by introducing feedback coefficients.

• A simple example of a first order recursive low-pass filter is given by
  \[ y(n) = b_0 x(n) + a_1 y(n-1) \]

• The general difference equation for LTI filter therefore, is given by
  \[ y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_M x(n-m) \]
  \[ - a_1 y(n-1) - \cdots - a_N y(n-N) \]

IIR Lowpass

• To obtain a lowpass filter with a cut-off frequency of \( f_c \) and an amplitude response of one (1) at DC, define an intermediate variable:
  \[ C = 2 - \cos(2\pi(f_c/f_s)) \]

• The coefficients are then
  \[ a_1 = \sqrt{C^2 - 1 - C} \]
  \[ b_0 = 1 + a_1 \]

  \( fs = 44100; \)
  \( fc = 1000; \)
  \( C = 2-\cos(2*\pi*fc/fs); \)
  \( A(2) = sqrt(C^2 - 1) - C; \)
  \( A(1) = 1; \)
  \( B = 1+A(2); \)
  \( [H, w] = freqz(B, A); \)
  \( H = 20*log10(abs(H)); \)
  \( plot(w/(2*\pi)*fs/1000, H, fc/1000, -3, 'o'); \)
  \( title('Recursive Low-pass with f_c = 1 kHz'); \)
  \( xlabel('Frequency (Hz)'); \)
  \( ylabel('Magnitude (dB)'); \)

IIR plots

Figure 11: A simple non-recursive low-pass \( y(n) = x(n) + x(n-1) \) (top) and a recursive low-pass \( y(n) = x(n) + y(n-1) \) (bottom).
IIR High-Pass Filter

This filter can be made to have a hi-pass characteristic with the following change to the coefficients

\[
a_1 = C - \sqrt{C^2 - 1} \\
b_0 = 1 - a_1
\]

Bi-quadratic Resonant Filter

- The difference equation for another bandpass filter is given by

\[
y(n) = x(n) - Rx(n-2) + 2R \cos(2\pi f_c T) y(n-1) - R^2 y(n-2)
\]

where \( f_c \) is the resonant (or center) frequency, and \( R \) is set according to the desired bandwidth of the resonator using the following approximate relation

\[
R = e^{-\pi B_w T},
\]

where \( B_w \) is the bandwidth at -3dB in Hz given by

\[
B_w = \frac{f_c}{Q},
\]

and \( T \) is the sampling period.

Biquad

- This function is often called bi-quadratic or simply a biquad because both the numerator and denominator of its transfer function are quadratic polynomials.

- If more tuning is required, a higher order filter may be used by placing several such filters in series.

- If a resonator requires more than one mode, multiple bi-quad filters can be placed in parallel (one for each mode), with the overall output being the sum of the filter outputs.

- The two control parameters for this filter are
  1. the center frequency \( f_c \)
  2. the quality factor \( Q \) (or alternatively, the bandwidth).

- These parameters determine the characteristic of the resonator, and can be changed in real-time, making this an efficient and ideal implementation for performance situations.

Helmholtz Resonator

- Named after H. von Helmholtz (1821-1894), who used it to analyze musical sounds.

- The resonant frequency of the Helmholtz resonator is given by

\[
f_0 = \frac{1}{2\pi} \sqrt{\frac{S}{VL}}
\]

where \( S \) is the surface area of the hole, \( V \) is the volume and \( L \) is the length of the neck.

- Once the frequency of the Helmholtz resonator is known, it can be simulated using a simple two-pole two-zero resonant digital filter.
Designing butterworth filter’s using Matlab’s butter function

BUTTER Butterworth digital and analog filter design.

\[ [B,A] = \text{BUTTER}(N,Wn) \] designs an \( N \)th 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.

Sources

- A pulse waveform has significant amplitude only during a relatively brief time interval, called the pulse width.
- Unlike noise generators, pulse generators produce periodic waveforms at a repetition frequency \( f_0 \).
- Like noise, pulses have very broad spectra, making them good candidates for subtractive synthesis.
- The pulse is characterized by shape the ratio of its width to the period of the overall waveform. Narrower pulses contain a larger number of harmonics.

Band-Limited Pulse Generator

- Though pulses can take on many shapes, we should choose one that has a bandlimited spectrum to avoid aliasing.

![](image1)

Figure 14: Pulses bandlimited by \( N \) harmonics.

- We can design a pulse where the number of harmonics \( N \) in the spectrum is determined by

\[ N = \text{int}\left( \frac{f_s}{2f_0} \right) \]

Creating a Pulse Generator

- Let \( N \) be the number of harmonics. The spectrum of the pulse generator may be synthesized by the sum of harmonics:

\[ f(t) = \frac{A}{N} \sum_{k=1}^{N} \cos(2\pi kf_0 t) \]

- There is however, a more efficient way which makes use of the following closed form expression.

\[ \frac{A}{N} \sum_{k=1}^{N} \cos(2\pi kf_0 t) = \frac{A}{2N} \left( \frac{\sin((2N + 1)\pi f_0 t)}{\sin(\pi f_0 t)} - 1 \right) \]
Possible Division by Zero

• Notice that though we got rid of the computational expensive summation, we are left with a possible division by zero which will either crash the program or create an undesirable NaN (not a number) output.
• We must check if there is a division by zero, and if so, use instead the equation

\[ f(t) = \frac{A}{2N} \left( \frac{(2N + 1) \cos((2N + 1)\pi f_0 t)}{\cos(\pi f_0 t)} - 1 \right). \]

Matlab implementation

• The following Matlab implementation avoids the use of for loops.

```matlab
fs = 1024;
nT = -0.5:1/fs:0.5-1/fs;
f0 = 1;
A = 1;
N = 17;
DIV = sin(pi*f0*nT);
i1 = find(DIV==0);
i2 = find(DIV~=0);
x(i1) = A/(2*N)*(sin((2*N+1)*pi*f0*nT(i1)) ./ DIV(i1) -1);
x(i2) = A/(2*N)*((2*N+1)*cos((2*N+1)*pi*f0*nT(i2)) ./ cos(pi*f0*nT(i2)) -1);
```

Figure 15: The output of the closed form and discrete summation is the same, though the time to compute is drastically different.