21M.380 Music and Technology
Sound Design

Pd assignment 2 (pd2)
Abstractions

Due: Monday, February 29, 2016, 9:30am
Submit to: MIT Learning Modules ▸ Assignments
5% of total grade

1 Instructions

Write the following abstractions in Pd according to the specifications provided. Each abstraction must be accompanied by its own help patch.[1] Submit your assignment as a single `.zip` archive that includes no directories, one `.pd` file for each abstraction and one for each help patch.

[1] In Pd, help patches are associated with their abstraction by saving the former to a `.pd` file whose name is identical to that of the latter, except that it has `-help` appended to the common base name (e.g., `mypatch.pd` and `mypatch-help.pd`). If Pd finds a help patch of this name in its search path, it will open it automatically whenever the user right-clicks and selects Help on an instance of the abstraction. You can also *pre*pend `help-` to the base name, but I do not recommend it, since doing so will result in abstraction and help patch not being shown next to each other in alphabetically sorted lists (e.g., file browsers).

1.1 BPM-to-ms converter

Names to be used

- Abstraction: `bpm2ms.pd`
- Help patch: `bpm2ms-help.pd`

Desired functionality Translates a BPM (beats per minute) value to a time interval between two beats in ms (suitable for setting the `[metro]` object in a sequencer).

Creation argument

- `$1` represents the initial BPM value. Decimal points should be ignored.

Inlets

- The right inlet accepts integers to override the initial BPM value set by `$1`. Again, any decimal points should be ignored. Changing this inlet should *not* yield any output immediately (cold inlet).
- A `[bang(` message to the left inlet should result in the current conversion result being output at the outlet.

Outlet

- The abstraction's only outlet should produce the time interval in ms that corresponds to the current BPM value whenever it receives a `[bang(` at its left inlet. In addition, it should once output the ms value that corresponds to the initial BPM value specified by `$1` directly after the abstraction is loaded. If no `$1` has been specified, no ms value should be output at load. Under no circumstances should the abstraction generate any output while the current BPM value is equal to zero.

1.2 Sawtooth oscillator

Names to be used

- Abstraction: `saw~.pd`
- Help patch: `saw~-help.pd`

Desired functionality Provides a sawtooth waveform at audio rate that oscillates between -1 and $+1$. Note that this is similar to, but different from `[phasor~]`, which provides a sawtooth signal that oscillates between 0 and $+1$.

Creation arguments

- `$1` specifies the oscillator's initial frequency in Hz and defaults to 440 if not provided.
- `$2` should be either of two symbols (strings), `up` or `down`, which specifies the direction of the sawtooth as shown in figures 1 and 2. The waveform shape should default to `up` if no argument is povided, or if the argument matches neither `up` nor `down`. In either case, an informative warning message should be printed to the main Pd window.

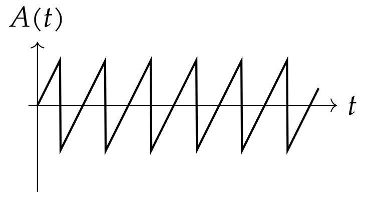

FIGURE 1. Sawtooth wave (up)

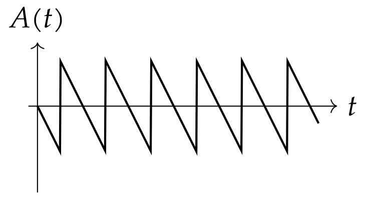

FIGURE 2. Sawtooth wave (down)

Inlet

- The abstraction's only inlet should allow to override the initial frequency specified by `$1`.

Outlet

- The abstraction's only outlet should provide the sawtooth signal.

1.3 Sine object at audio signal rate

Pd vanilla includes a [cos~] object that calculates the cosine of the incoming audio signal *multiplied by* 2π. A corresponding [sin~] object is absent from Pd vanilla, but can easily be derived, which is your task for this abstraction. You can confirm that your abstraction works by plugging it into the patch by Farnell (2010) that is shown in figure 17.5 of the book.

Names to be used

- Abstraction: `sin~.pd`
- Help patch: `sin~-help.pd`

Desired functionality Calculates the sine of an incoming audio signal *times* 2π, analogous to the [cos~] object.

Creation arguments This abstraction does not have any creation arguments.

Inlet

- The abstraction's only inlet is the input audio signal.

Outlet

- The abstraction's only outlet should provide the audio signal that represents the result of the calculation.

2 Assessment criteria

Functionality How closely do your abstractions stick to the specifications outlined above? Do they behave as expected?

Readability of abstraction How easy is it to understand the internals of your abstractions by looking at their code? Have you visually organized the patch in a clean manner? Are there comments to clarify the patch's behavior?

Quality of help patch How well does the help patch illustrate the usage of the respective abstraction? Can the user understand what the abstraction does from a quick glance at the help patch? Is the entire functionality of the abstraction explained in the help patch, including all inlets, outlets, and creation arguments, and the latters' defaults? Can a user actually *try* all of these different behaviors in practice through the help patch without having to edit it?

3 Code documentation guidelines

Students are sometimes confused as to what kind of documentation and comments should go in the help patch, and what in the abstraction itself. Here are some recommendations.

The help patch accompanying an abstraction should showcase the abstraction as a 'black box' to a user who need not be concerned with the details of its actual implementation. It can also serve you as a testbed for debugging purposes as you develop your abstraction.

Documentation inside the abstraction itself should target developers who do need to understand the details of the abstraction's implementation for the purpose of debugging, fixing, or extending its behavior. Often this will include yourself, and it is worth including such documentation inside an abstraction even if you will remain its only developer. Do not underestimate how quickly you will forget how exactly you solved a specific problem!

These recommendations have some more specific implications:

- The help patch should include a verbal description of the abstraction's behavior (very much like the specifications provided above), but it should also allow the user to *test* all aspects of this behavior in actual practice. For this you may well want to include multiple instances of the relevant abstraction, e.g., once with and once without a creation argument, in order to demonstrate default values.
- By extension, the abstraction itself should *not* include any code that merely serves demonstration or debugging purposes. In most cases, such code will just unnecessarily burn additional CPU cycles once your abstraction is put into action. Remove any such superfluous code before you complete and submit your abstraction.
- That being said, it is still meaningful to document the abstraction itself through comments that explain the nuts and bolts that a more casual user is not interested in, but a developer will be.

For example, you might implement a mathematical equation in your abstraction, which can become hard to read once given as Pd code. In these situations, it is meaningful to spell out the equation as a comment, even if just to remind yourself of it.

- As you develop an abstraction, you will typically want to embed an instance of it into a parent patch for debugging and testing purposes. So why not turn this into your future help patch right from the beginning? I recommend that you create a help patch with any abstraction that you start to design. This is much more rewarding than considering the help patch an afterthought.
- Do not duplicate code from the abstraction's internals in the help patch. The help patch's purpose is to showcase its behavior, not how this behavior is implemented.
- Do not write help patches that output any audio. However, if you have a good reason to let your help patch output audio, do not write help patches that output audio at full-scale level and make sure they do not output audio automatically whenever they are loaded.[2]

[2] Imagine a public live-coding performance in which you need to check an object's help patch, and boom... you just sent a 100 dB tone at 1 kHz to the PA system.

References and useful resources

Farnell, Andy (2010). *Designing Sound*. Cambridge, MA and London: MIT Press. 688 pp. ISBN: 978-0-262-01441-0. MIT LIBRARY: 001782567. Hardcopy and electronic resource.