Comp 150 - Music Apps for the iPad
Lab 1: Building a Sequencer in Pure Data

In this lab you will build a sequencer in Pure Data with a basic wavetable synthesizer as an audio source. The goal of the lab is to get you familiar with Pure Data and the tools you'll be using for the first project.

For this lab you will want:

1) An installation of Pd-extended - you can find one at www.puredata.info. Pd-vanilla will also work, but you will be able to use externals from Pd-extended in your iOS project later on if you like. There is a comprehensive list of objects for both versions at:

   http://en.flossmanuals.net/pure-data/list-of-objects/introduction/

2) The Pd-for-LibPd Library. This is a collection of Pd-Vanilla-compatible patches that offer extended functionality to save you time:

   http://www.github.com/cpenny42/Pd-for-LibPd

   To get the git repository on your computer, copy the clone URL from Github page, open your terminal, and navigate to the folder where you'd like to download it. Type the following command to make a local copy onto your computer:

   ```
   $ git clone https://github.com/cpenny42/Pd-for-LibPd.git
   ```

   Next, launch your installation of Pure-Data and link it to the /source folder by adding it to Pd's search path. You can do this in Pd->Preferences. You should now have access to all the patches from the Github repository. The patch *pd_for_libpd.pd* shows you most of the objects in the library.

Plan of the lab

We will start by making a very simple sequencer that can play 4 notes. We will then expand it to be able to sequence any number of notes as well as synchronize a tremolo effect to the sequencer's internal metronome.

Basic Elements

In addition to the link above, you can find all of the core objects by right-clicking and selecting “help”. The Pd FLOSS manuals are also a great resource: en.flossmanuals.net/pure-data/

The object [keyboard_shortcuts] has a convenient list of all the available keyboard shortcuts in one place.

Keep in mind that Pd only lets you undo & redo once, so you need to be careful about accidentally deleting sections of a patch. Save often & make backups so you don't lose any of your work.

Part 1 – A basic sequencer

Create a new patch and save it as lab1-A.pd.

In the app menu, click Media Audio Settings. Make sure the output device is configured to play from your speakers.

1. **Create a switch**

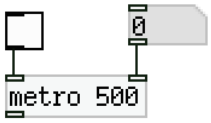
   Put a **[toggle]** in the patch. (Windows: ctrl-shift-T, Mac: command-shift-T). We're using a toggle in this example, but there are many ways to turn your sequencer on & off.

2. **Regulate the timing**

   a) Put a **[metro]** object with a creation argument in the patch - it will send a bang periodically to regulate timing. All timing in Pure-Data is in terms of milliseconds, so an argument of 500 would put the metronome at 120 BPM (500 == 2 beats / second == 2 * 60 beats / minute).

   b) Connect the outlet of the **[toggle]** to the left inlet of the **[metro]**

   c) Add a number box (ctrl/command-3) and connect it to the right inlet of the **[metro]**. You can now change the metronome's tempo. Your patch should look like this:



3. **Set up a counter**

   a) Put in a **[float]** (or **[f]**). This stores the number in the right inlet and outputs the result when it receives a bang.

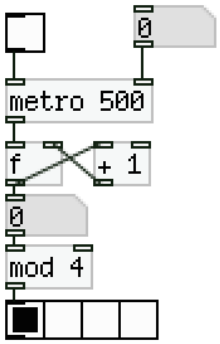
   b) Connect the outlet of the **[metro]** to the inlet of the float.

   c) Add a **[+ 1]** object.

   d) Connect the outlet of the float to the left inlet of the **[+ 1]** and the outlet of the **[+ 1]** to the *right* inlet of the float. This will store the incremented number in the **[float]** object. If you connect the **[+ 1]** to the *left* inlet of the **[float]**, you will get a feedback loop (a.k.a. stack overflow).

   e) Put in a number box and connect it to the float. You can now see your counter work when you click the toggle switch.

   f) Add a **[mod 4]** object and connect it to the number box above.



   g) Put a horizontal radio in the patch (Put Hradio, or command/ctrl-shift-I). Right-click it to see its preferences, set the *number* to 4, and connect it to the output of the **[mod 4]**. You will now cycle through a loop of 4 beats.

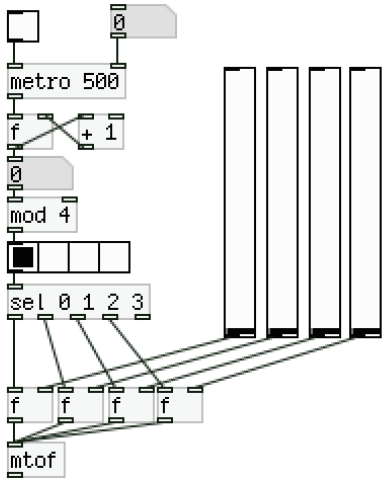
4. Set up a distribution channel

a) Put in a **[select 0 1 2 3]** object and connect it to the Hradio's outlet. A bang will be sent out one of the left four **[select]** outlets when 0, 1, 2, or 3 are sent to the left inlet. Other input will be routed through the right outlet.

b) Add four **[float]** objects – one for each note in the sequencer. Connect the left inlets of each **[float]** to the left outlets of **[select 0 1 2 3]**.

c) Put a vertical slider (Put Vslider, or command/ctrl-shift-V) in the patch for each step in the sequencer, and connect each to the right inlet of a **[float]** object. The default range for sliders is 0-127, so this will allow you to set any note from 0 to 127 in each step. You can change this range or the slider's appearance by right-clicking and selecting "Properties".

d) Add a **[mtof]** object at the bottom of the patch, and connect the outlets of each **[float]** to its inlet. This will convert the MIDI note from 0-127 into a frequency in Hz.



5. Send audio to your speakers

a) Put an Array in the patch of size 4099, and name it "wavetable" (Put → Array). This will be the waveform you play to the speakers. We will be reading from the Array with the **[tabosc~]** object, which requires the array size be a power of 2 + 3 (aka 4096 + 3).

b) Draw a waveform into the array. Exit edit mode and use your cursor to draw any shape in the table.

c) Add a **[tabosc4~ wavetable]** object (where "wavetable" is the creation argument), and connect the outlet of the **[mtof]** to its inlet. **[mtof]** will now tell the **[tabosc~]** the frequency at which to read from the wavetable.

d) Add a **[*~ 0.3]** and connect it to the output of the **[tabosc~]**. This will make sure the audio is not too loud. You can add a number box and connect it to the right inlet for interactive volume controls. Note that the "~" in the object name implies that it works on signals instead of floats.

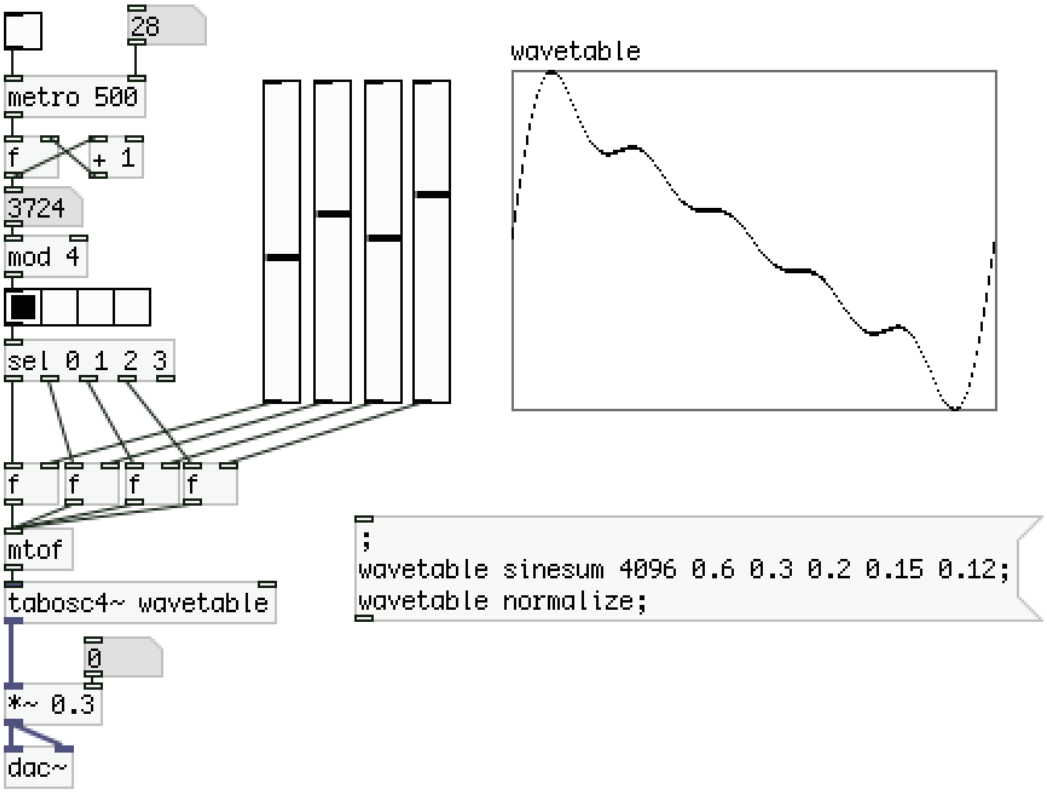
e) Add a **[dac~]** object, and connect the outlet of the **[*~ 0.3]** to both inlets of the **[dac~]**.

If you turn DSP on (command/control-/), you should be able to hear your waveform. The positions of each slider will determine the notes that are played, and the number connected to the right inlet of the **[metro]** will let you change the sequencer's speed. If your table is empty, you won't hear anything.

f) Add a message object (Put → Message, or command/control-2). Type the following in it:

```
;
wavetable sinesum 4096 0.6 0.3 0.2 0.15 0.12;
wavetable normalize;
```

This will put a waveform consisting of a sine wave and four of its harmonics with the amplitudes given in the above command, and then it will normalize the waveform from -1 to 1. Play around with the coefficients to change the sound, or check out the ./sinesum executable in the Pd-For-LibPd folder for more sounds.



/ A note about signals: When you are working with audio, you are actually manipulating a stream of numbers that flow very quickly (usually 44100 "samples" per second). Pure-Data abstracts this away from you by providing a Signal type. Objects that work on signals receive individual buffers of samples that are passed between objects, and math operations such as **[*~]** and **[+~]** will perform the operation across the entire buffer. You can set the Buffer Size in Pd's audio settings – Larger buffers will cause latency, and smaller buffers can put more strain on your computer as it has less time to process the audio. */*

Part 2 – Adding functionality

We are now going to make a more advanced sequencer that is dynamically configurable – while the sequencer above is nice, it is very limited.

Save & Close your patch (*lab1-A.pd*), and make a new patch called *lab1-B.pd*.

When you use Pd in your iOS projects, you will need to communicate with your patches via an interface you define. The first step for Part 2 is defining the interface for the sequencer we will build.

1. **Define the interface**

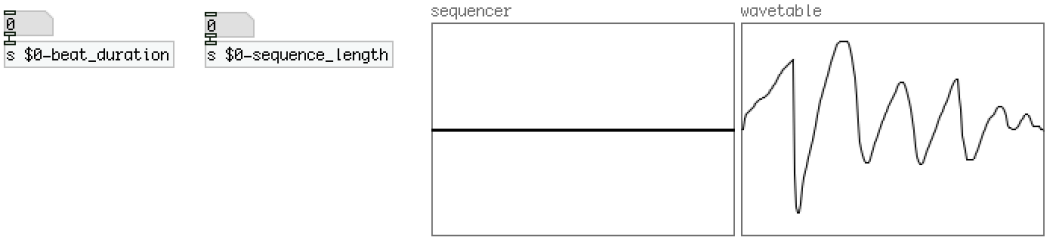
   a) We will be making a sequencer that can hold any arbitrary sequence of notes. The user will be able to change the speed of the sequencer along with the number of notes in a loop. Create the objects **[send $0-beat_duration]** and **[send $0-sequence_length]**, and create number boxes that are connected to each **[send]**. The `beat_duration` will be in milliseconds, and the `sequence_length` will be the number of notes in the sequence.

   / Note about $0: You can access a patch's creation arguments through the dollarsign notation - $1 is the first argument, $2 the second, etc. $0 is a special argument given to each patch that is guaranteed to be unique to each separate abstraction. You can use this to implement **[send]** and **[receive]** objects that are guaranteed not to accidentally communicate other instances of the same abstraction – in this case the patch "lab1-B.pd" - that have already been loaded. */*

2. **Add a Wavetable & Sequence table**

   a) We will implement the sequencer by using an Array to read, store, and iterate through note values. First, create a "wavetable" Array of length 4099 (same as before), and then create a "sequencer" Array of length 124. Draw whatever wave shape you want in the `wavetable` Array.

   The length of the `sequencer` Array will be the maximum possible number of notes in the sequence – in this example we will cap it at 124, though you could make your Array bigger or smaller if you like.



3. **Read from the Sequence table**

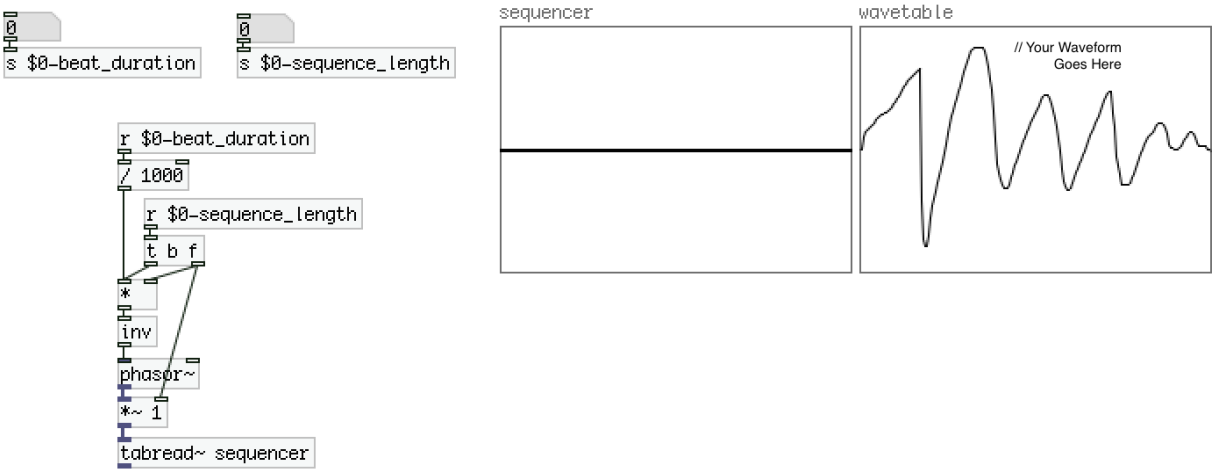
   a) Now we will implement the sequencer. Add **[r $0-beat_duration]** & **[r $0-sequence_length]** to receive the input parameters.

   b) We will use a **[phasor~]** and a **[tabread~]** object to read from the table.

   [phasor~] outputs a steady audio ramp from 0 to 1 at the input frequency (in Hz), so by multiplying the output by the highest index we want to read, we can use these two objects to loop through indeces of the `sequencer` table. Frequency in Hz = 1 / s, where s is the time in seconds for one full loop. Therefore, the formula we'll use to convert the input `beat_duration` and `sequence_length` is:

   `Frequency of` **`[phasor~]`** `= 1 / ((beat_duration / 1000) * sequence_length)`

   This will cause the **[phasor~]** to iterate through the indices of the `sequencer` table chosen by the user. In your patch, add the necessary objects to make it look like the patch on the next page:



The signal coming out of **[tabread~]** will now be the values stored in the `sequencer` table between index 0 and the index set by `sequence_length`.

c) Arrays in Pd store values from -1 to 1 within their borders (though any value can be stored – it just might fall outside the draw box for the Array in Pd). In an iOS app, it doesn't matter how the array would appear when opened in Pure-Data, but for this lab, we want to scale the output of the array to go from 0 – 127 for easier interaction.

Since the default range is -1 to 1 and we want a range from 0 to 127, we will need to transform the output with the formula below:

```
value_out = ((value_in + 1) / 2) * 127
```

Implement this formula to transform the output of the **[tabread~]** object. The answer is below if you get stuck.

You can use the **[view~]** object connected to a number box to see the current value of an audio signal.

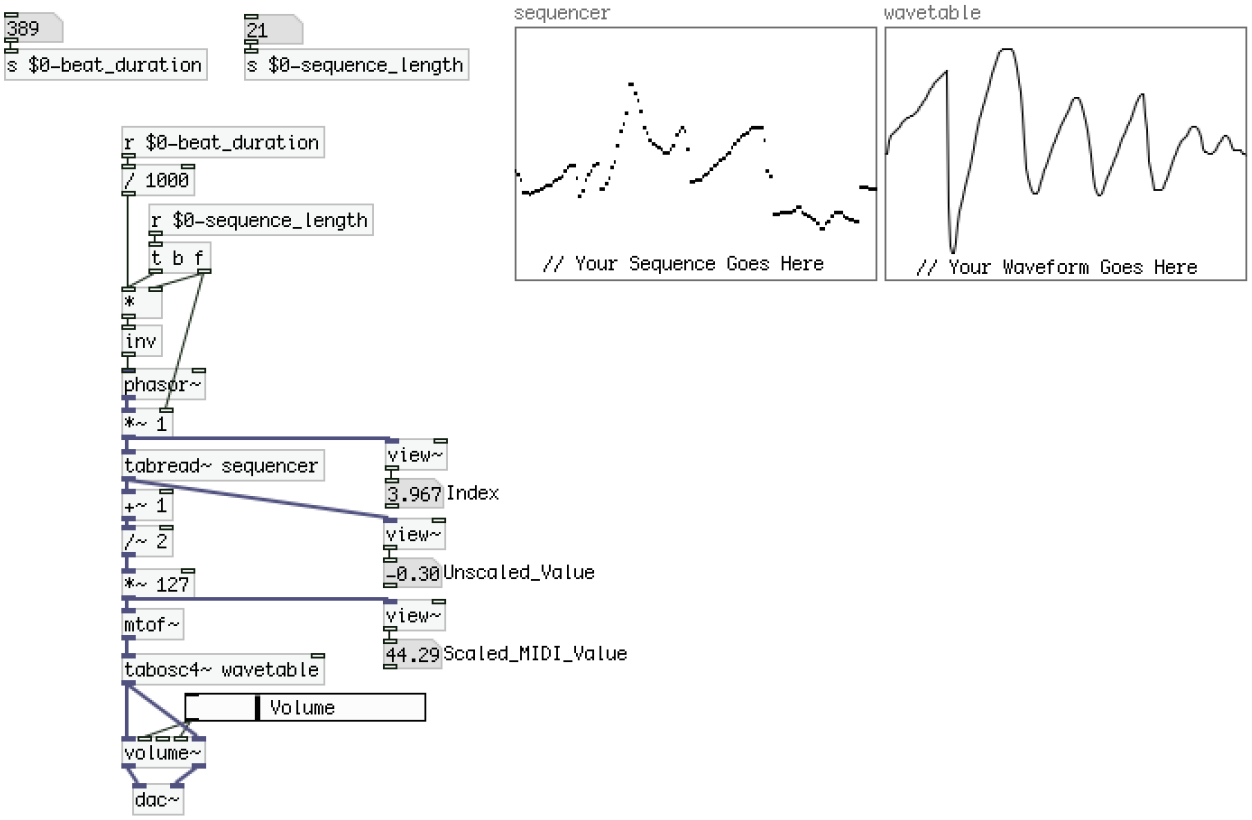
d) Next, add a **[mtof~]** object to convert the `sequencer` output from MIDI note numbers to frequencies in Hz.

e) Add a **[tabosc4~ wavetable]** object to read from your `wavetable` Array at the frequencies stored in the `sequencer` Array.

f) Connect the **[tabosc4~]** to a **[dac~]** object to hear the result.

You won't hear anything if you haven't first set the `beat_duration` or the `sequence_length` or if your `wavetable` Array is empty. Draw in your `sequencer` Array to change the sequence. For easier volume control, check out the **[volume~]** object, or implement your own with **[*~]**

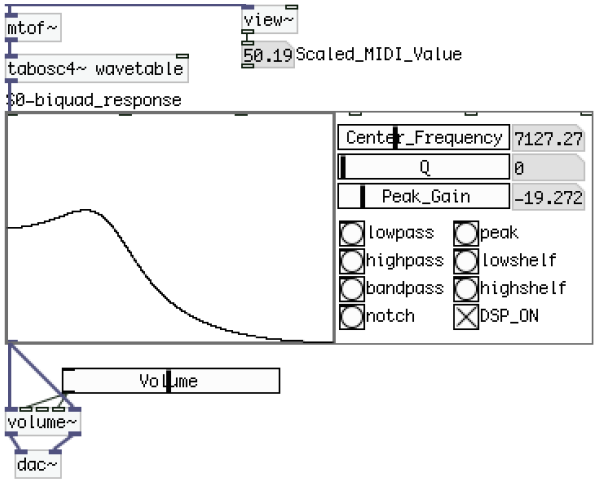
Your patch should look similar to the patch on the next page:



Part 3 – Two Simple Effects

1. You will now add two basic effects for your sequencer, one of which has already been implemented for you.

   Add a **[guifilter~]** object and put it somewhere between the **[tabosc~ wavetable]**. This is a simple Biquad filter that will convert filter parameters to coefficients that **[biquad~]** will accept. We will cover filters more in a future lecture.



2. Implement a tremolo effect that allows the user to set the rate to be proportional to the sequencer's `beat_duration`. This will involve using a **[*~]** object to multiply the amplitude of the sequencer's output signal with another signal, possible from an **[osc~]**. You would then need to set the frequency of the **[osc~]** to cause the tremolo to beat in synch with the sequencer. If you get stuck, the FLOSS manuals have a chapter on Amplitude Modulation: `en.flossmanuals.net/pure-`

data/ch021_amplitude-modulation/