1 Exercise: Scaling numeric ranges

- Write a Pd patch that scales an input range 0…127 to an output range -1…+1

2 Audio signals in Pd (Farnell 2010b)

- Thick wires
  - Carry data as soon as DSP is turned on
  - Data flows at sample rate ([samplerate~] object)
- Objects have ~ sign appended (by convention)
- Processed in blocks of (by default 64) samples
- Check block content with [print~]; requires [bang]
- Get instantaneous amplitude with [snapshot~]
- Monitor integrated RMS level with [env~]

2.1 Soundcard inputs/outputs

- [adc~] ... mic input
- [dac~] ... loudspeaker output

2.2 Sending audio signals without wires

- [send~] or [s~], [receive~] or [r~]
- [throw~], [catch~]

2.3 Audio oscillators in Pd

- [osc~], [noise~] (-1...+1)
- [phasor~] (0...+1)
2.4 Wavetable oscillators
- [tabosc4~]
- Or use [phasor~] to feed [tabread~] or [tabread4~]
- [tabsend~]

2.5 Syntax of [vline~] and related objects
- [line]
- [line~]
- [vline~]: Syntax

2.6 Filters
- Design filters from scratch: [rpole~], [rzero~], [cpole~], [czero~], [biquad~]
- More user-friendly: [hip~], [lop~], [bp~], [vcf~]

2.7 Exercise: EDM break patch
- Sequencer with kick and hi-hat, gradually (and automatically) increasing LP cutoff

3 Subpatches (Farnell 2010a)
- Subpatches are really just containers to organize larger patches
- Subpatches have pd prepended to their name, e.g., [pd my_subpatch]
- Add [inlet], [outlet], [inlet~], [outlet~] objects to connect subpatch to parent

3.1 Exercise: Pythagoras subpatch
- Write a subpatch [pd magnitude] that calculates $c = \sqrt{a^2 + b^2}$
- 2 inlets for $a$ (hot) and $b$ (cold)
- What if I wanted to turn $b$ into a hot inlet?

4 Abstractions (ibid.)
- Are also containers
- But more importantly, they are a way of abstracting frequently used code into a new Pd object!
– Save code to new file, e.g., myabs.pd
– Reuse abstraction as [myabs] in another Pd patch (posibly multiple instances of it)
– Abstraction must be somewhere in Pd’s search-path (what always works: put abstraction in same directory as patch that uses it)

4.1 Creation arguments ($1, $2, etc.)
• Meaning of $N in an abstraction (as opposed to in a message)
• Unspecified creation arguments default to 0
• How to default an unspecified creation argument to a number other than 0? With [sel 0]!
• Tricky: We cannot distinguish an unspecified creation argument from one that’s been specified as 0!
• Providing lists of parameters and (un)packing them inside the abstraction with [pack] and [unpack]

4.2 Exercise: Numeric range scaler abstraction
• Write an abstraction [scale] that scales an input range 0...127 to an output range $1...$2.

4.3 $0 notation
• $0 is some random number that is guaranteed to be unique for each instance of an abstraction.
• Useful when using multiple instances of abstractions that use arrays etc.

5 Graph-on-parent
• A fun (and probably overused) feature.
• Idea: Provide GUI as part of an object.

6 How to write your own help patches
• Prepend help- or append -help (better, why?) to abstraction’s name
• E.g., myabs.pd’s help patch is myabs-help.pd
• Help patch needs to be in Pd’s -helppath
• Then you can right-click (Win, Linux) or ctrl-click (Mac) and [Help] on your own abstractions.
• Writing a help patch actually helps to clarify an abstraction’s specs for yourself (and remind yourself and others of them later).
6.1 Exercise: Help patch for [scale] abstraction

- Write a help patch that shows the functionality of your [scale] abstraction.

7 PD2 assignment

References and further reading

