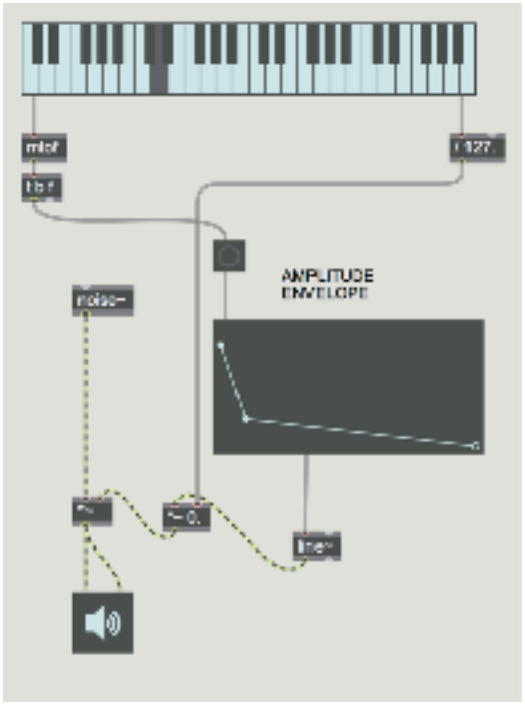
SUBTRACTIVE SYNTHESIS

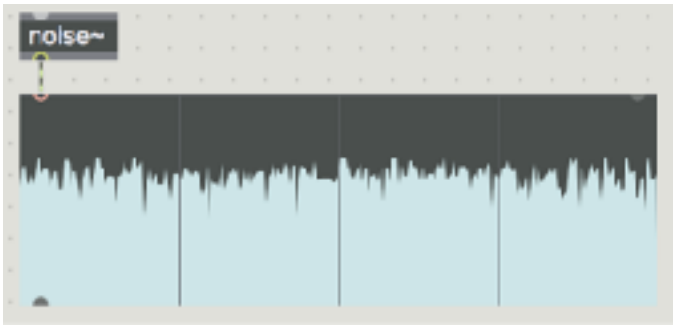
In additive synthesis, we layered sine tones to create unique timbres. **Subtractive synthesis** works in the opposite direction, beginning from a rich frequency spectrum and using **filters** to sculpt the frequency spectrum into a unique timbre.

FILTERED NOISE

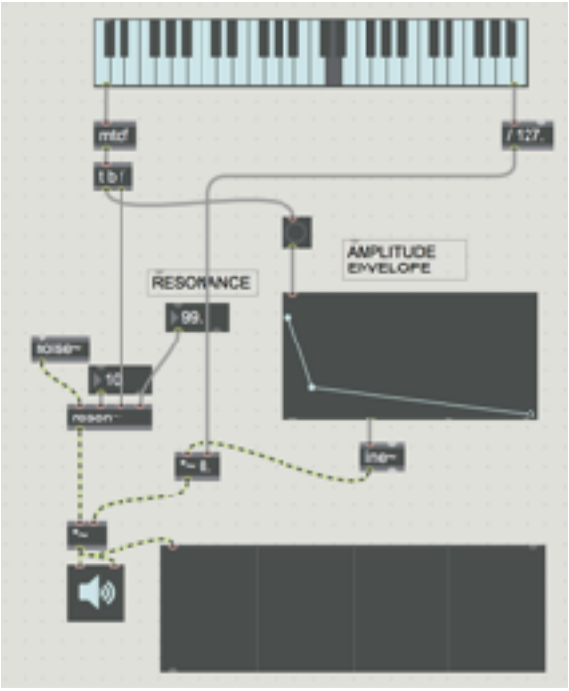
What kind of sound has the richest possible frequency spectrum? That's right! **White noise** has the richest possible frequency spectrum, with even energy distribution throughout the spectrum. Let's use this as the basis for our first foray into subtractive synthesis. We can begin by first opening up our patch from last time (additive_synthesis_ex5) and deleting our oscillators. This will leave us with our keyboard and amplitude envelope. In place of our oscillators, let's put a **[noise~]** object. **PATCH: subtractive_synth_ex0**



What does it sound like when we play a key? There is definitely a percussive element to it. Subtractive synthesis is frequently used for percussion synthesis, since many percussion sounds are quite noisy - noisy in the sense that they have widespread energy distribution in their frequency spectrum, not noisy in the sense of a construction site or rock 'n' roll. This kind of noise can be seen when we connect the spectroscope~ object to our output.

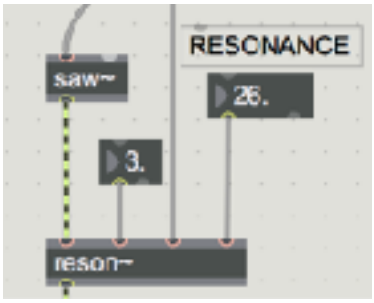


What if we wanted to add pitch? This is where we use filtering to subtract from our frequency spectrum. Let's use a bandpass filter with the **[reson~]** object. We can now pitch our noise, and play with the resonance of our filter for less or more noise in our sound. We also should boost our signal's gain, since when we cut out that much energy from the frequency spectrum, the overall amplitude of our signal is greatly decreased. **PATCH: subtractive_synth_ex1**

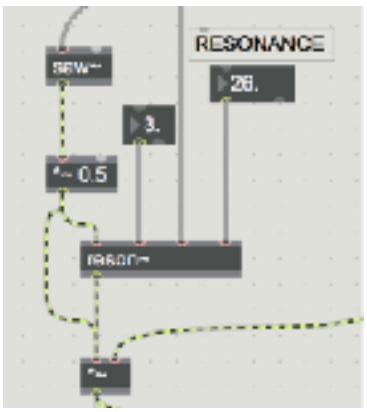


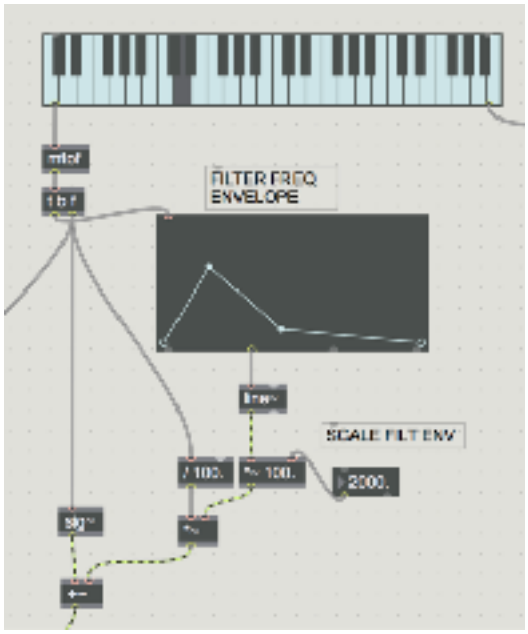
SAW & DRY/WET

Let's take a look at **[saw~]**. The sawtooth waveform has even energy in all of its partials, giving it a very bright, harsh sound - perfect for subtractive synthesis. Let's replace our noise~ object with a saw~ object and see what kind of aesthetic is produced. **PATCH: subtractive_synth_ex2a**



This sounds very much like a sine wave. Why is that? That's right, we are only isolating the fundamental frequency through our reson~ object. Let's brightness the sound up by mixing in the unfiltered sound with our filtered sound. How might we do this?
PATCH: subtractive_synth_ex2b





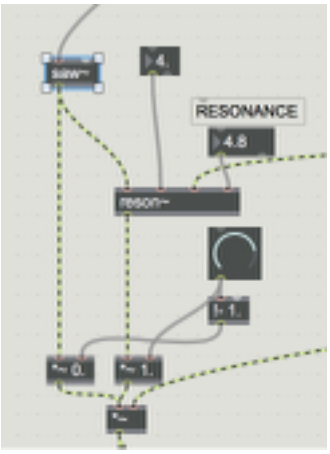
We can simply halve the signal then add it back to itself. This will insure that our signal doesn't **clip** (exceed 1.0). We can hear that the sound is now **brighter** (has more energy in the upper partials).

We can make our sound more dynamic by creating an envelope for our filter. How might we do this?

We have to add (or subtract) a value from our frequency. However, frequency is scaled

logarithmically, so the value we add or subtract shouldn't be fixed but rather proportional to our input frequency. Therefore, we can divide our input frequency by 100, then multiply that by a value according to a new function object. Let's try to code that. **PATCH: subtractive_synth_ex3**

We can change the gain of the **dry** signal versus the **wet** signal to make our signal more or less filtered. If we wanted this to be controlled by a signal knob, how could we do that?

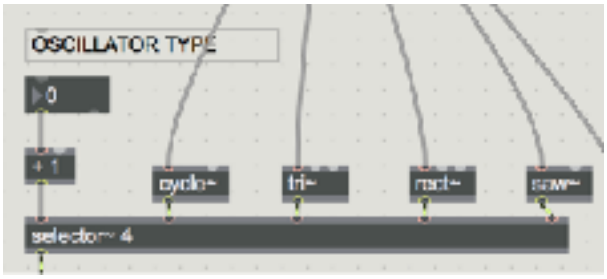


The **[!- 1.]** object subtracts the input from 1, so if we use this to control the gain for the dry signal, and the original value of the dial (0-1.) to control the gain for the filter, there will always be a combined value of 1. for their mixed signal. Max/MSP requires a bit of logic in this way - figuring out how basic arithmetic can be strung together to create a functioning, logical and elegant signal flow. **PATCH: subtractive_synth_ex4**

Right now we are using the most elementary of synthesis building blocks, but a synthesizer can use a large number of objects to create very lush, nuanced and evolving sounds. Let's take a quick look at other filters and oscillators offered by Max.

MORE FILTERS AND OSCILLATORS

Let's try out different oscillators. We can do this by using a **[selector~]** object to move through different signal inputs.



The four basic waveforms are the **sine wave, triangle wave, square wave,** and **sawtooth wave,** shown in this patch as the **[cycle~], [tri~], [rect~] and [saw~]** objects. These objects are specially designed to be used as audio rate oscillators, since other objects, such as **[triangle~]** and **[phasor~]** which produce the same waveform have not been **antialiased** (meaning they have harmonic noise in their signal since their harmonics extend above the 22.1kHz Nyquist frequency (more on this next lesson) - however, those objects produce their own aesthetic and could also be used if one preferred their sound).

While these are the most basic waveforms for subtractive synthesis, they are far from the full extent of what's possible with synthesis. When we look at sampling in two weeks, we can look

at using sample fragments as **wavetables**, meaning using an array of values that are played back very quickly to produce completely unique and individual timbres.
There are many filter types. One common way to move through different kinds of filters is to use the **biquad** object. This object allows you to make any filter type and graphically control it with the **filtergraph** object. Combining our oscillators with the biquad and filtergraph already makes for a significantly more flexible synthesizer from our first round. Play around with the settings for this. How could we expand on it? **PATCH: subtractive_synth_ex5**