# Max and Filters

Filters create the "sound" of any digital composition system. After all, everything else is designed for perfect fidelity. With version 2, MSP has added a very nice suite of filters.

## Filter terms

Here are some definitions:
A filter is a circuit or software routine that changes the frequency response of a signal. The terms you really need to know are:

- Frequency response - this is a graph of the output amplitude produced by applying a sweep of frequencies to the filter..
- Phase response- this is a graph of how the phase of signals is changed at various frequencies.
- Passband-- the range of frequency that is relatively unaffected by the filter. If the passband is below the cutoff frequency, it is a lowpass filter. If the passband is above the cutoff frequency, it's highpass. With two cutoff frequencies, you can have bandpass or notch.
- Cutoff frequency -- the frequency at which the output is reduced 3 dB below the level of the passband.
- Turnover frequency. The frequency at which the output level begins to fall off.
- Bandwidth- a band pass filter has two cutoff frequencies. The bandwidth is the distance between them.
- Slope -- (or rolloff) the rate at which output declines outside of the passband. Generally quoted in dB per octave.
- Q- for band pass filters, the ratio of amplitude to bandwidth. High Q implies a narrow peak. Q is also loosely used in other kinds filters that add a bump in response at the cutoff frequency.
- Gain - the change in peak amplitude of signal in the passband. Many filters increase the signal level substantially.
- Cascade -- a method for obtaining complex filter shapes (or steeper cutoffs) by sending the signal through a series of filters.
- Parallel -- connecting several filters to the same signal and adding their outputs.

A digital filter works by combining a delayed signal with the original. A rough example can be created with tapin~ and tapout~. If the original signal is mixed with the delayed signal, certain frequencies will be reinforced because the time it takes to get through the delay is exactly equal to the period of the signal. This will also happen at harmonics of the magic frequency. If the delay is a half period, 1.5, 2.5 3.5 periods and so on, the original and delayed signals will be out of phase and null out.

Here is the frequency response when the tap delay is 5 milliseconds:

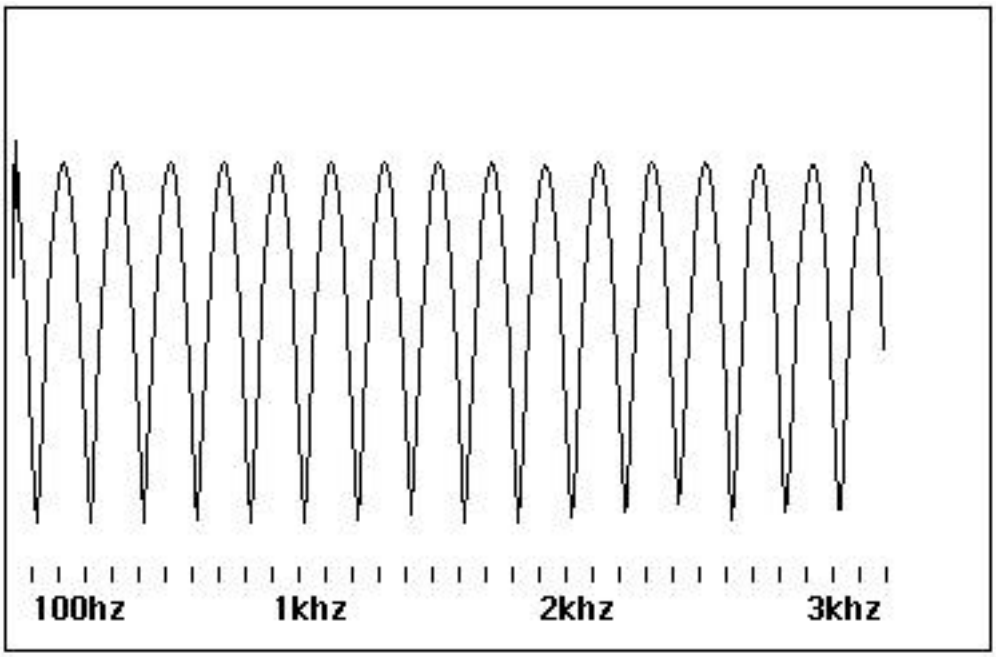


Fig 1. tapin~ to tapout~ 5ms

The maximum amplitude occurs at a frequency of 200 hz and its harmonics. The first null is 100 hz, and the others fall at the midpoints between the peaks. Note that the transition from the peaks to the nulls is gradual. This is an important feature of digital filtering, because it means that combinations of delays can be mixed to produce smooth and complex curves.

A word about the method of displaying frequency responses. These were all made with the LCD object using average~ to measure the amplitude at each frequency. The tap delay frequency response is shown with a linear frequency scale to reveal the equal spacing of the peaks. The vertical range is from 0 to 1.0, matching the full scale amplitude in MSP.. The average~ object is set to measure rms, so a sine wave with amplitude of 1.0 will fill about $7/10^{ths}$ of the screen.

These curves are often plotted on a logarithmic frequency scale. This gives each octave equal space from left to right. Likewise, the vertical scale is often in db. The other responses will be shown this way.

## Lores~

The easiest filter to use is lores~, which emulates the kind of filter found in most analog synthesizers -- a sweepable low pass response with adjustable Q (or resonance). Here is a set of responses for the lores~:

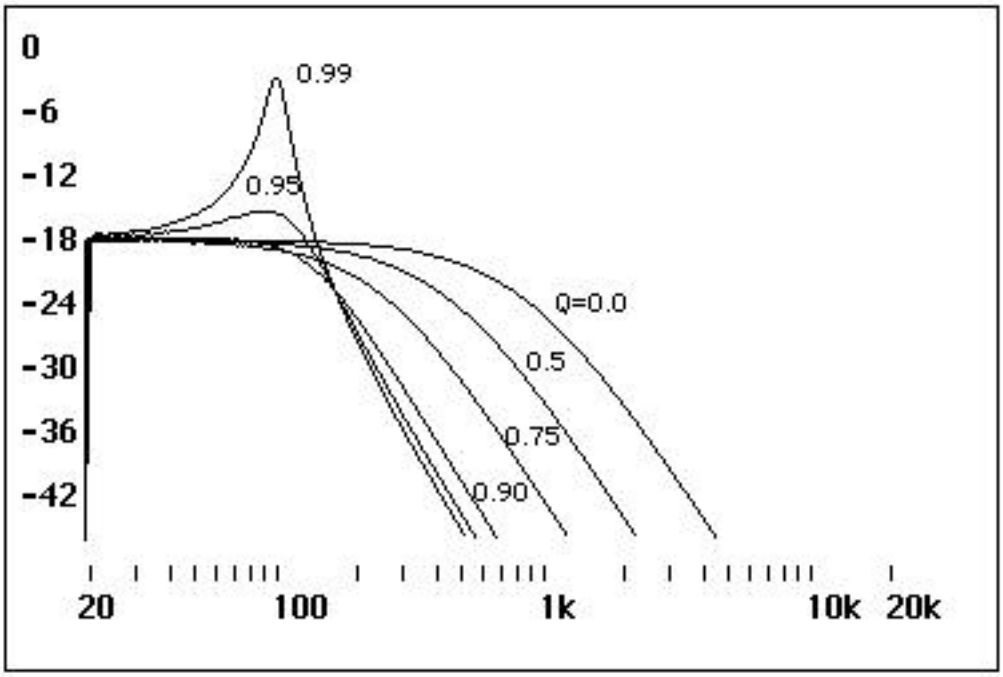


Fig 2. lores~ filter, 100hz, various resonance settings (Q)

The frequency parameter was set at 100 hz, and the response was taken for six different Q settings. Notice that the cutoff frequency ( the -3dB point) varies with the Q. This is typical of simple filter designs, analog and digital. The frequency specified would be more properly called the turnover frequency, the point at which the level begins to drop.

In filter design, Q is a number that varies from some fraction to 2000 or more, depending on the type of filter. The limit of Q will either be the point at which the circuit resonates, or where the softest practical input will distort. For lores~, Q is specified as a resonance factor, with 1.0 being the limit of Q. If you go beyond this, the filter will "explode", perhaps giving a loud distorted sound, but more likely just shutting off. To fix an abused filter, turn audio off and adjust the parameters.

Figure 3 is a plot run at 1000 hz, which follows the textbooks more closely.

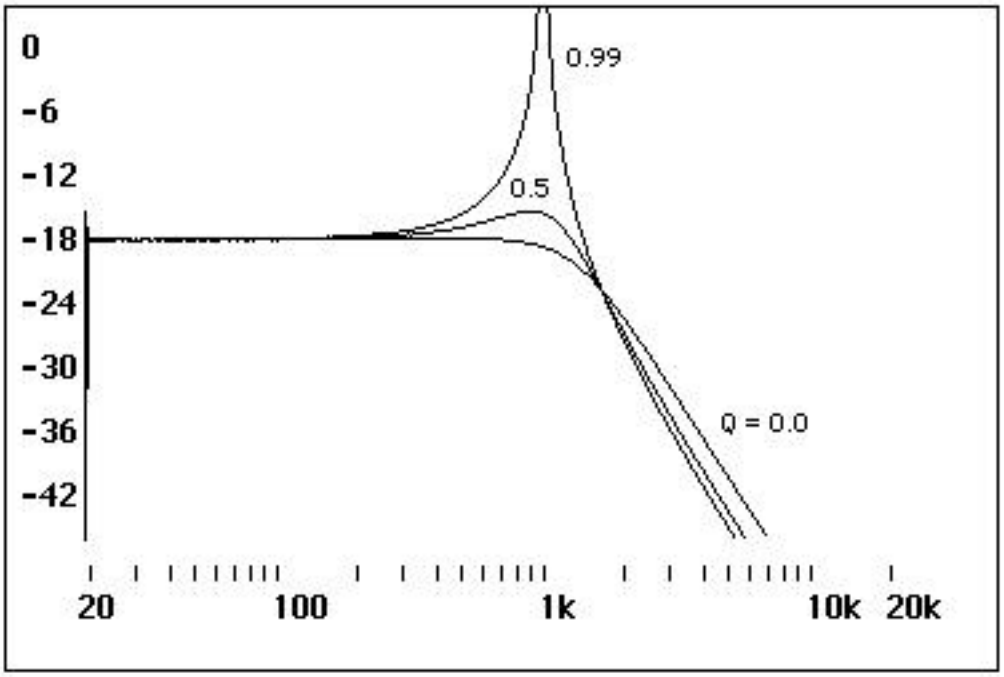


Fig 3. lores~ filter, 1khz, various resonance settings (Q)

You can also see that the height of the resonant peak varies with the frequency setting. This limits the kind of signal you can put through lores~. Typically you'll need to stay with waveforms that don't have strong high frequency partials. Here's a run with various frequency settings and Q held constant:

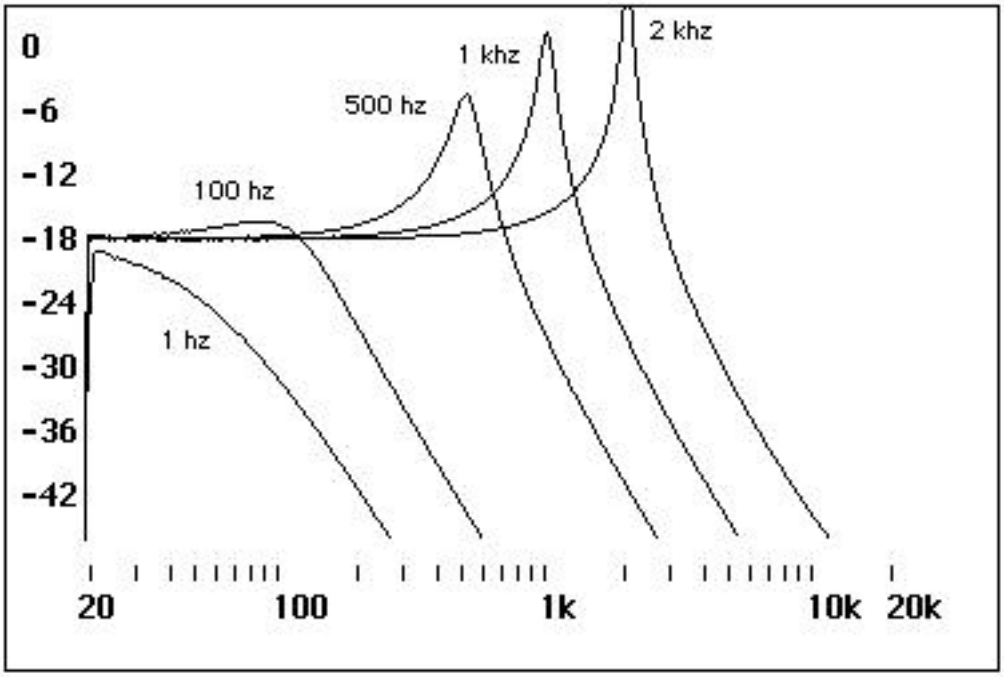


Fig. 4 lores~ filter, resonance = 0.94, various frequency settings.

As a design exercise, lores~ would not win any awards, but it's a very interesting filter. It's very much like those on analog synthesizers. My favorite way to use lores~ is for gating:

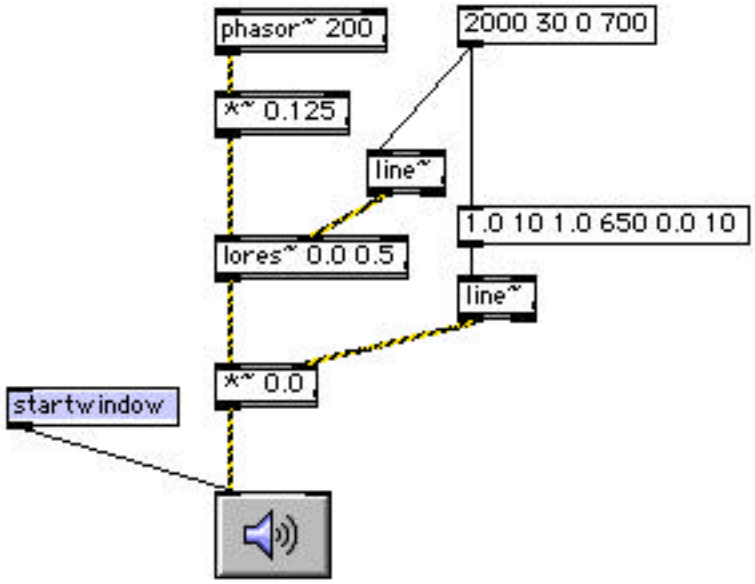


Fig. 5 Lores~ as a gate

In Figure 5, line~ is used to produce an envelope that sweeps the frequency of the filter up and down in the classic analog synthesizer way. This gives a brassy sound. The second line is there to apply an amplitude envelope, since with the resonance factor at 0.5 a 200hz tone will still be audible when the filter frequency is 0. With higher resonance settings, the partials of the waveform are emphasized as the filter sweeps.

You always need to control levels going into lores~, because it very often gives back more than it got.

You can get a really funky high pass effect by subtracting the original signal from the output of lores~.

### Reson~

Reson~ is a better behaved version of lores. It includes an independent control of gain to keep the resonance peak under control. Figure 6 shows the effect of changing gain.

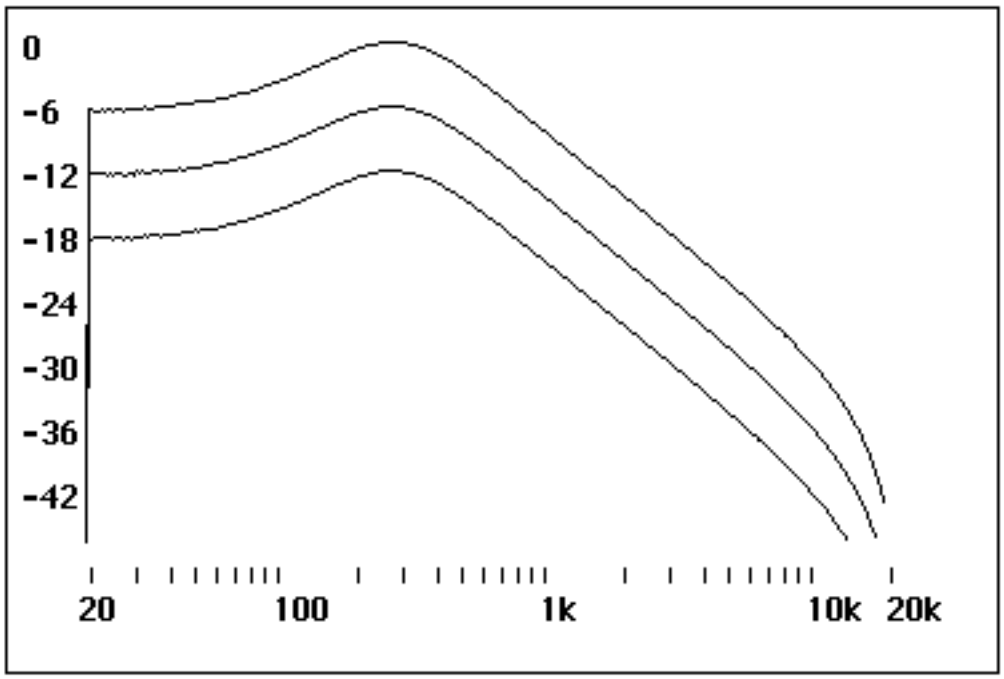


Fig 6. Reson~ gain of 1, 0.5 and 0.25

Note that cutting the gain in half produces an overall reduction of 6 dB.

Q is specified as a ratio up to 100. Figure 7 shows the effect of various Q settings. Gain is automatically recalculated for each Q setting to make the peaks equal.

The rolloff of reson~ is more gradual than lores~. This makes it a bit less effective for gating signals, unless you cascade two of them., and then the peaking effect is fairly prominent. What it really does nicely is isolate a narrow band of frequencies when the Q is high. The fffb~ object (fast fixed filter bank) is a group of reson~ type filters combined for computational efficiency. This really shines for graphic EQ type situations. The filters share an input but can be otherwise used independently.

The frequency specification for fffb~ is a little awkward, as it consists of a base frequency and then a ratio to derive the others. Here are some handy ratios:

- 2.0 for one per octave
- 1.4142 for half octave spacing
- 1.25992 for third octave spacing

- 1.2246 for 1/6th octave
- 1.0905 for 1/8th octave
- 1.059463 for equal temperament (you can play tunes!)

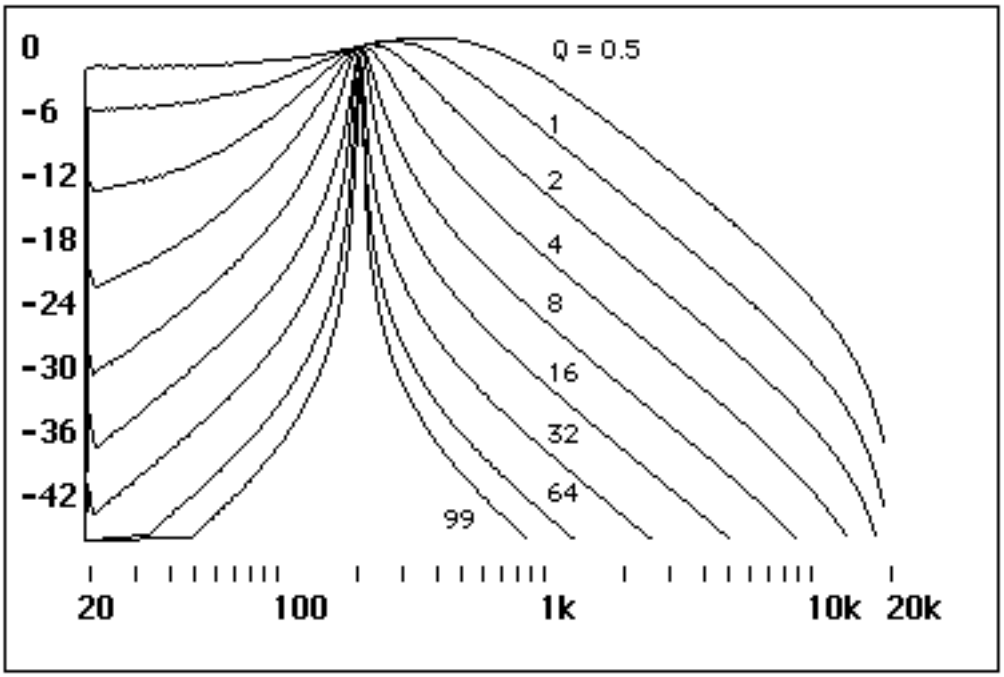


Fig 7 reson~ at 200hz and gain of 1.0 with various Q settings.

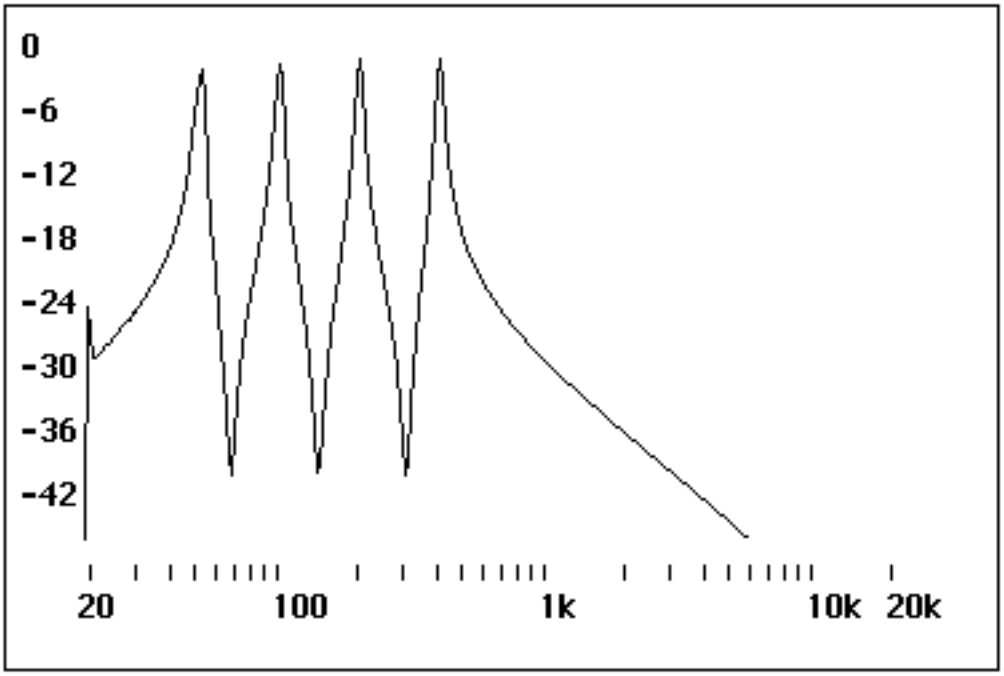


Fig 8. Fffb~ configured for 4 filters with a Q of 50 (summed)

### Svf~

The svf~ filter probably should win some kind of award. Its performance is right out of the textbooks. Svf stands for State Variable Filter, and is another circuit common in old analog synthesizers. It was usually called a multi mode filter, because lowpass, highpass, bandpass, and notch responses are available simultaneously from separate jacks. They would typically be stable at enormous Q values- the best would ring like a bell with a pulse input, but would never distort. Figure 9 shows how the four filter outputs relate to each other:

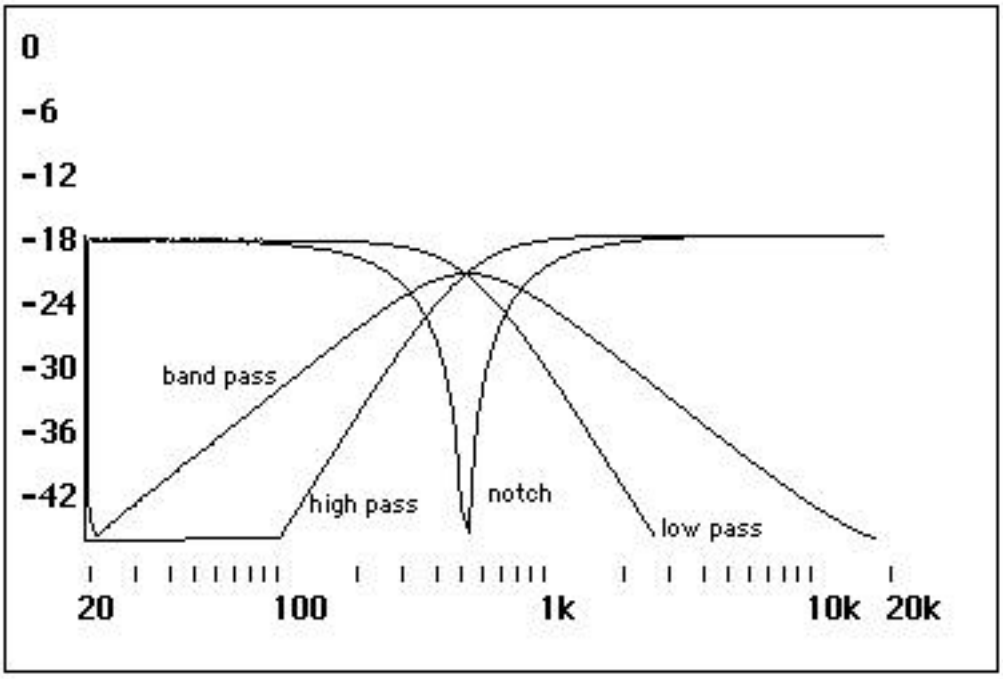


Fig 9. State Variable Filter outputs

The frequency parameter specifies the center of the bandpass, which corresponds to the turnover of the high and low pass sections and the notch. The resonance setting was zero for this plot - figure 10 shows it at 0.75:

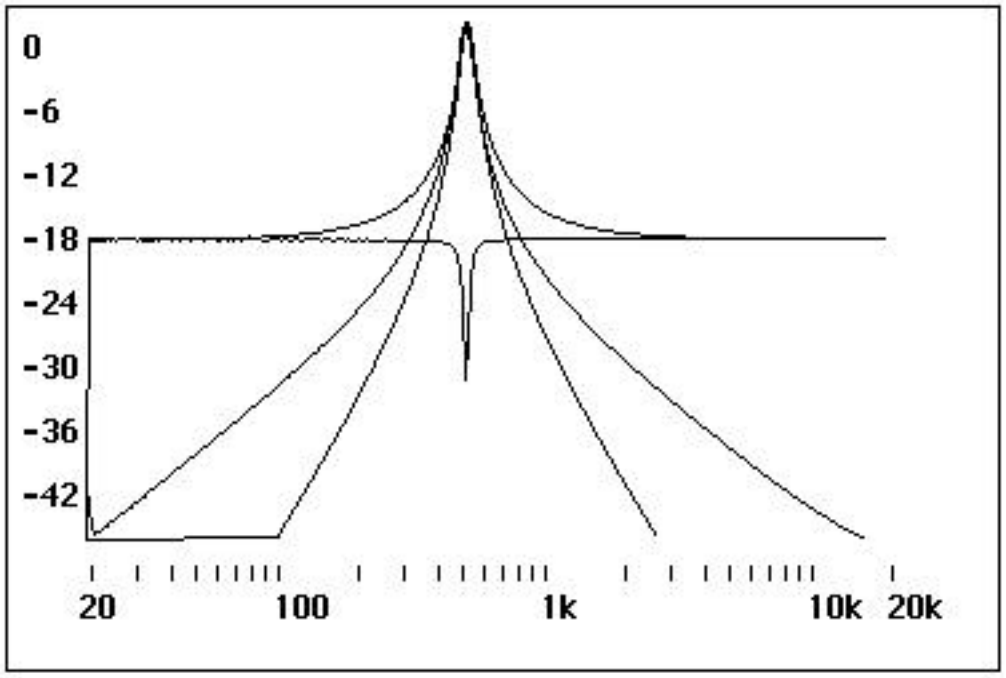


Fig 10. Resonant State Variable filter

These are nice peaks, and there is no interaction between frequency and Q. A comparison of various frequencies bears this out:

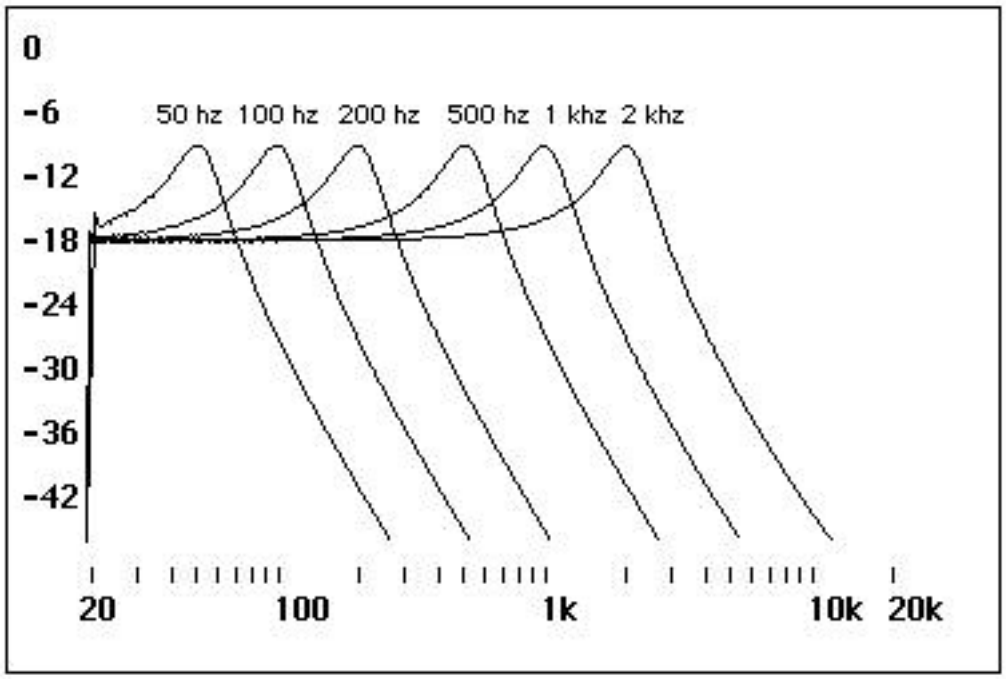


Fig 11. State Variable Filter with constant Q

The Q bump is the same width for any frequency on the logarithmic scale.

The State variable is useful for just about any general filtering job. You can cascade two for a steep rolloff and they won't even register on the CPU meter. It is quite effective for low pass and high pass gating, although you still need to add an amplitude envelope to shut the sound off entirely.

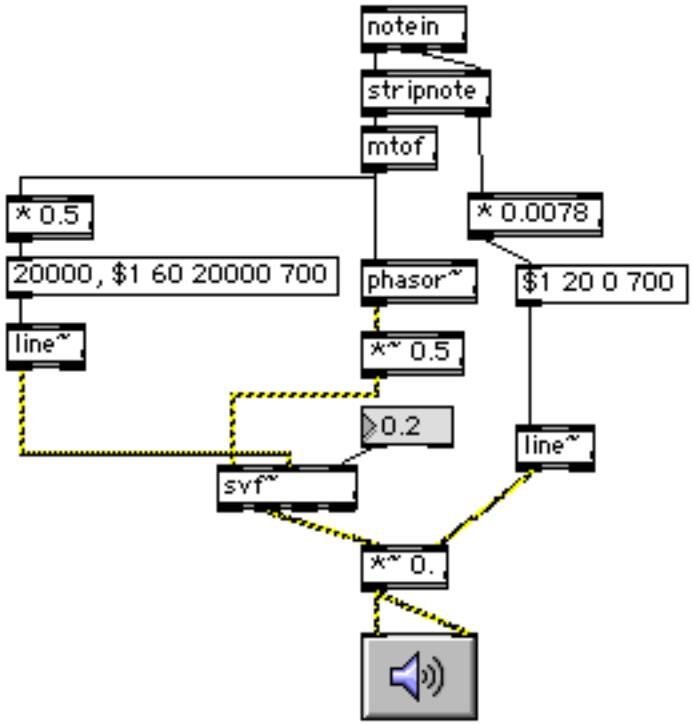


Fig 12. High pass gating with SVF

One interesting feature of state variable filters is that with high Q, they will ring on their own when pulsed. Figure 13 illustrates this. When a noteon comes in, there is a vibraphone like sound.

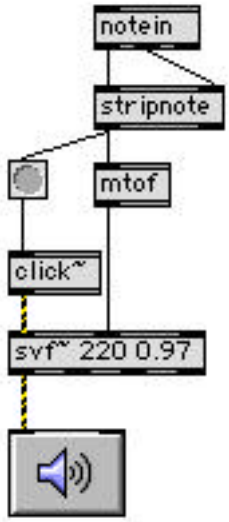


Fig 13. SVF as a tone generator.

**Biquad~ and Filtergraph~**

Biquad is a do it yourself filter for those who want to really get into the numbers. Luckily for the mathematically less agile, MSP 2 features a fine design tool for biquad. This is filtergraph~.

The biquad filter is a building block that enables you to construct custom frequency responses. It will be easier to understand if we look at how the filter actually works.

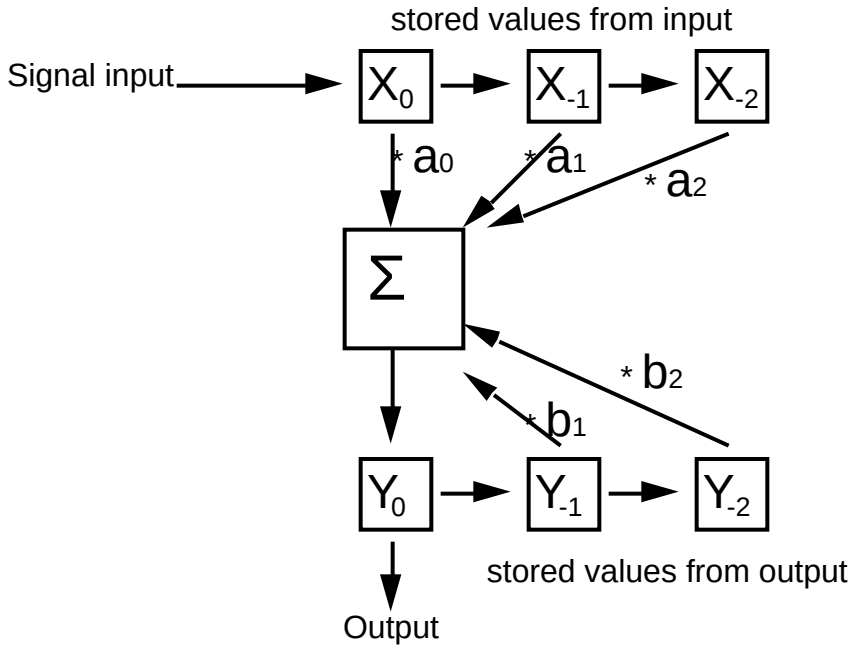


Fig 14. flowchart of biquad filter

In figure 14, the signal arrives as a series of samples. These samples are used immediately and stored to be used again later. The boxes $X_0$ represents the newly arrived sample and $X_{-1}$ and $X_{-2}$ represent the stored samples. Likewise, the filter process produces a series of samples, which are passed along to the next module and stored to be used again. $Y_0$, $Y_{-1}$, and $Y_{-2}$ represent these output samples. You can think of the samples as moving horizontally from box to box. The process is really very simple: each of the samples mentioned is multiplied by some coefficient and they are all added together to produce $Y_0$. The coefficients used on the input samples are called $a_{something}$ and the coefficients for the output samples are $b_{something}$.

In digital filterspeak the equation for this is

Y[n] = a0 * x[n] + a1*x[n-1] + a2*x[n-2] + b1 * y[n-1] + b2 * [n-2]

The terms derived from storing input are called "zeros" and the terms derived from storing output are called "poles". Most digital filters can be described this way. For instance the tapin- tapout pair just has a zero at x[n-220] (It should be 220.5 for 5 ms at 44.1 khz sampling rate, but you can't have delays of part of a sample.)

Biquad implements a two pole, two zero filter. The filter type, frequency, gain and Q are all determined by the coefficients. Unfortunately, the relationship between, say, $a_2$ and frequency is not simple and straightforward. All of the coefficients interact in a very complex way[1] to set the filter properties. It takes a computer program to work it all out. Filtergraph~ contains the program.

Looking at the filtergraph~ help file will give you an idea of what it can do. Start with the display on the right of the screen -- choose a filter shape, then play with the frequency and gain boxes. You will see how the shape can be changed. You can also change frequency and gain with the mouse. Click on the curve near the turnover point, and the point will follow the mouse. The corresponding coefficients are shown at the bottom of the window. Note that all change as the frequency and gain are moved. These can be dumped directly into biquad~ so you can listen to the effects of the changes. In fact, the help window does just that.

If you select filtergraph~ and hit command I, an info window will appear. This has a lot of important settings, starting with the type of filter you are designing. Other highlights:

- Gain enabling allows the mouse to change gain when moved up and down. If this is not checked, gain will remain at 1.0.
- The constraints limit the range of the frequency and gain parameters.
- The frequency display sets the range of frequency that will be shown, as well as choosing log or linear display.
- A full spectrum shows how the frequency response is wrapped into the negative side of the spectrum. It's an interesting theoretical trick.

---

[1] It's like this:
With frequency F and resonance Q
First find the angular frequency with $\omega = 2\pi F / SR$
Find alpha = sin(w) / 2*Q)
Then for a lowpass filter
A0 =( (1-cos(w)) / 2) / (1+alpha)
A1 = 1-cos(w) / (1+alpha)
A2 = A0
B1 = - 2 * cos(w) / (1+alpha)
B2 = (1-alpha) / (1+alpha)
(thanks to R Dudas for the math)

- The amplitude display lets you set the range of amplitude values that will fit on the display. You can also choose linear or log (dB) response
- Phase response can be as interesting and important to know as amplitude response. Take a look at the phase response of any notch setting.
- Output coefficients on load lets you save filter settings with the patch.
- Colors are always fun to play with.

The graph on the left of the help window will follow along with your changes, as it is in display mode, converting coefficients to a response curve. If you get coefficients from somewhere else, such as a book on digital filter design, display mode will show you what to expect.

Display mode can also show you what will happen when biquad filters are cascaded. Double click the [p cascade] box in the help window to get an example. This represents an EQ made of a low shelf, four peak / notch and a high shelf filter connected in series. To hear this, you would send the coefficient list from each of the small filtergraphs to a set of biquads wired left to right. The cascade command to a filtergraph in display mode will make it combine the responses graphically to reflect the effect of a series connection.

If you cascade two or more biquads with the same coefficients, you get a generally steeper cutoff slope. In theory, you should be able to adjust the frequency and gain of each filter slightly to change the corner response (that's what happens right at the cutoff frequency), but it doesn't work well and doesn't make a whole lot of audible difference.

You can use filtergraph and biquads for sweeping and gating effects. Use line (instead of line~) to apply envelopes directly to the frequency input. The graphic display may not keep up[2], but it won't affect what you hear. Two cascaded biquads can give a nice Moog-like sound as shown in figure 15.

[2] In fact, you may want to hide the filtergraph~ in a subpatcher to save space.

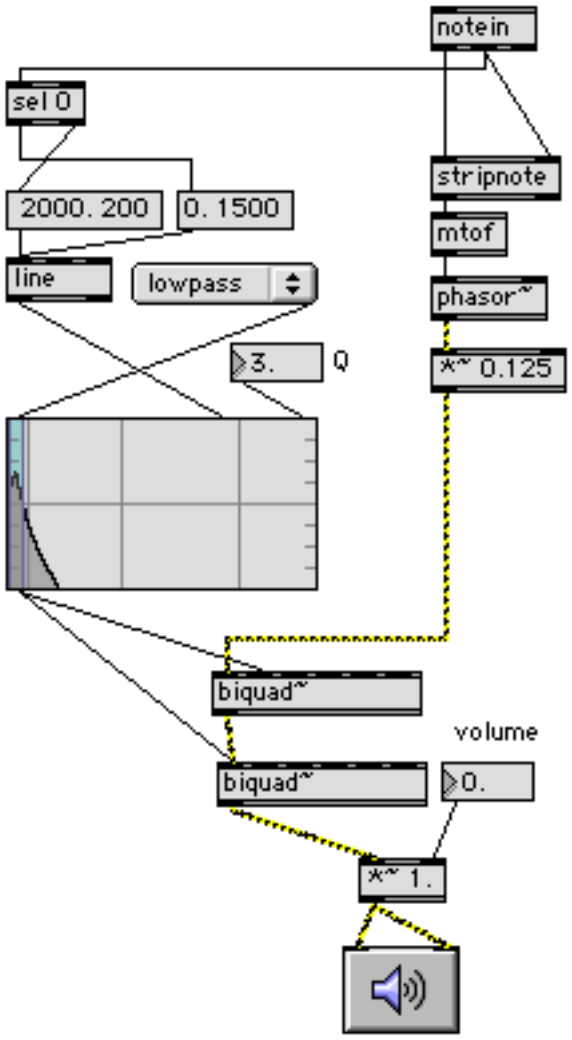


Fig 15. Gating with biquads.

### Allpass~

Allpass is an interesting filter. It does not change the frequency response at all -- it's dead flat. What is interesting about it is the Phase Response. Most filters change the phase of the signal in some way. Generally the change is fairly subtle, at least in the passband. Allpass~ changes phase in a very dramatic fashion. Figure 16 shows the output of allpass~ added to the original input to give an indication of how the phase changes.

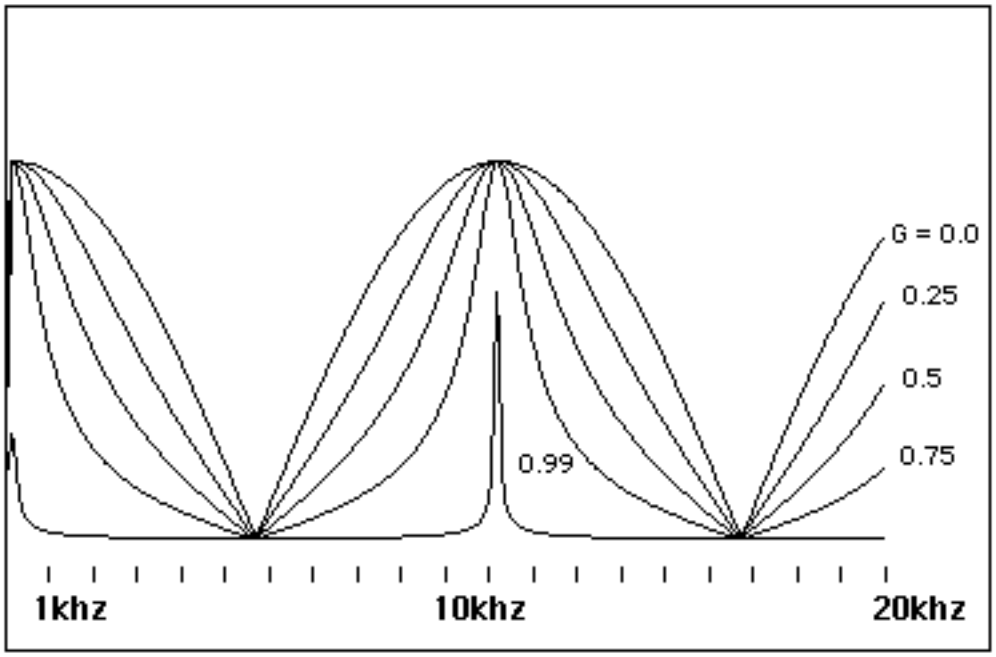


Fig. 16 AllPass~ phase response at various levels of gain

This plot was taken with a delay time of 4 samples. If you follow the top curve as the frequency increases, you will see that the phase changes from 0 to 180 degrees (0 to π radians) at 5.5 khz. It continues to change to 2π at 11 khz, is 3π at 16.5 khz, and would be 4π at 22 khz if the graph went that high. Figure 16 shows only the amplitude of processed + unprocessed signal, so the drawing is folded back into itself. This is similar to the tap delay example - in fact it's the same at a higher frequency. Increasing the gain makes the phase changes more sudden., which sharpens the response peaks in this example. We can't hear phase on steady sounds, but with transients there are some mysterious lingering tones as the help file demonstrates. Even better is to change the delay time, as figure 17 illustrates.

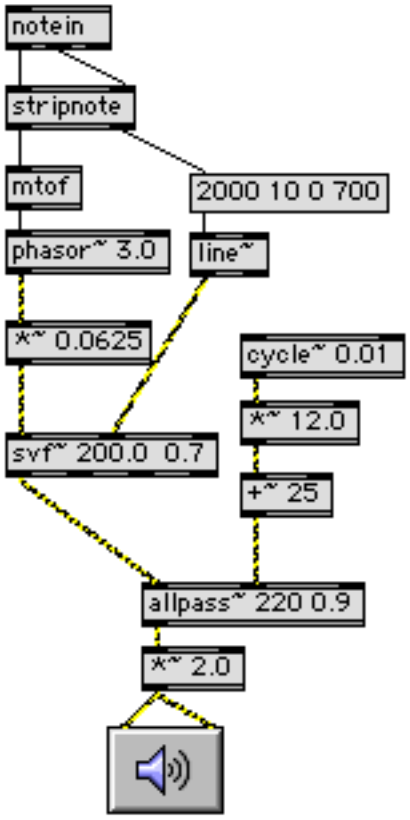


Fig 17. Flanging with allpass.

### Comb~

The comb~ filter is similar to allpass~ but adds some features. First, feedforward (the trick of adding the original to the processed signal to get the phasing effects) is built in. Second, there is also feedback, which runs some of the processed signal through again. (as with any feedback, levels have tpo be watched careflully. Here the limits are -1.0 to 1.0). figure 18 shows the effects of feedforward and feedback.

With no feedforward or feedback, you get a nice flat allpass response. Adding feedforward gets the same deep notches we have seen before. Feedback alone is similar, but you just get dips instead of notches. Negative values for feedback flip this curve upside down. With both feedforward and feedback, the peaks jump through the roof.

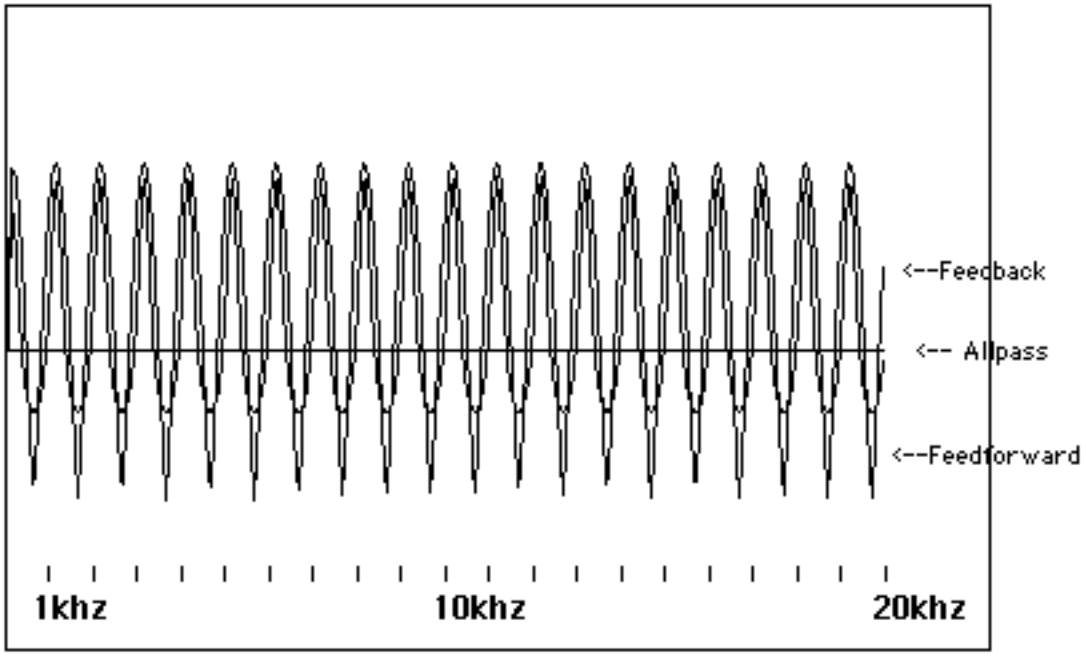


Fig 18 comb~

With the feedback up, there is a lingering effect as the echoes of any input die down. In fact, with a pulse input you can make an instrument with a very rich twangy sound.

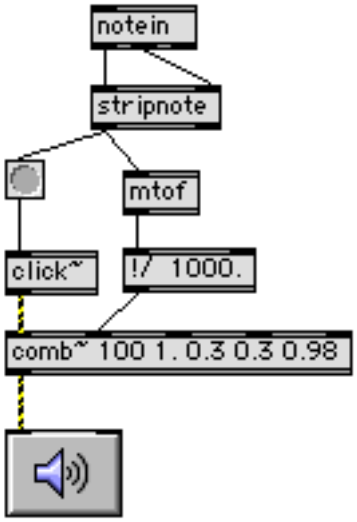


Fig 19. Comb~ as a signal generator.

In figure 19 the frequency is converted to a delay time with the !/ 1000. object.

### Summing up

It's easy to get involved with the minutiae of filter specifications and math, but the important thing about any filter is how it sounds. You should try them all in all of the major applications:

- Coloring synthesized sounds
- Gating synthesized sounds
- EQ of input sounds
- As resonant objects
- In analysis, such as pitch detection or vocoding.

I have my own ideas about what works best in each of these situations, but you should come to your own conclusions. Just start with low levels and modest parameter values until you get a handle of what can be pushed and which way.