Introduction to Pure Data

DT2140 Autumn 2024

Emma Frid

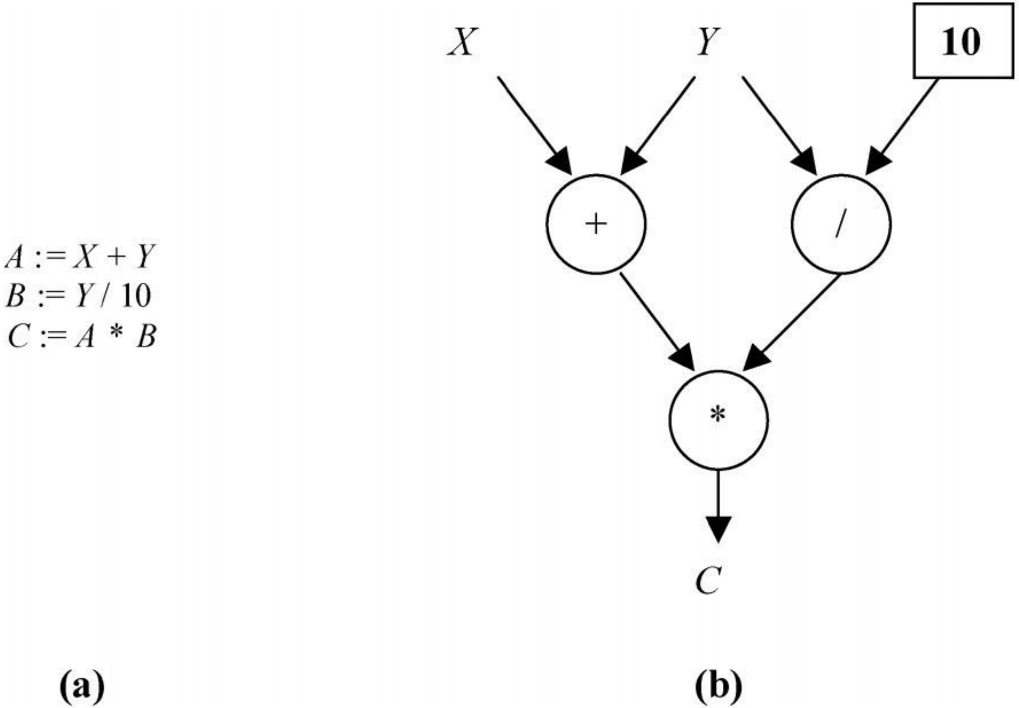
Outline

- History & Background
- How to set up Pd
- Pd Mechanics: patches, abstractions, objects, connections, data types, order of execution…
- Basics of Digital Audio
- Sound Synthesis Examples

Dataflow Languages

- Conceptually different to programming in Java, Python, Processing and similar languages
- Continuous flow of data in a graph instead of sequential execution of lines of code
- Visual dataflow environments are considered by many as a more gentle introduction to programming than writing source code
- Used by many as sketching tool, especially the environments where the graph can be changed while running

Dataflow Languages



$A := X + Y$
$B := Y / 10$
$C := A * B$

Johnston, W. M., Hanna, J. R., & Millar, R. J. (2004). Advances in dataflow programming languages. ACM Computing Surveys (CSUR), 36(1), 1-34.

Examples of Dataflow Languages

Primarily for audio

Pure Data (www.puredata.info)

Max/MSP (www.cycling74.com)

Kyma (http://kyma.symbolicsound.com/)

Primarily for graphics

Touch Designer (www.derivative.ca)

Quartz Composer (developer.apple.com)

vvvv (http://vvvv.org/)

History & Background

Open source visual programming language

Developed by Miller Puckette (IRCAM) in the 1990s, with the purpose of creating interactive computer music and multimedia

Miller Puckette also invented Max in the 1980s

The purpose with Pd was to extend Max data processing to applications other than just audio and MIDI, such as e.g. video and web

Setting up Pd (1) : Installing Pd

Pd can be downloaded from https://puredata.info/downloads

It runs on GNU/Linux, Mac OS X, iOS, Windows and Android

Two versions: **Pd Extended** (includes more libraries) & **Pd Vanilla** (only core functionality)

Pd Extended is no longer under development

IN THIS LAB WE WILL USE PD VANILLA

Setting up Pd (2) : Basic Configuration

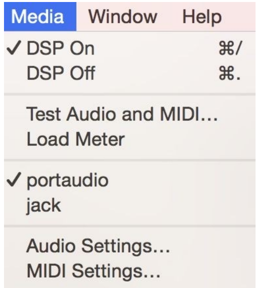
1. Make sure you have selected the correct audio driver

OSX : Media / standard (portaudio) or jack

Windows : Media / ASIO (via portaudio)

Linux : Media / OSS/ALSA/jack

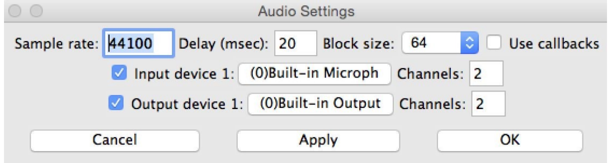
Pd can use a variety of audio drivers to connect to the sound card - make sure you have selected the right one!



2. Verify Audio Settings

Media / Audio Settings

make sure you have configured the correct input and output : choose the soundcard you wish to use with Pd and the number of channels you want to use (2 for normal stereo)


Audio Settings

Sample rate: 44100 Delay (msec): 20 Block size: 64 □ Use callbacks

☑ Input device 1: (0)Built-in Microph Channels: 2

☑ Output device 1: (0)Built-in Output Channels: 2

3.Test Audio and MIDI *Media / Test Audio and MIDI…*

Make sure to turn on DSP!

Pd Mechanics

Software called “patches” are developed graphically : visual metaphor from analogue synthesizer patches or electronic circuits

Algorithmic functions are represented by objects, which are placed on a screen, called canvas

Objects can be connected using cords: data flows from one object to another through these cords. Each object performs a specific task

Basics

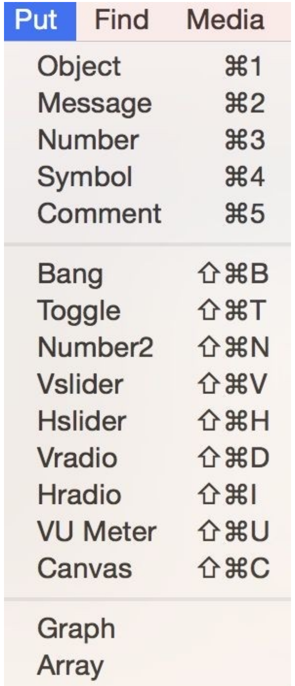
Pd Window / the terminal

Help Browser Window

Switching between edit - and play mode : ⌘E

Use the “Put” menu to place an object in your patch
Learn how to use shortcuts!

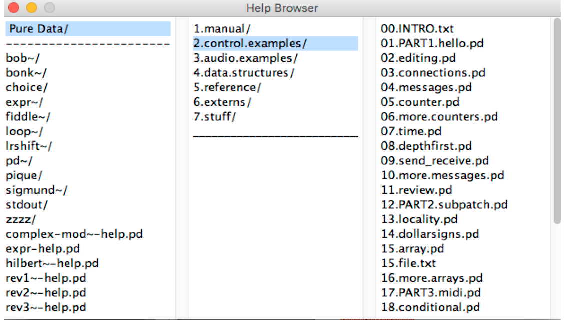
Right-click objects to get help (reference) for a specific object



Help- and Example Patches (Pd Vanilla)

Example patches from the *Help* menu:

- click “Help” and then “Browser”
- select ”Pure Data/” in the left column
- you will find many help patches here



```
Pure Data/
-----------------------
bob~/
bonk~/
choice/
expr~/
fiddle~/
loop~/
lrshift~/
pd~/
pique/
sigmund~/
stdout/
zzzz/
complex-mod~-help.pd
expr-help.pd
hilbert~-help.pd
rev1~-help.pd
rev2~-help.pd
rev3~-help.pd
```

```
1.manual/
2.control.examples/
3.audio.examples/
4.data.structures/
5.reference/
6.externs/
7.stuff/
______________________________
```

```
00.INTRO.txt
01.PART1.hello.pd
02.editing.pd
03.connections.pd
04.messages.pd
05.counter.pd
06.more.counters.pd
07.time.pd
08.depthfirst.pd
09.send_receive.pd
10.more.messages.pd
11.review.pd
12.PART2.subpatch.pd
13.locality.pd
14.dollarsigns.pd
15.array.pd
15.file.txt
16.more.arrays.pd
17.PART3.midi.pd
18.conditional.pd
```

Additional examples from the provided file archive:

- these files are referred to in *blue* in this pdf
- they are are located at “*Archive/introtoPdpatches/ExamplePatches*”

Patches, subpatches and abstractions

Pure Data files are called “patches”

Sometimes you want to use external patches or several instances of the same patch. In such cases, it is a good idea to encapsulate patches in object boxes, either as subpatches or as abstractions.

Subpatches :

```
[pd subpatch]
```

create an object called “pd subpatch” directly in the main patch

See the following patch in the archive folder (*Archive/introtoPdpatches/ExamplePatches*):

1_abstraction.pd

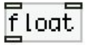
Abstractions :

```
[abstraction]
```

save a patch with a name such as "abstraction.pd" in the same folder as your main patch

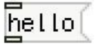
invoke it by writing "abstraction" in an object box in your main patch

Basic Elements



Objects

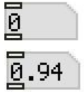
rectangular - object name and default value



Messages

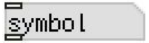
indentation on the right side

passes data which is stored inside of them when clicked



Numbers

If you hold the Shift key while using the mouse to change the number, you will scroll through decimal numbers



Symbols

Comments

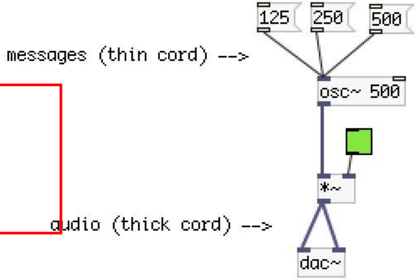
Objects and connections

Objects can be :

- built-in Pd-objects
- abstractions, i.e. reusable patches created in Pd itself
- externals, i.e. objects developed in another programming language

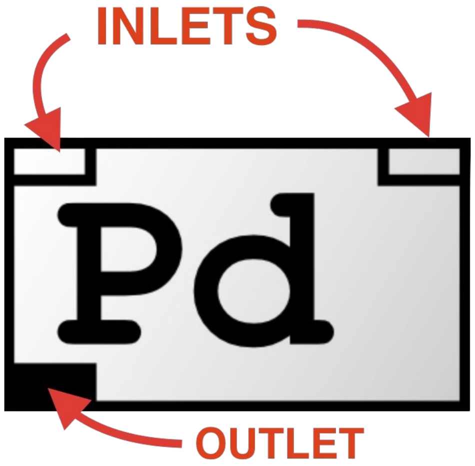
Cords connect objects to each other:

- signals (audio) have thick cords
- messages (control data) have thin cord



Inlets and outlets

objects have inlets and outlets



Useful Objects

`[bang]` does stuff!

`[tgl]` toggle 1 or 0

`[trigger]` sequence messages in right-to-left order

`[metro]` set bangs periodically

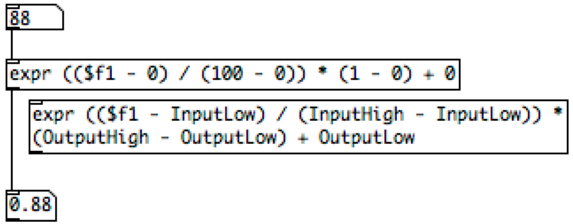
`[select]` bangs when received specific number

`[route]` route messages using selectors

`[pack]` packs lists

`[$1]` in message or object boxes, arguments starting with a dollar sign and a number are variables

`[maxlib/scale]`scaling input/output ranges (only available in Pd Extended, for Pd Vanilla use `[expr ]` to rescale, see below)



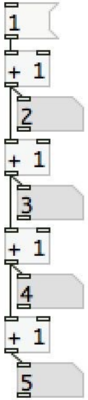
```
88
expr (($f1 - 0) / (100 - 0)) * (1 - 0) + 0
expr (($f1 - InputLow) / (InputHigh - InputLow)) *
(OutputHigh - OutputLow) + OutputLow
0.88
```

`[send]` `[receive]` use send and receive to avoid using too many patch cords

`[line]` ramps to target value over time

Order of Execution (1)

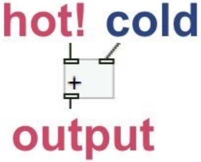
Patches (and objects) are executed from right to left, top to bottom!



Order of operations in Pd is determined by the following rules :

1. hot and cold inlets
2. connection order
3. depth first message passing

Order of Execution (2) - Hot and Cold



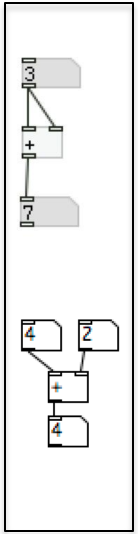
The leftmost inlet of any object is always a hot inlet

whatever the object receives on this inlet will trigger the object AND create output

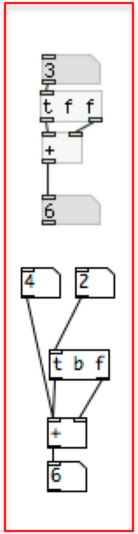
All other inlets are cold inlets

whatever the object receives here, it will store as a value, but not output the result

Problems can arise when a single outlet is connected to different inlets of a single object



bad



good!

Order of Execution (3) – Connection Order

2_connection_order.pd

Be careful

- when the order of operations is important
- when you have multiple outgoing connections from a single outlet

The order of events is determined by the order that the connections were made!

Solution: use trigger `[t  ]`

Order of Execution (4) - Depth First Message Passing

At a forking point (where you have a trigger object, or multiple connections from a single outlet), a single scheduled action is not finished until its whole underlying tree is done

The bottom-most message runs first

Everything below a spot in a chain is run before

Arithmetics

[+]
[-]

[*]

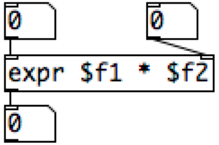
[/]

[pow]

[max]

[min]

[expr] allows you to write mathematical formulas



```
0        0
expr $f1 * $f2
0
```

remember that Pd differentiates between hot and cold inlets!

http://www.pd-tutorial.com/english/ch02s02.html#id410422

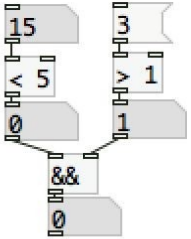
Comparing Numbers

Compare incoming data flow with static numbers and do something when condition is true (i.e. 1)

Logic operators & | << >> && || %

Relational operators > >= = <= <

Multiple comparisons



Decision making

using continuous data to trigger events

Example patch can be found under ”Help/Browser” in Pd Vanilla:

”Pure Data/2.control_examples/18.conditional.pd”

`[moses]` splitting numbers

`[select]` bang when specific number is received

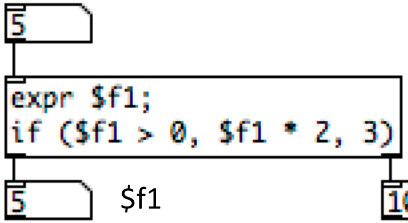
`[change]` bangs only when value changes

`[spigot]` passes messages to outlet if nonzero number is sent to the right inlet

`[route]` sort messages by type

If statements using `[expr]`

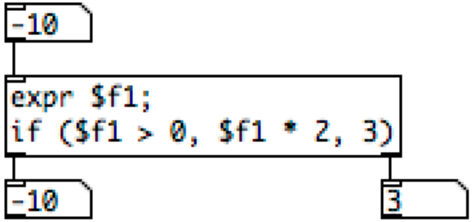
Syntax: if (statement, output if TRUE, output if FALSE)



```
5
expr $f1;
if ($f1 > 0, $f1 * 2, 3)
5                 10
```

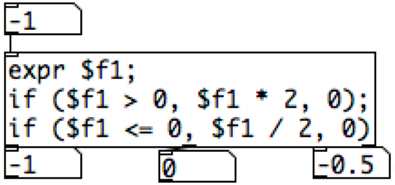
$f1

returned output



```
-10
expr $f1;
if ($f1 > 0, $f1 * 2, 3)
-10               3
```

Multiple conditions using “;” between if statements:



```
-1
expr $f1;
if ($f1 > 0, $f1 * 2, 0);
if ($f1 <= 0, $f1 / 2, 0)
-1       0        -0.5
```

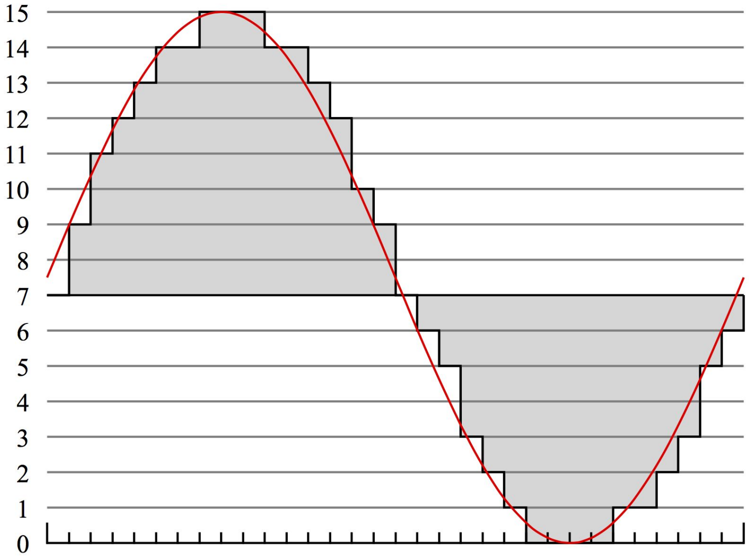
returned output from 2nd statement

$f1

returned output from 1st statement

Basics of Digital Audio

Digital representation of audio



Sample Rate

Sampling is the process of converting a signal (e.g. a function of continuous time) into a numeric sequence (a function of discrete time)

Objects like [osc~] generate a very fast sequence of numbers between -1 and 1 that is sent to the speaker by the [dac~] object

The loudspeaker makes 44100 tiny movements between -1 and 1 within one second (44100 is the sampling rate)

What sampling rate should one use when working with signals?

Nyquist's Sampling Theorem - avoid aliasing / foldover

use a sampling rate with double the max frequency of the signal you want to sample to avoid undesired audible effects

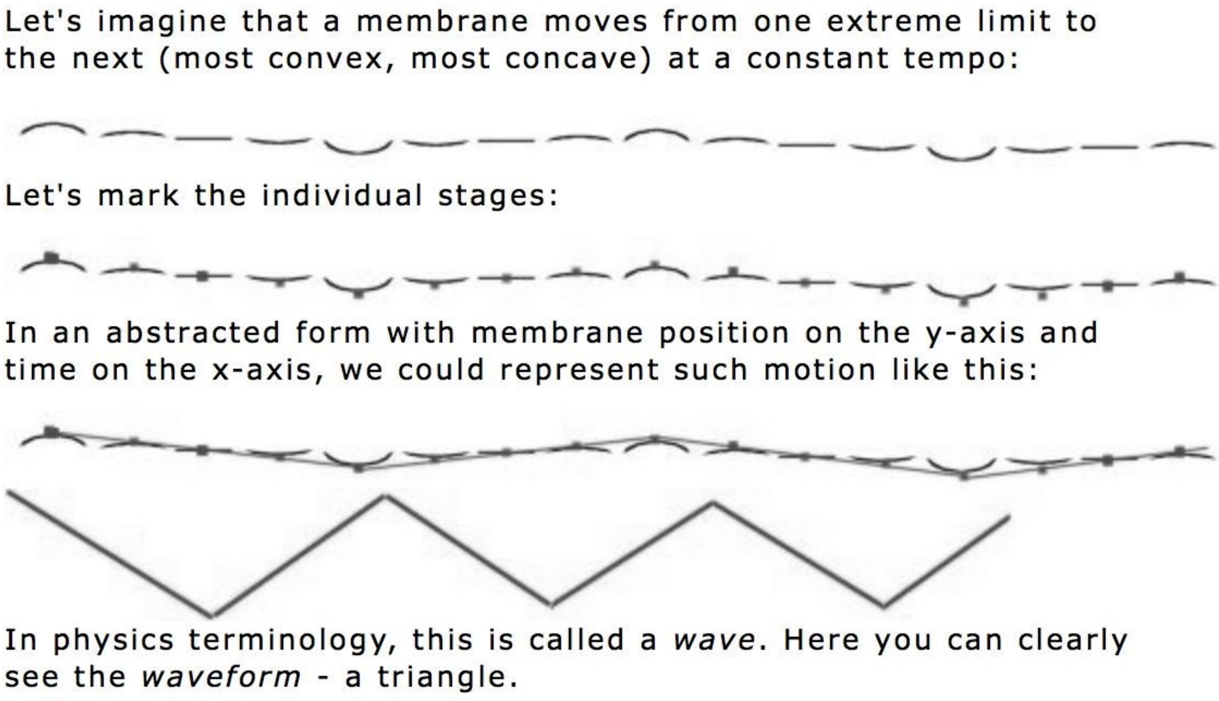
Bit Depth

One **bit** is a piece of information which is either 0 or 1

If we have 16 bits together to make one sample, then there are 2^{16}= 65,536 possible values that each sample could have

The more places to make one sample, the more detailed something can be processed. For Pd, which uses numbers to calculate frequencies, amplitude, etc., this means that the numbers can be processed more precisely, i.e., more decimal places can be used.

Waveforms



Let's imagine that a membrane moves from one extreme limit to the next (most convex, most concave) at a constant tempo:

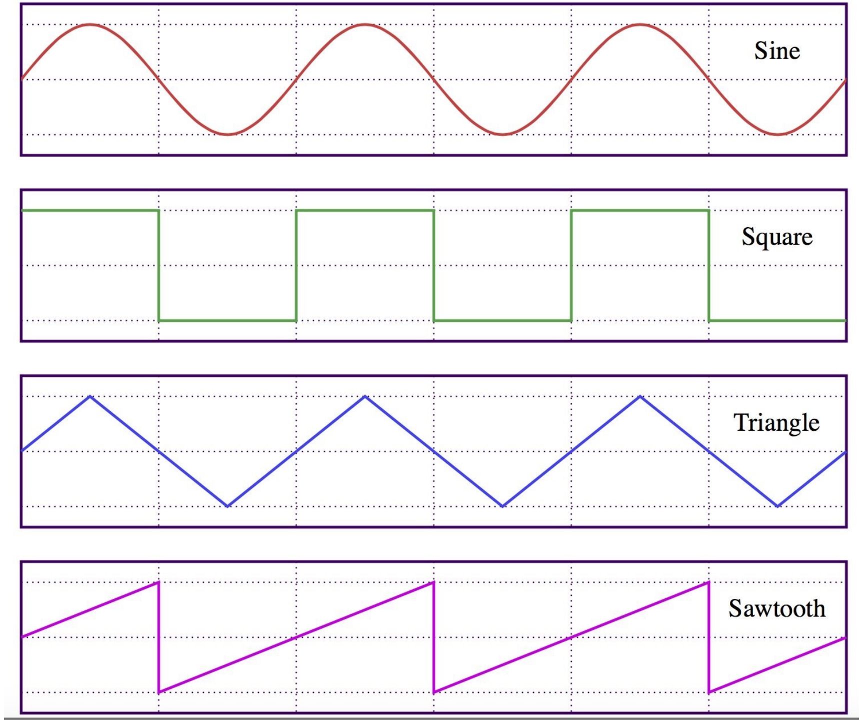
Let's mark the individual stages:

In an abstracted form with membrane position on the y-axis and time on the x-axis, we could represent such motion like this:

In physics terminology, this is called a *wave*. Here you can clearly see the *waveform* - a triangle.

http://www.pd-tutorial.com/english/ch03.html#chapt3.1.1.1.2

Waveforms



Digital Audio in Pd

Pd’s audio signals 32 bit floating point numbers (but hardware usually limited to 16 or 24 bits)

Pd assumes sampling rate 44100 Hz

Inputs between -1 and 1

Digital Audio in Pd

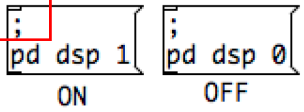
The "dac~" object turns numbers into sound by converting numbers into fluctuations of electrical current that - once amplified - cause the speaker membrane(s) to vibrate accordingly.

When we ask Pd to play a sound, it will read the samples and send them to the soundcard.

The soundcard then converts these numbers to an electrical current which causes the loudspeaker to vibrate the air in front of it, thus producing a sound.

tilde objects ~ & thicker patch cords are used for signals

Turn DSP on and off by sending “1” or “0” to the `[pd]` objec



```
;
pd dsp 1
```

ON

```
;
pd dsp 0
```

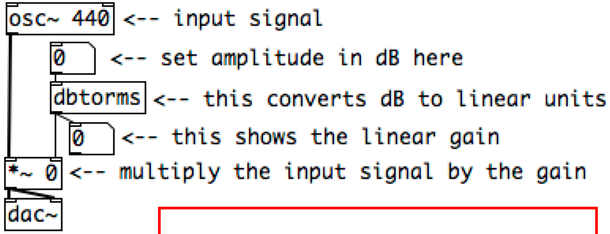
OFF

`[dac~ 1 2]` stereo audio output

`[adc~ 1 2]` stereo audio input

Amplitude Control

General approach for scaling the output level



<-- input signal

<-- set amplitude in dB here

<-- this converts dB to linear units

<-- this shows the linear gain

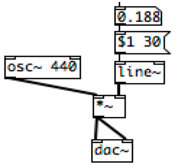
<-- multiply the input signal by the gain

When the speaker membrane has to span a large interval suddenly (e.g., when you turn on a sound) we can get undesired "click" effects

solution: use `[$1 ramptime(+[line~]`

for smooth transitions

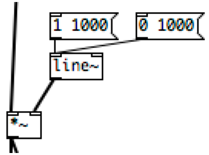
30 ms is preferable minimum time to ramp from zero to one



This is a good approach if you want to continuously change amplitude using an external input, for example

Example of turning audio ON/OFF using a smooth transition

incoming signal



1st argument: target value

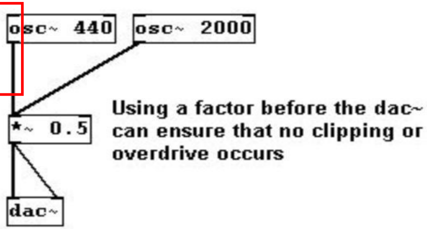
2nd argument: time (ms) for ramp

signal ramping from 0 to 1 in one second

It is important to note that amplitudes above 1 and below -1 will be 'clipped'

solution: normalize output

(make sure it is between -1 and 1)



Using a factor before the dac~ can ensure that no clipping or overdrive occurs

External Connections : MIDI

MIDI (Musical Instrument Digital Interface) is a technical standard that describes a protocol, digital interface and connectors

MIDI allows a wide variety of electronic musical instruments, computers and other devices to connect and communicate with each other

MIDI protocol itself doesn't contain any sounds, but comprises commands for controlling the patch or other instruments, e.g., "note-on", "velocity", and "note-off"

Every standard MIDI command (except for system-exclusive data) carries a channel number in addition to its command ID and command data. The channel number is 4 bits long, which means 2^4=16 channels can be controlled.

https://www.youtube.com/watch?v=10SdP_gviHY

MIDI in Pure Data

3_MIDI.pd

1. NOTE: External MIDI setup must be done before you launch Pure Data!
2. Download SimpleSynth (http://notahat.com/simplesynth/) and set MIDI source to “SimpleSynth virtual input”
3. Open Pd
4. Set MIDI output to “SimpleSynth virtual input”
5. Test MIDI.pd!

Audio Synthesis

Getting started with example patches reached from the ”Help/Browser” menu in Pd Vanilla:

- Making a sine wave : *Pure Data/3.audio examples/A01.sinewave.pd*
- Additive synthesis: *Pure Data/3.audio examples/D07.additive.pd* and *D13.additive.qlist.pd*
- FM modulation *Pure Data/3.audioexamples/A09.frequency.mod.pd*
- Subtractive synthesis: *PureData/3.audio.examples/J08.classicsynth*
- Sampler: *Pure Data/3.audioexamples/B07.sampler.pd*

Resources

- Pure Data Portal http://puredata.info/
- Download Pure Data http://puredata.info/downloads
- Flossmanuals http://write.flossmanuals.net/pure-data/introduction2/
- List of Pd Tutorials https://puredata.info/docs/tutorials
    - Video Tutorials with Dr. Rafael Hernandez: https://www.youtube.com/playlist?list=PL12DC9A161D8DC5DC
    - Programming electronic music in Pd : http://www.pd-tutorial.com/
    - Didactical Material by Alexandre Torres Porres : Computer Music with Examples in Pd https://sites.google.com/site/porres/ComputerMusic.zip
    - Didactical Material by Alexandre Torres Porres : Pd Tutorial https://sites.google. com/site/porres/Tut-Eng.zip