21M.370 Digital Instrument Design

Lab 4a - Mar 2

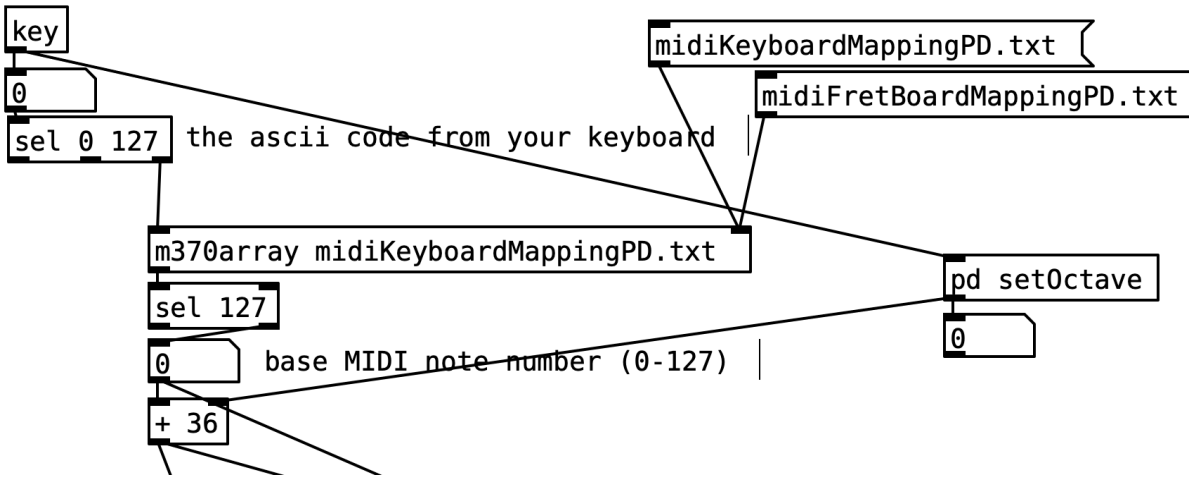
This week we will be exploring working with sensor signals and mapping them to synthesis parameters. Our goal for today is to look at some common approaches to generating control signals:

1. Mapping discrete inputs to lists of pitches
2. Working with continuous data to control amplitude
3. Mapping continuous data to other parameters.

The files we will be using are in NIME/Puredata/MappingExamples.

Part 1: Working with lists

It is common to map discrete values (like keys or valves) to select pitches. Open the 'pitchMappings' file. At the top are some objects we can use to do this mapping.



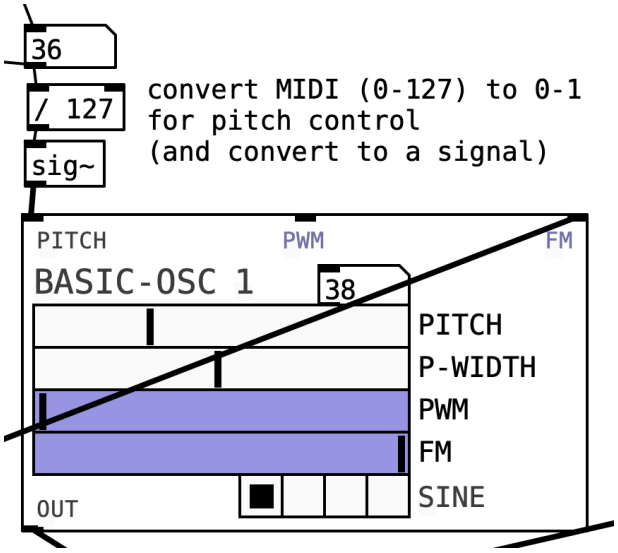
1. m370array is the main object we will use. We will send it .txt files with pitch mappings, and then use the ascii codes from our laptop keyboards to select pitches.
2. Click the ‘midiKeyboardMappingPD.txt’ message box. (Note the curve on the right side indicates it is a message box, not a standard object).
3. Press the bottom row of keys (z-slash) on your laptop. The ‘key’ object sends out the ASCII values of the key, and you should it as well as a base MIDI note.
4. This mapping uses the bottom two rows of your keyboard to emulate a piano layout. Note the black keys are on the upper row, and some of the ascii keys are not uses to simulate white keys with no black keys between them, (e.g. E and F).
5. You can use the minus and equals keys to switch octaves.
6. Click the ‘midiFretBoardMappingPD.txt’ message box to switch to a mapping like a guitar or bass, where each row of keys is chromatic, and the rows ascend by fourths. If you play guitar, type ‘z’ ‘d’ ‘e’ to play a power chord (c g C) or ‘z’ ‘b’ ‘d’ to play a major chord (‘c’ ‘e’ ‘g’).
7. Look in the NIME/puredata/externals folder and you can see the .txt files with the mappings. The format is <ASCII code> <MIDI note> <laptop key as a hint>.
8. You can make your own mapping, save it as a .txt file, and then load it in m370array. . . what would a good mapping be?
9. How might we emulate the way a trumpet works? (e.g. the final note is a function of the key combinations rather than being mapped one note per key)

Examining the patch

Let's take a look at the rest of the patch.

1. This patch uses 3 VCOs (oscillators) with a few different timbral modifiers.
2. The amplitude of these VCOs is modulated by two Decay modules controlling VCAs (amplifiers).
3. The sum of the VCAs goes into a VCF (Lowpass filter).
4. The output of the VCF goes directly to the Maestro module, which sends it to your speakers.

Each VCO has a pitch input. We are sending MIDI Note values, which are integer values from 0-127, representing pitches. We need to convert these MIDI Notes to a normalized range of 0-1. This range of 0-1 for pitches is very unusual, but is trying to emulate the way that analog synths use voltages to control pitch.



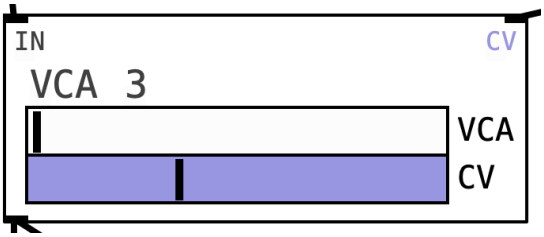
We also need to convert the final pitch to a signal in order for the automata modules to accept them. This is true for all of the automata modules - they all need signals (rather than ints or floats). To do this conversion we just use a 'sig~' object. Remember, the '~' indicates that an object is working with signals.

VCOs have several parameters:

1. pitch controls the base pitch (and represents the pitch as a MIDI note number)

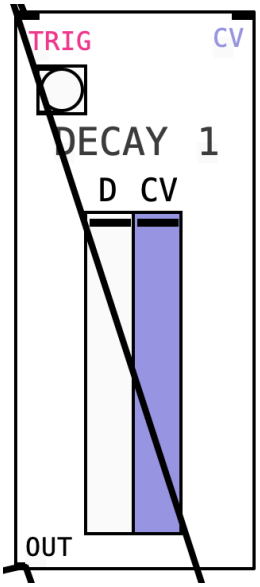
2. P-Width is pulse width, which won't work for all waveforms but basically modifies the waveforms shape and sound.
3. There is also a selector at the bottom to choose the waveform for the VCO.
4. The purple boxes don't do anything directly, but serve as attenuators for the two CV inputs. Note the purple inputs are PWM and FM, and there are two similarly labelled purple faders. If there is a signal coming into the CV inputs, the corresponding slider will act as either an attenuator (all the way left is 0). If no CV is coming in, the purple slider will do nothing.

VCA and Decay



Each VCO goes into a VCA. The VCA controls the volume of the VCO. The white slider is a normal volume slider - if you turn it up you will always hear the VCO. If you want to have the VCO go to silence, you need to keep the white slider all the way down and use the CV input and purple slider.

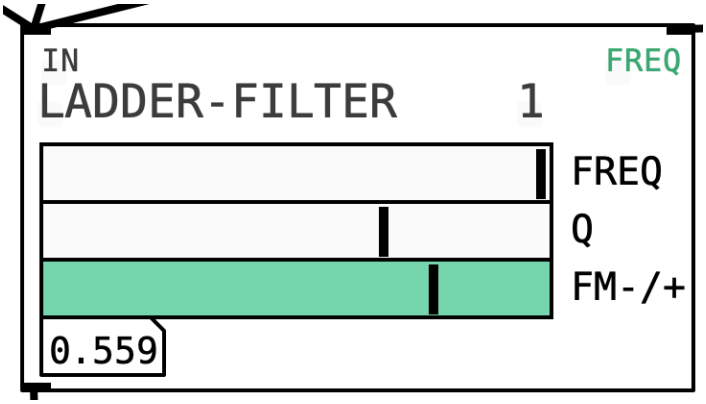
Two Decay modules are used to generate an envelope, which we use to control the volume at the VCA. The envelope always ranges from 0-1, and has a controllable decay time. Experiment with the CV slider on the VCA and the decay time to see how they work together. Remember, if nothing is going into a CV input the purple slider doesn't do anything.



VCF

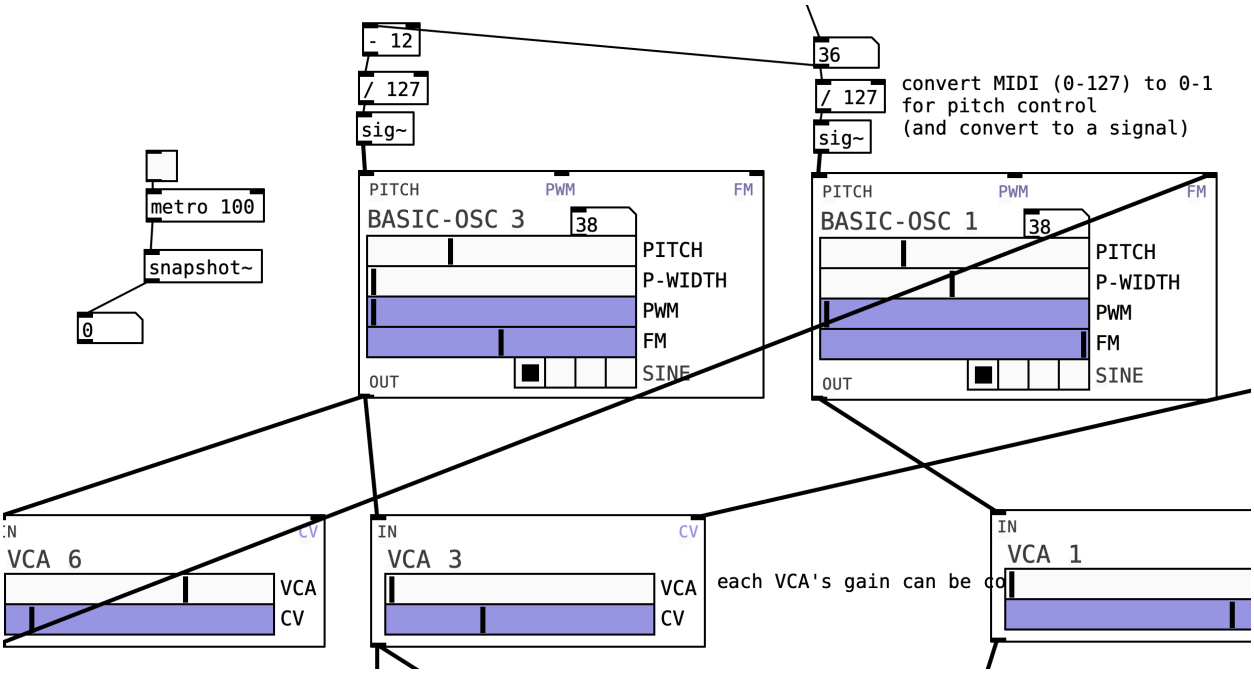
The output of the VCA modules all go to the VCF. The VCF (Ladder-Filter 1 in this patch) attenuates frequencies above the filter's

cutoff frequency. You can move the Freq slider to directly change the cutoff, and also use the Freq CV and FM slider. Note the FM CV input and slider are green. In automatonism, this means that they are bipolar. The slider acts as an attenuverter, in which the middle of the slider is 0, all the way right scales the incoming CV by 1, and all the way left scales it by -1, which also inverts the CV signal.



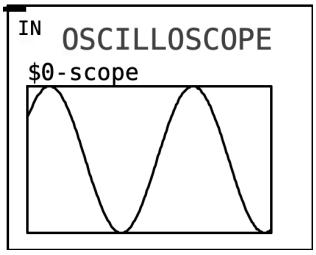
A decay envelope is connected to the Freq CV input. It functions exactly as the other decay modules, but now instead of controlling the volume of a VCO it controls the filter frequency. Try setting the main filter freq (white slider) low and turning the green FM attenuverter all the way up and all the way down. Can you hear the difference?

The ‘Q’ control is a resonance control for the filter. With higher values of Q the cutoff frequency of the filter is amplified, creating a characteristic synthy sound.



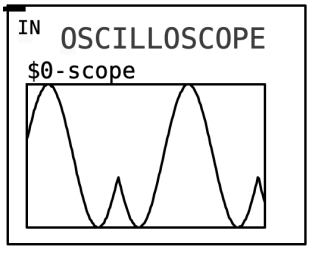
Frequency Modulation

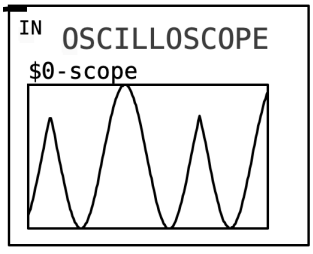
Note that VCO 3 is connected to two VCAs. VCA 3 is used to control the amplitude of VCO3, but VCO6 is used for something totally different. The output of VCO6 is sent to the FM input of VCO1. What this means is that the waveform of VCO3 directly modulates the frequency of VCO1. This is a characteristic form of synthesis called Frequency Modulation, and can be quite a strong effect. Try turning up the purple CV control on VCA6 as well as the purple FM slider on VCO1 to hear what this can sound like.



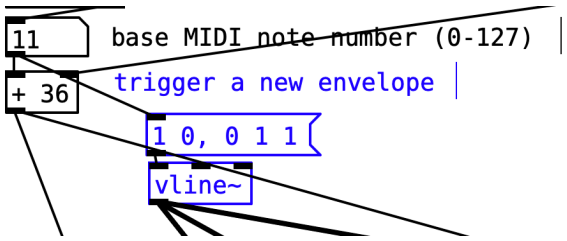
Wavefolding

VCO2 goes into a different signal processing module - a wavefolder. This takes the peaks of a waveform and reflects them, and looks much like you might expect. Take a look at the oscilloscope snapshots above.





Creating Triggers in Automatonism



Triggers are very useful for triggering envelopes or working with sequencers or other rhythmic processors. A trigger in Automatonism is a signal which jumps from 0 to 1 and then immediately back to 0. This patch creates a new trigger every time you play a note.

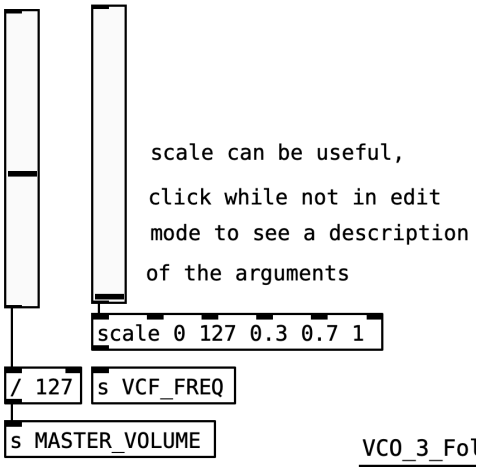
Like everything in automatonism, triggers are signals (e.g. audio signals) and are created by '~' objects. Automatonism has modules dedicated to creating triggers (like the clock modules) but if we want to create triggers using controllers we need to create them ourselves. Doing this is easy:

1. The vline~ object creates an audio ramp
2. We send it a message '1 0, 0 1 1'. The values of this message are: <destination value> <time to get there> <comma indicating a separate command> <destination value> <time to get there> <time to wait untiil triggering this second command>.
   This message can be translated as:
   1. A first command: Go to a value of 1 in 0 milliseconds.
   2. A second command: Then after 1 millisecond has passed, go to a value of 0 in 1 millisecond.

vline~ is useful for lots of things - take a look at its helpfile to learn more about it.

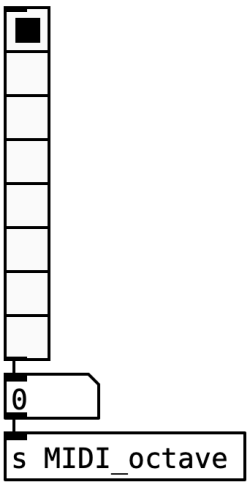
Working with continuous values

Open ‘modificationControls’ patch.



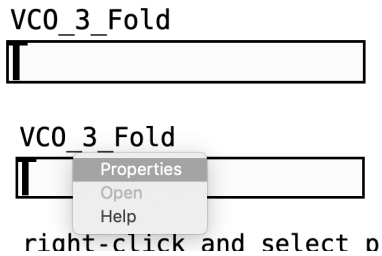
This patch is the same as the previous one, but now we have some user interfaces objects allowing us to change the parameters of our synthesizer. We are using sliders and radio boxes, which are standard PD objects. You can find them under the ‘put’ menu in both horizontal and vertical flavors. Right clicking on the objects allows you to set their parameters.

By default, slider objects output values fro 0-127. For Automatonism we will always need to convert these to values of 0-1, and then turn them into signals. The image above shows two ways to convert them to 0-1: a simple math object, and a scale object. The scale object allows you to convert an input range to an output range, and also has a fifth argument to let you set the exponential curve of the output range. Scale is an object I made - clicking on it will open it up and you can see how I made it, and a description of the arguments.

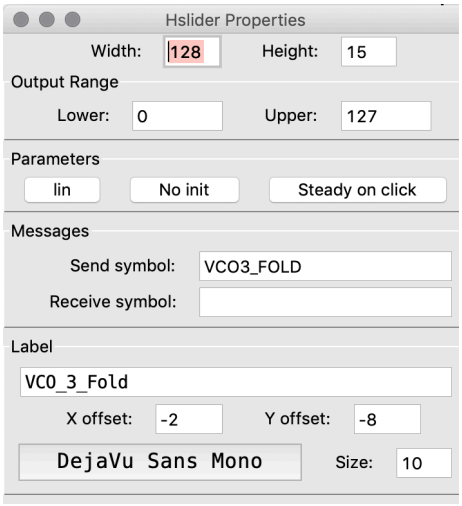


The radio box is a simple selector, which outputs an integer representing the number of the selected box. This radio box is used to select the octave of our synth. Note that I am using a ‘send’ or ’s’ object instead of using a patch cable. This can tidy up our patch, but also makes it hard to see exactly where a signal is going. A corresponding ‘receive’ or ‘r’ object with the same name receives whatever is sent to the send object.

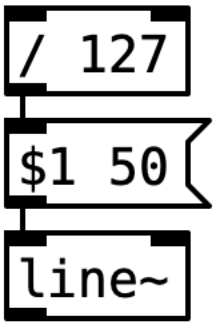
If you right-click on a slider you can directly enter a send name, which will send the slider's value to a corresponding receive object. you can also set a label for the slider, giving it a readable name. Right-click the VCO_3_FOLD slider and you will see how this is done.



Note, for this patch we are converting control signals to audio signals for automatonism in a slightly different way than we were doing for pitch. While pitch is okay to change values instantaneously, for most parameters instantaneous changes are undesirable. Instead, we will create smooth ramps between parameter changes. Two objects can help us do this: line~ and vline~.

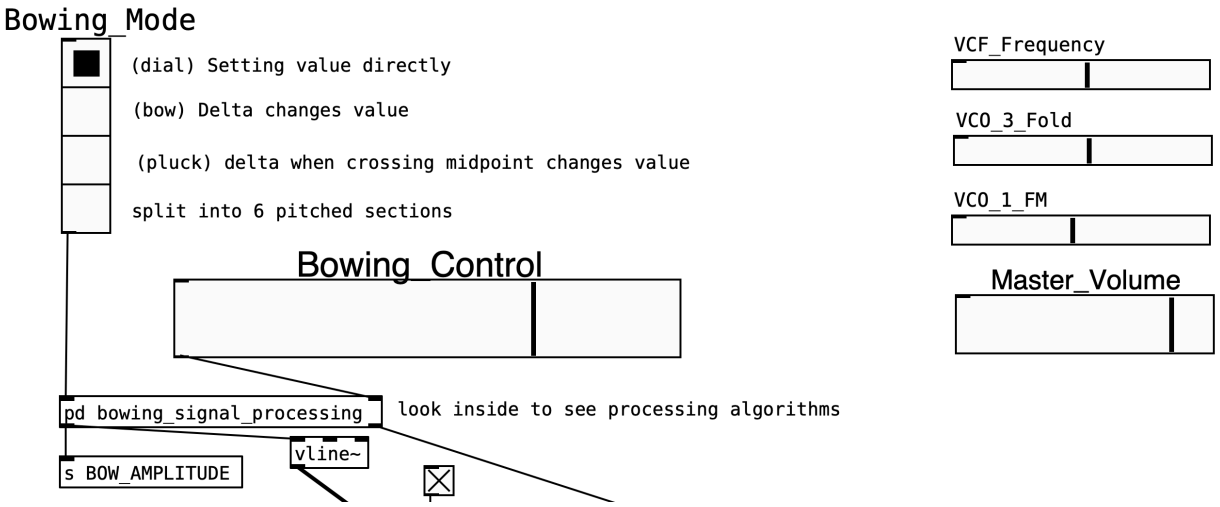


Both of these objects can work with messages consisting of a destination value and a time value. The image to the right expects an input value of 0-127, scales that value to 0-1, and then creates a message with the scaled value and a time value of 50ms to the line~ object. Line~ then makes a smooth ramp going to the value in the indicated time. If we change the 50 to a 500, it would take 500ms to get to our new value.



Creating Excitation signals

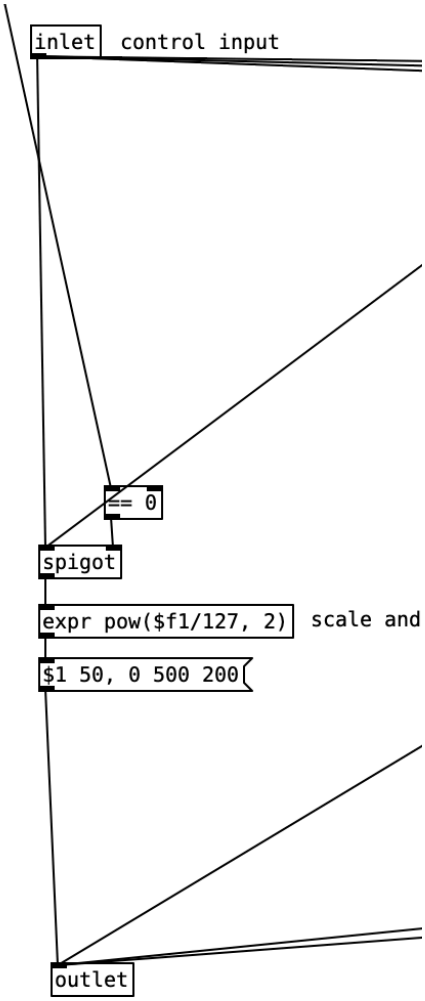
Controlling the volume and intensity of a sound is the most direct form of control, and you will almost always want some way to do this. The ‘excitation’ patch for this week shows several ways to accomplish this. Most of the patch is the same as the other patches, but there is a new element: a bowing control. Use the radio box to select a mode, turn the master volume slider and other sliders up, and then move the bowing slider. The four modes will all function differently- see if you can tell what they do by interacting with them.



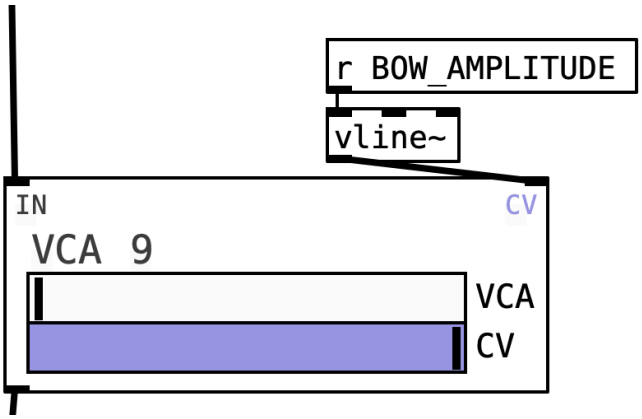
Essentially, all of these modes control the volume of our synthesizer. (For the most part we can also control the pitch using our MIDI mappings as in the first patch). As you continuously move the slider the volume should change in perceptible, hopefully meaningful ways.

Click the ‘pd bowing_signal_processing’ object. This is a subpatch I created. Unlike the scale object, which is saved in the NIME/puredata/externals folder, an object that starts with ‘pd’ is a oneoff subpatch and is saved with the file it is located in.

Mode 1: Simple creation of a ramp setting the amplitude to the new value and then setting amplitude to 0 when the slider is not being moved. Note the ‘expr’ object being used: this is a general purpose math objects and is useful for doing multiple operations within a single object. Here we are scaling the input signal to 0-1 and then squaring it to create an exponential response.

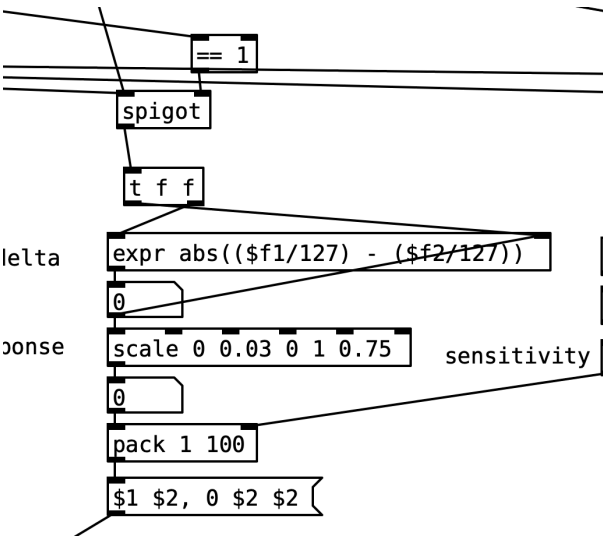


The message ‘$1 50, 0 500 200’ is sent to a vline~ object located right above the VCA 9, where the BOW_AMPLITUDE receive object is.



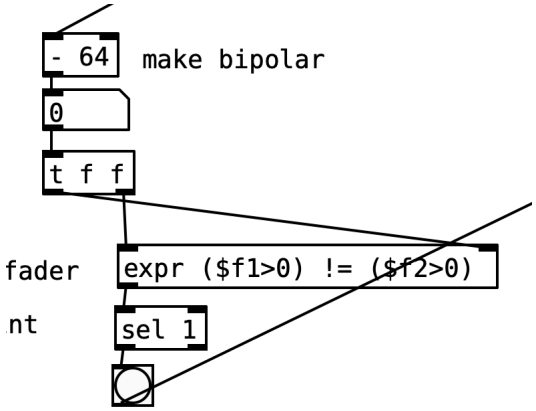
Mode 2: Changes in slider value control volume

Here we take the difference between successive slider values and use this to control the amplitude. Note that the range of the difference will most often be quite small, so we need to scale it correctly and perhaps add an exponential curve to get it to sound and feel right.



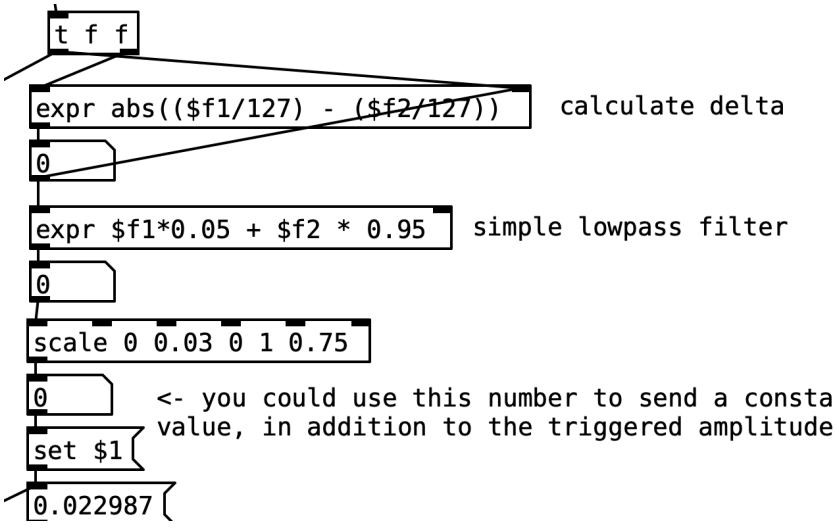
Mode 3: Trigger at the midpoint

This mode is kind of like using a string instrument, where we pluck the string at the middle of the slider. Like mode 2, the amplitude is dependent on how fast we are moving the slider.



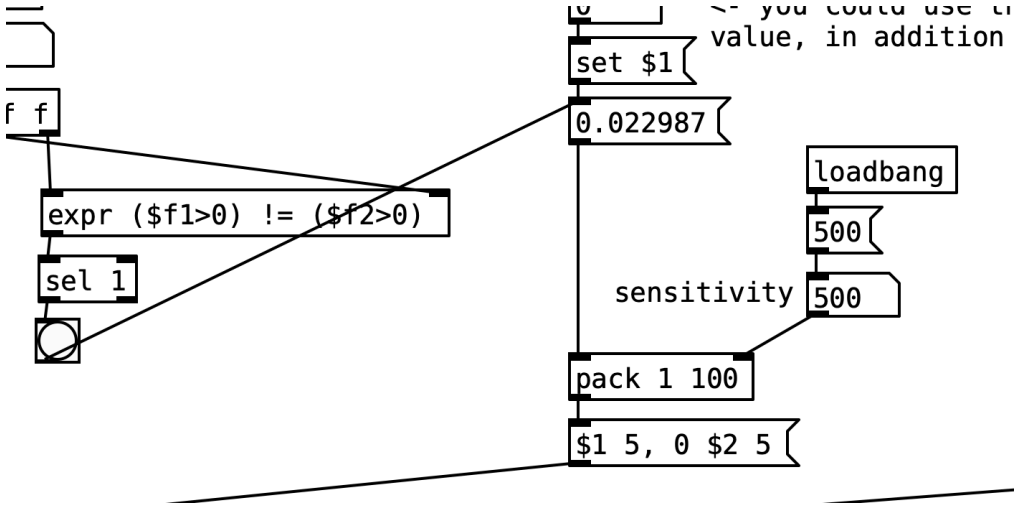
There are two parts to this mode:

1. Make the signal bipolar and look for when the signals moves from positive to negative or vice versa.
2. Track changes in slider value, use a lowpass filter to take the average movement, and

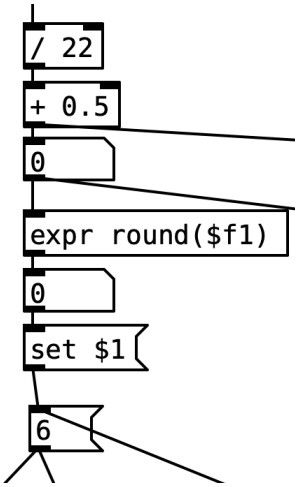


then scale and store that value. Note the ‘set $1’ message will store the value in the message box below, but the value will not be sent out of that message box until it is triggered by the triggering algorithm.

3. When a trigger is detected this will trigger the stored value to be sent to the message box formatting the vline~ message.
4. The sensitivity control determines how it takes for the envelope to go to 0 after a trigger.

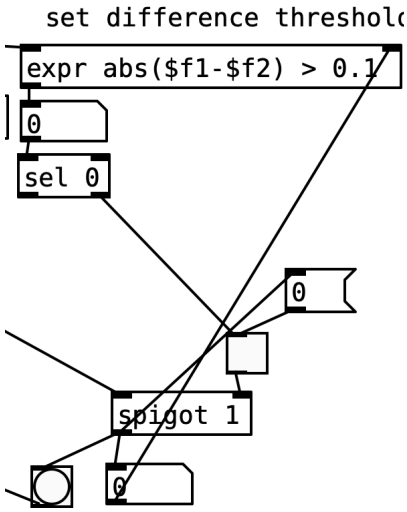


Mode 4: Segmenting the slider into different pitches
This mode chops the slider range into 6 segments, and then triggers a new note when we pass to a new segment. The image to the right shows the segmenting process, again using the ‘expr’ object.



Note that once a note has been triggered we prevent another note from being triggered until the slider has been moved at least a little bit. This prevents accidental retriggering when the signal has small fluctuations right near the threshold.

To do this, we set a threshold. Once a note has been triggered and passes through the spigot, the ‘0’ message closes the spigot until the difference between the new value and the current slider value exceeds the difference threshold. When it does, the spigot is opened, allowing a new note to be triggered.



When a new note has been triggered the segment number is sent to a select object, which sends a bang out the appropriate outlet and triggers a MIDI note.