

Einführung in Puredata 2 (Audioverarbeitung)

(zum Noise Watchers-Kurs Freie Musiksoftware 2008) 1w Februar 2008

Dieser Text beschreibt alle signalverarbeitenden Objekte der aktuellen Pd-Kernversion (0.41). Die meisten Kontrollobjekte sind im ersten Teil der Einführung beschrieben; der dritte Teil wird Datenstrukturen und die eingebauten Zeichenfunktionen behandeln.

0. Installation und Konfiguration

Pd ist ein freies Programmpaket zur graphischen Programmierung von Musikanwendungen; es kann von der **Website** seines Hauptautors, Miller Stuart Puckette, bezogen werden: crca.ucsd.edu/~msp. Dort findet man auch HTML- und pdf-Versionen seines Buches "Theory and Techniques of Electronic Music", einer hervorragenden Einführung mit Beispielprogrammen, natürlich für Pd. Etwa die gleichen Beispielpatches findet man als Tutorial in Pd unter Help->Browser->3.Audio.examples.

Hauptanlaufstelle für alle Fragen um Pd ist puredata.info mit Links zu Dokumentation, Downloads, Patchbibliotheken, Tutorials u.s.w. Hier stehen neben aktuellen und archivierten Quelltexten auch vorkompilierte Pakete für Debian, Ubuntu, Fedora, MacOSX 10.3 und 10.4, Windows 2000, XP und Vista sowie FreeBSD bereit. Für IRIX liegt nur eine ältere Version vor. Pd 0.41-1 läuft auch unter MacOSX 10.5.

Pd-Pakete finden sich in zwei Formen. Pd ("vanilla"), momentan in Version 0.41, enthält Kern und Dokumentation, dagegen sind in Pd-extended (aktuell 0.39-3) möglichst viele (zur Zeit 38) Zusatzbibliotheken von externen Entwicklern integriert. Täglich neu generierte Entwicklersnapshots von Pd-extended für Debian, Ubuntu und MacOSX 10.4 findet man auf autobuild.puredata.info.

Die **Zusatzbibliotheken** dienen unterschiedlichen Zwecken, sie bieten etwa:

- erhöhte Kompatibilität zu Max/MSP: *cyclone*, *zexy*,
- Filter höherer Ordnung (und vieles mehr): *iemlib*,
- listenverarbeitende Objekte: *list-abs*,
- 3D-Graphik: *GEM*
- Videoverarbeitung: *pdp*, *pidip*, *gridflow*
- Hallgerät: *freeverb*

Die **Konfiguration** von Pd erfolgt über die Menüs

- File->Startup.. : beim nächsten Start zu ladende Bibliotheken, Startflags
- File->Path.. : Suchpfade
- Media: Auswahl des Audio-/MIDI-Systems (etwa unter Linux: OSS, Alsa, jack)
- Media->Audio Settings: Geräteauswahl, Abtastrate, Kanalzahl
- Media->MIDI Settings: Geräteauswahl

Ab Pd 0.40 erlaubt ein [declare]-Objekt die Festlegung eines Suchpfades im Patch.

Die Konfiguration wird je nach Betriebssystem verschieden gespeichert (Unix: `~/pdsettings`, Windows: Registry, MacOS: plist in `~/Library/Preferences`); hier können zur Zeit noch Probleme auftreten (Zahl der Pfade und Bibliotheken,..).

Kommandozeilenparameter haben Vorrang, einen Überblick gibt "pd -help":

```

File Edit View Terminal Tabs Help
lw@charm:~$ pd -help
usage: pd [-flags] [file]...

audio configuration flags:
-r <n> -- specify sample rate
-audioindev ... -- audio in devices; e.g., "1,3" for first and third
-audiooutdev ... -- audio out devices (same)
-audiodev ... -- specify input and output together
-inchannels ... -- audio input channels (by device, like "2" or "16,8")
-outchannels ... -- number of audio out channels (same)
-channels ... -- specify both input and output channels
-audiobuf <n> -- specify size of audio buffer in msec
-blocksize <n> -- specify audio I/O block size in sample frames
-sleepgrain <n> -- specify number of milliseconds to sleep when idle
-nodac -- suppress audio output
-noadc -- suppress audio input
-noaudio -- suppress audio input and output (-nosound is synonym)
-listdev -- list audio and MIDI devices
-oss -- use OSS audio API
-32bit ---- allow 32 bit OSS audio (for RME Hammerfall)
-alsa -- use ALSA audio API
-alsaadd <name> -- add an ALSA device name to list
-jack -- use JACK audio API
      (default audio API for this platform: OSS)

MIDI configuration flags:
-midiindev ... -- midi in device list; e.g., "1,3" for first and third
-midioutdev ... -- midi out device list, same format
-mididev ... -- specify -midioutdev and -midiindev together
-nomidiin -- suppress MIDI input
-nomidiout -- suppress MIDI output
-nomidi -- suppress MIDI input and output
-alsamidi -- use ALSA midi API

other flags:
-path <path> -- add to file search path
-nostdpath -- don't search standard ("extra") directory
-stdpath -- search standard directory (true by default)
-helppath <path> -- add to help file search path
-open <file> -- open file(s) on startup
-lib <file> -- load object library(s)
-font-size <n> -- specify default font size in points
-font-face <name> -- specify default font (default: Bitstream Vera Sans Mono)
-font-weight <name> -- specify default font weight (normal or bold)
-verbose -- extra printout on startup and when searching for files
-version -- don't run Pd; just print out which version it is
-d <n> -- specify debug level
-noloadbang -- suppress all loadbangs
-stderr -- send printout to standard error instead of GUI
-nogui -- suppress starting the GUI
-guiport <n> -- connect to pre-existing GUI over port <n>
-guicmd "cmd..." -- start alternative GUI program (e.g., remote via ssh)
-send "msg..." -- send a message at startup, after patches are loaded
-rt or -realtime -- use real-time priority
-nrt -- don't use real-time priority
lw@charm:~$

```

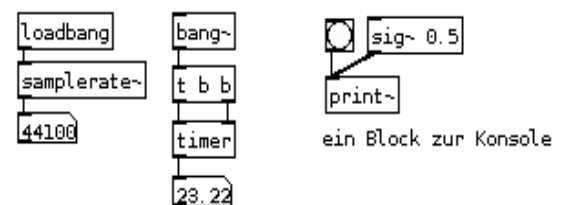
1. Grundlagen

Puredata und ähnliche Programmierumgebungen verarbeiten neben **Nachrichten** (messages der Typen float, symbol, bang...), die durch den Anwender (MIDI, Tastatur, ..) oder zeitgesteuert ([delay], [metro], ..) ausgelöst werden, auch **Audiodaten** (signals), die laufend berechnet werden. Die Namen der Objekte für die Signalbearbeitung enden auf ~, Signalverbindungen sind als dickere Linien hervorgehoben. Signalwerte sind in Pd wie alle Werte 32-bit-Fließkommazahlen; sollen diese über Audiohardware ausgegeben werden, müssen sie in den Bereich (-1,1) normiert werden, eventuell stehen dort auch nur geringere Wortbreiten zur Verfügung.

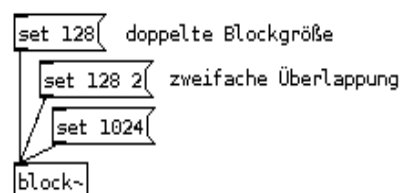
Pd berechnet immer einen **Block** (teils Vektor genannt) an Audiodaten gleichzeitig, wenn nicht anders eingestellt 64 Samples (Signal Vector Size). **[block~]** ändert für ein Fenster und dessen Unterfenster die Blockgröße, es ermöglicht wählbare Überlappung der Blöcke und Up-/Downsampling, **[switch~]** kann

(zusätzlich) die Audiotbearbeitung für das jeweilige Fenster ein-/ausschalten. Die Blockgrößen sollten Zweierpotenzen sein. Die Berechnung der Vektoren erfolgt also im Zeitabstand von Blockgröße/Abtastrate s, zu diesem Zeitpunkt sendet **[bang~]** eine bang-Nachricht (bei 48kHz Abtastrate und Defaultblockgröße also alle 1,33 ms). Die aktuelle Abtastrate liefert **[samplerate~]**. **[print~]** zeigt (mit bang ausgelöst) einen oder (Zahl am Eingang) mehrere Blöcke im Konsolenfenster an.

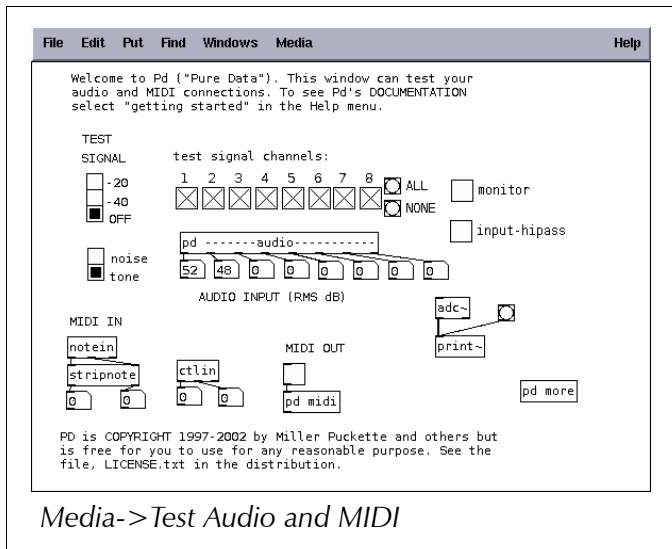
Die **Ein-/Ausgabe** von Audiodaten erfolgt über die Objekte **[adc~]** (Analog-to-Digital Converter) und **[dac~]** (Digital-to-Analog-Converter), die auf Funktionen des Betriebssystems zurückgreifen und mit unterschiedlichen Systemen (OSS, alsa, jack, coreaudio, ..) arbeiten können. Stehen mehrere Kanäle zur Verfügung, kann man mit Argumenten ([dac~ 2 4]) Quell- bzw. Zielkanäle anwählen. Default sind 2 Kanäle. **[adc~]/[dac~]** funktionieren nicht mit geänderten Blockgrößen. (Error: dac~: bad vector size). Man überprüfe die Funktion der Hardware über den Testpatch Media->Test Audio and MIDI.



Zeit zwischen zwei Blöcken



2el.pd



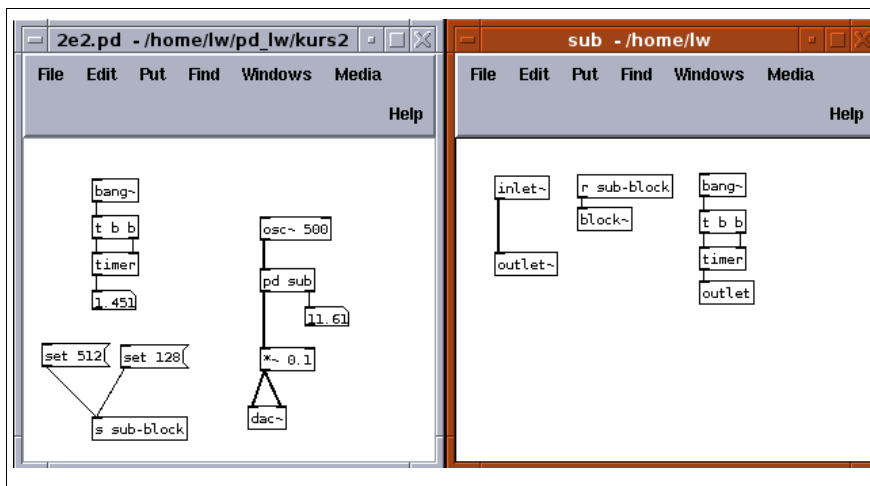
Die Audiotbearbeitung kann **ein-/ausgeschaltet** werden

- durch die Tastaturkürzel Ctrl-. bzw. Ctrl-/
- über die Menüpunkte Media -> Audio ON bzw. OFF
- über die Checkbox "compute audio" im pd-Fenster.
- durch eine Nachricht an das System [;pd dsp 1(bzw. [;pd dsp 0(

(Diese letzte Art ist ein Spezialfall einer Nachricht, die aus einem Patch (oder aus File->Message oder über pdsend von einem anderen Programm, sogar von einem entfernten Rechner) an das ausführende Pd-System gesendet wird.

Mittels dieser Nachrichten lassen sich sehr viel weiter gehende Programmieretechniken realisieren; sie ermöglichen auch die Erzeugung neuer Objekte und Verbindungen und somit eine Art von selbstmodifizierenden Programmen. Unvollständige Dokumentation: Help->Browser->Manuals->pd-msg in Pd-extended;)

In Subpatches und Abstractions dienen **[inlet~]** und **[outlet~]** als Ein- und Ausgänge. Signale können von einem (!) **[send~ name]** zu mehreren **[receive~ name]**-Objekten gesendet werden. Signale von mehreren **[throw~ name]**-Objekten werden an einem (!) **[catch~ name]** summiert empfangen. **[throw~]** und **[receive~]** können mit **[set name2(** umgeleitet werden. Diese nichtlokalen Verbindungen funktionieren nicht zuverlässig zwischen Fenstern mit verschiedenen Blockgrößen; Unterfenster mit veränderter Blockgröße sind daher mit **[inlet~]/[outlet~]** zu verbinden.



2e2.pd: Subpatch mit Signal-Ein- und -Ausgang, veränderlicher Blockgröße über send/receive

2. Signalquellen

[sig~] liefert ein konstantes Signal, dessen Wert durch eine Zahl am Eingang oder einen Argument gegeben wird. Es dient also dem Übergang vom Kontroll- in den Audiobereich, z.B. zur Steuerung des Signalnetzwerkes durch MIDI-Daten.

[phasor~] erzeugt ein Sägezahnsignal im Bereich 0 bis 1 (also nicht "gleichstromfrei" und nicht bandbreitenbegrenzt), es dient daher nicht primär als Audioquelle, sondern zum Durchlaufen von Wavetables mittels **[tabread4~]**. Ohne Argument erwartet das Objekt ein Eingangssignal, mit Argument einen Eingangswert zur Festlegung der Frequenz.

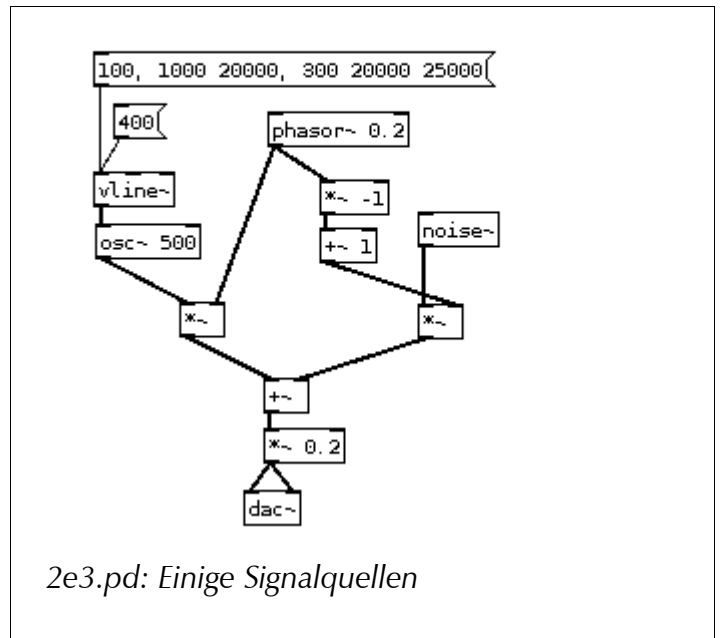
[osc~] liefert ein Cosinussignal zwischen -1 und 1. Ohne Argument erwartet das Objekt ein Eingangssignal, mit Argument einen Eingangswert zur Festlegung der Frequenz. Der rechte Eingang setzt die Phase (etwa 0,25: -sin).

[noise~] erzeugt weißes Rauschen im Wertebereich -1, 1. (In *iemlib* findet sich auch eine [pink~]-Quelle.)

[line~] erzeugt lineare Rampen. Es erwartet Listen von Wertepaaren (z, t) und liefert ein Signal, das vom momentanen Wert in t ms zum Wert z führt. Ein einzelner Eingangswert bewirkt einen sofortigen Sprung zu diesem Wert, [stop] friert den Wert ein.

[vline~] liefert eine genauere Variante (Interpolation zwischen den Samplezeiten), die sich mit einer Liste aus Parametertripeln (z, t, d) programmieren lässt. Dabei ist z das Ziel, t die Zeit zum Erreichen des Ziels und d die Verzögerung bis zum Auslösen des Segmentes. Die d-Werte müssen in aufsteigender Folge gegeben werden.

Weitere Signalquellen sind natürlich [adc~] sowie die weiter unten beschriebenen [tabread~],... und [readsf~].



3. Table und Array: Speicherobjekte für Zahlenfolgen

Arrays sind indizierte Zahlenlisten. Auch Signale sind Zahlenfolgen und können mit den Objekten [table] und [array] gespeichert werden.

[table]-Objekte werden normal mit Put->Object erzeugt; **Arrays** direkt über Put->Array. Beide werden mit Namen und Länge initialisiert, alle Werte sind mit 0 vorbelegt. Sie besitzen keine Ein- oder Ausgänge und werden über Nachrichten an ihren Namen angesprochen.

(Zur Erinnerung: Beginnt eine Messagebox mit einem Semikolon, wird das erste Wort nach dem Semikolon als Adresse interpretiert, an die der Rest der Zeile geschickt wird. Diese Adresse kann der Name eines [receive]-Objektes sein, der receive-name eines GUI-Objektes oder eben der Name eines Arrays.)

Zur graphischen Darstellung öffnet [table] ein Unterfenster (Kontextmenü -> Open), das Array zeigt seinen Graphen im Fenster eingebettet. Ein Kontextmenü erlaubt Veränderungen der dargestellten Größe und der Darstellungsart (Punkte, Polygone, Bézierkurve). Eine [;arrayname resize wert(-Nachricht verändert nachträglich die Länge, [;arrayname rename arrayname2(den Namen. Mit den Standard-Pd-Objekten ist die Länge des Arrays innerhalb des Patches nicht abzufragen, [;arrayname print(liefert diese und andere Informationen im Programmfenster. (Pd-extended bietet [arraysize] aus dem flatspace-Paket für Berechnungen, die die Länge benötigen.)

3.1. Eingabe

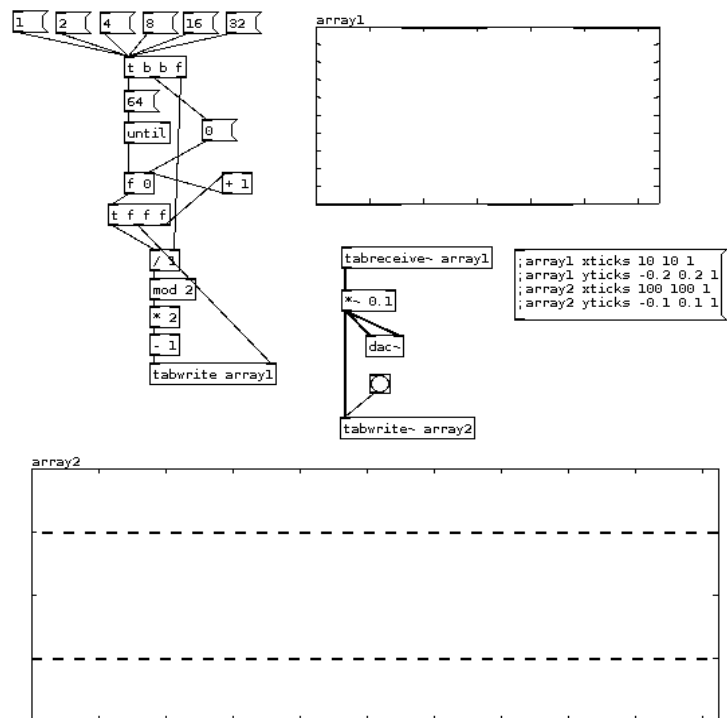
- Im Graphen kann mit der Maus gezeichnet werden, um Werte einzugeben und zu verändern.
- **[tabwrite arrayname]** schreibt den Wert am linken Eingang an die Stelle, die durch den rechten Eingang indiziert wird. **[tabread arrayname]** erwartet einen Index und liefert den zugehörigen Wert. Beide verstehen [set arrayname2(, um das angesprochene Array zu wechseln.
- Nachrichten der Form [;arrayname index wert1 wert2 wert3 ..(schreiben die Werte ab index.

- [`;arrayname write filename(` und [`;arrayname read filename(` schreiben und lesen den Arrayinhalt aus/in ein Textfile. [`savepanel`] und [`openpanel`] bieten Dateiauswahlmenüs.
- Bei der Erzeugung des Arrays sind alle Werte mit 0 vorbelegt; [`;arrayname const wert(` ändert alle Werte.

3.2. Graph

Der Graph des Arrays kann mit den Nachrichten [`;arrayname bounds low-x high-y high-x low-y(` verschoben werden. Skalen erzeugt man mittels [`;arrayname xticks start abstand unterteilung(`, desgleichen yticks, die Beschriftung der Skalen mit [`;arrayname xlabel y-wert x-wertliste(` ergibt die Platzierung der Werte aus x-wertliste in Höhe y-wert, dgl. **ylabel**.

2e4.pd: Array1 wird mit Rechtecken beschrieben und als Signal ausgelesen



3.3. Arrays als Signalspeicherobjekte

Die einfachsten Zugriffsobjekte sind [`tabsend~ arrayname`] und [`tabreceive~ arrayname`]; diese schreiben/lesen kontinuierlich einen Audioblock (d.h. 64 Werte, falls nicht über [`block~`] anders eingestellt) in das/aus dem Array.

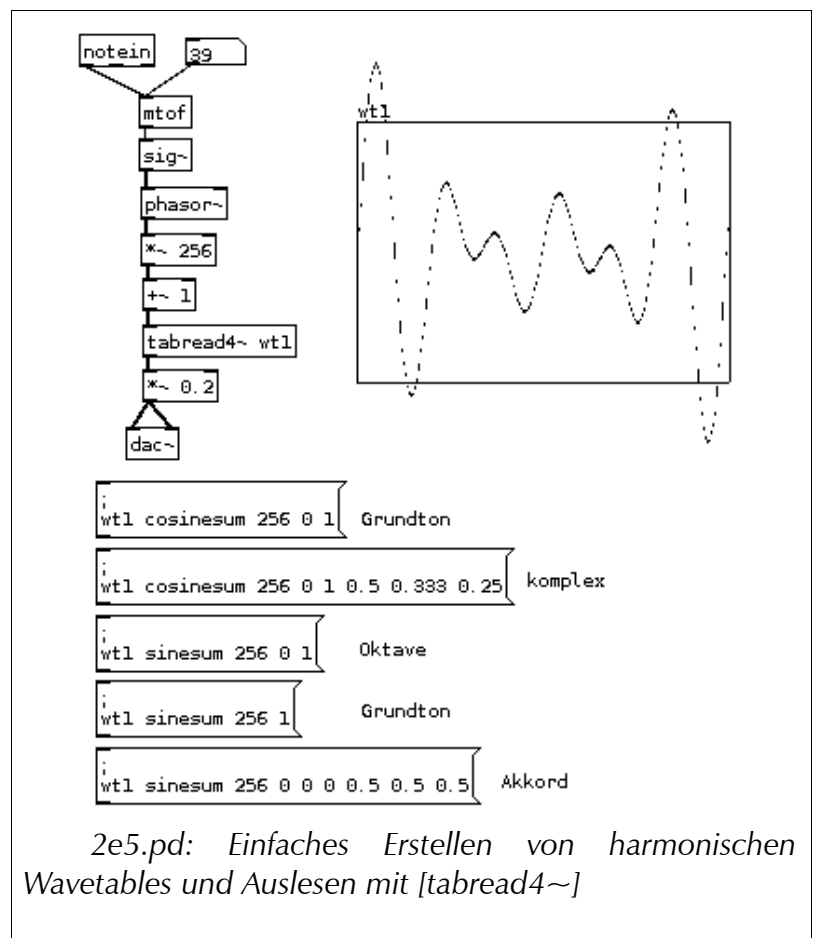
[`tabwrite~ arrayname`] schreibt ein Signal in ein Array. Die Aufnahme startet mit einer [`bang(`-Nachricht und endet mit [`stop(` bzw. am Arrayende.

[`tabplay~`] spielt den Arrayinhalt ab; teilweise Ausgabe erfolgt durch Angabe von Startindex und Weite, 0 oder bang spielt den ganzen Inhalt.

[`tabread~ arrayname`] liefert den indizierten Wert als Signal; zum Durchfahren des Arrays muss der Index also den Indexbereich durchlaufen. [`tabread4~ arrayname`] liefert durch Interpolation aus 4 benachbarten Werten auch Werte zwischen den Indizes. [`tabosc4~ arrayname`]

durchläuft das Array kontinuierlich; am Eingang des Objekts liegt die gewünschte Frequenz, mit der das Array durchlaufen wird. (cf 2e5a.pd)

[`;arrayname normalize(` normalisiert den Inhalt von `arrayname`.



2e5.pd: Einfaches Erstellen von harmonischen Wavetables und Auslesen mit [`tabread4~`]

3.4 Arrays für Wavetableoszillatoren

Für die Verwendung mit [tabread4~] oder [tabosc4~] lassen sich sehr einfach Arrays mit additiv definierten Wellenformen erstellen. [;arrayname **sinesum** punkte part1 part2 .. partn(berechnet in arrayname für punkte Punkte die Summe aus *n* harmonischen Sinussignalen, deren relative Intensitäten durch part1..partn angegeben werden. **cosinesum** liefert das Analoge mit Cosinuskomponenten.

4. Audiodateien

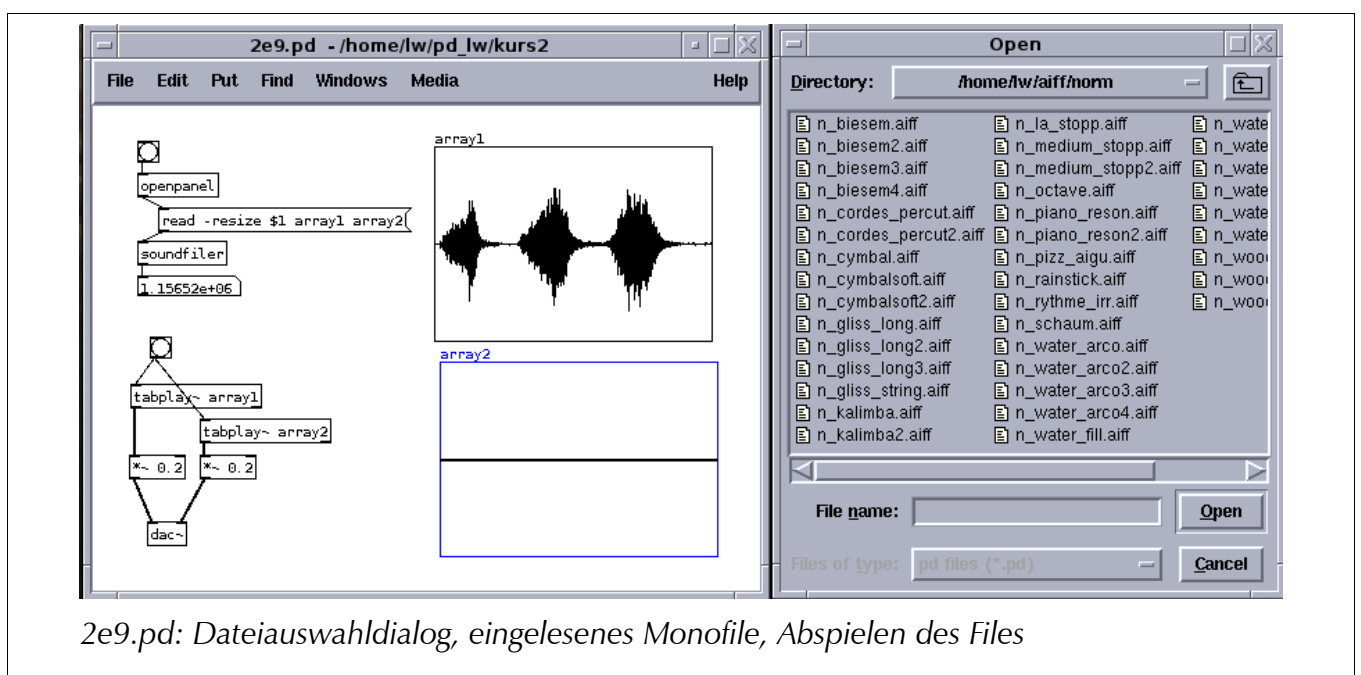
Zur Abspeicherung von Audiodaten sind etliche Dateiformate gebräuchlich. Diese unterscheiden sich etwa in Kanalzahl, Abtastrate, Wortbreite, Zahlendarstellung (16/24/32-bit, Festkomma, Fließkomma, big/little endian) u.a.m. Ein Dateikopf (*header*) enthält die Parameter, die das Programm zur korrekten Interpretation der Zahlen benötigt. Pd kann mit Dateien der Typen wav, AIFF und AU direkt umgehen, andere können eventuell mit Angabe der entsprechenden Parameter benutzt werden.

[**readsf~ channels bufsize**] liest aus einer Datei, die rechtzeitig (einige Sekunden) vorher mit einer [open filename(-Nachricht geöffnet wurde. *channels* legt die Kanalzahl und damit die Zahl der Ausgänge von [readsf~] fest, *bufsize* die Puffergröße pro Kanal. [1(oder [start(startet die Wiedergabe, [0(oder [stop(beendet sie. Der letzte Ausgang sendet [bang(am Dateiende.

[**writesf~ channels**] schreibt Audiodateien, *channels* legt die Kanalzahl und damit die Anzahl der Objekteingänge fest. Dateiname und Dateiformat (wav, aiff, next; 2, 3, 4-byte Samples) werden in einer [open filename(-Nhricht übermittelt. [start(, [stop(starten/beenden die Aufzeichnung.

[**soundfiler**] liest und schreibt Audiodateien in/aus Arrays; für jeden Audiokanal ist dabei ein getrenntes Array vorzusehen. Eine [read filename arrayname1 arrayname2(-Nachricht weist das [soundfiler]-objekt an, aus filename zwei Kanäle in 2 Arrays zu schreiben. Ein -resize-Parameter setzt die Arraygröße auf den erforderlichen Wert, -skip und -nframes geben Anfang und Größe des einzulesenden Bereichs an. Auch [write ..(-Nachrichten können neben dem Dateinamen Anfang und Größe des zu schreibenden Bereichs und die Headerinformationen übermitteln; -normalize weist [soundfiler] an, die Daten aus dem oder den Arrays zu normalisieren. Der Ausgang liefert die Zahl eingelesener Frames.

[**openpanel**] und [**savepanel**] bieten Dateiauswahldialoge und können benutzt werden, um die Nachrichten an [soundfiler],... zu generieren.



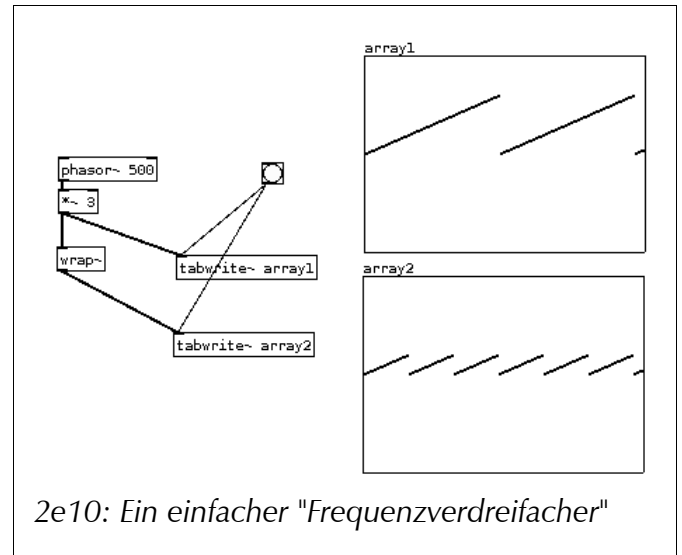
2e9.pd: Dateiauswahldialog, eingelesenes Monofile, Abspielen des Files

5. Mathematisches

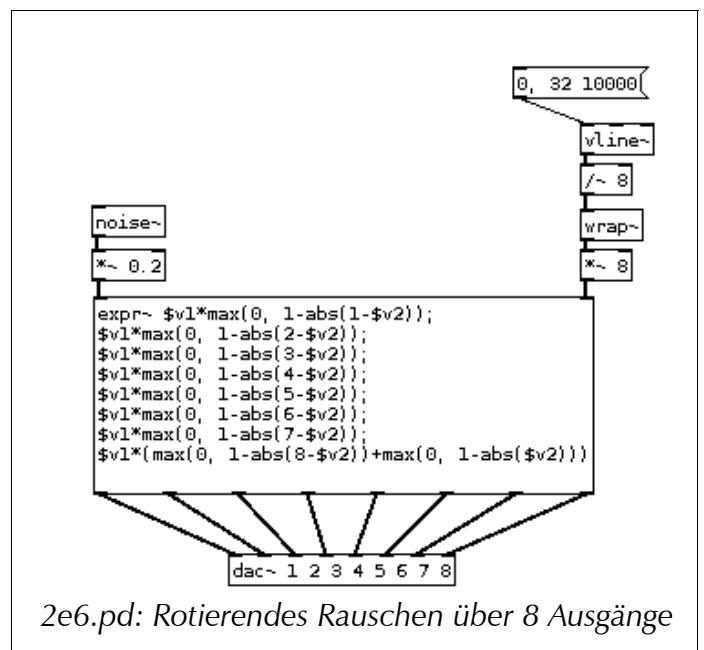
Alle Objekte verrechnen zwei Eingangssignale. Soll ein Signal mit einem Zahlenwert am rechten Eingang verknüpft werden, ist das Objekt mit einem Argument zu initialisieren.

Für Signale verfügbar sind:

- die Grundrechenarten: `[+~]`, `[-~]`, `[*~]`, `[/~]`,
- Maximum und Minimum: `[max~]`, `[min~]`,
- `[clip~ min max]` beschränkt das Signal auf Werte zwischen *min* und *max*,
- `[wrap~]` liefert als Ausgangssignal die Differenz zwischen Eingangssignalwert und der nächstkleineren ganzen Zahl d.h. bei positiven Zahlen $x - \text{int}(x)$, bei negativen $x - \text{int}(x) + 1$.
- spezielle Umrechnungen: (Frequenz \leftrightarrow MIDI-Notennummer) `[ftom~]`, `[mtof~]`, (Dezibel \leftrightarrow Amplitude) `[dbtorms~]`, `[rmstodb~]`, (Dezibel \leftrightarrow Leistung) `[powtodb~]`, `[dbtopow~]` verrechnen jedes Sample und sind daher rechenzeitintensiv, aber billiger als Nachbildungen im Eigenbau. Für die dB-Werte gilt als Referenz: 0 dB bei Amplitude 10^{-5} , 100 dB bei Amplitude 1. `[mtof~]` rechnet mit 440Hz für Note 69.
- `[sqrt~]` liefert die Quadratwurzel des Eingangssignals, `[rsqrt~]` den Kehrwert der Quadratwurzel. Beide Objekte verwenden einen schnellen, etwa auf 20 bit genauen Näherungsalgorithmus.
- `[cos~]` liefert den Cosinus von $2\pi \cdot \text{Eingangswert}$, d.h. eine Rampe von 0 bis 1 durchläuft einen Vollkreis. `[phasor~]---[cos~]` entspricht also `[osc~]`.
- Fast beliebige Berechnungen sind mit `[expr~]` und `[fexpr~]` möglich. Ersteres arbeitet auf Eingangssignalblöcken (als $\$v$ angesprochen), letzteres ermöglicht auch die Verrechnung der Samplewerte innerhalb eines Blocks ($\$x[n]$) und sogar die Rückführung der Ausgangswerte ($\$y[n]$), also das Schreiben eigener rekursiver Filter. Mehrere Ein- und Ausgänge sind möglich. Ausdrücke werden von `[expr]` und Verwandten eventuell genauer berechnet als durch eine Folge von arithmetischen Einzelobjekten. (Man lese hierzu die Originaldokumentation und die Hilfetexte.)
- Signalversionen der logischen und Vergleichsoperatoren bietet die zexy-Bibliothek.

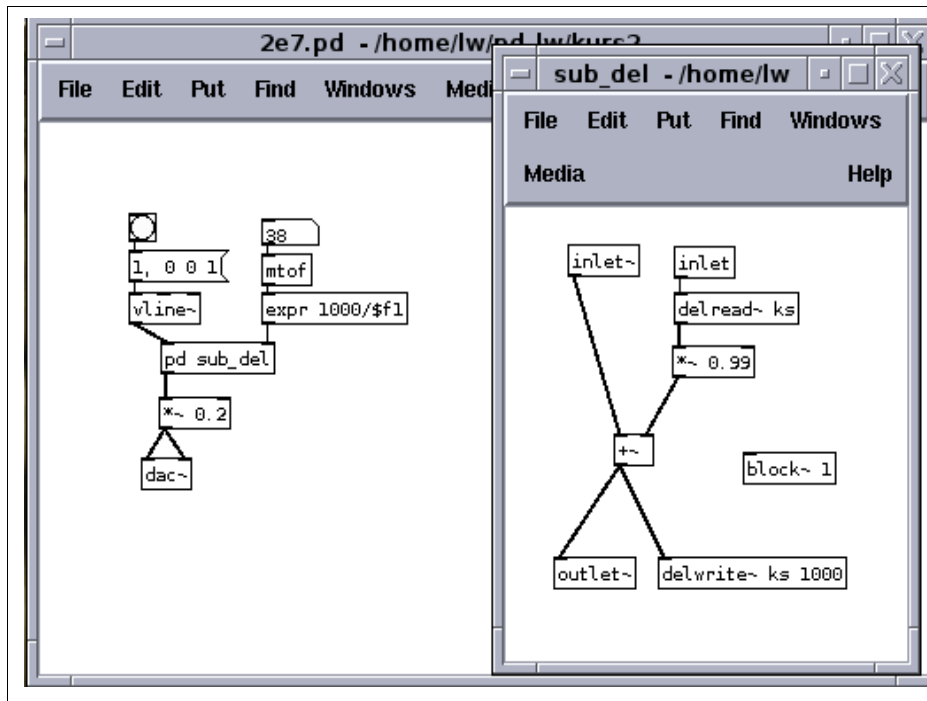


2e10: Ein einfacher "Frequenzverdreifacher"



2e6.pd: Rotierendes Rauschen über 8 Ausgänge

6. Verzögerungen

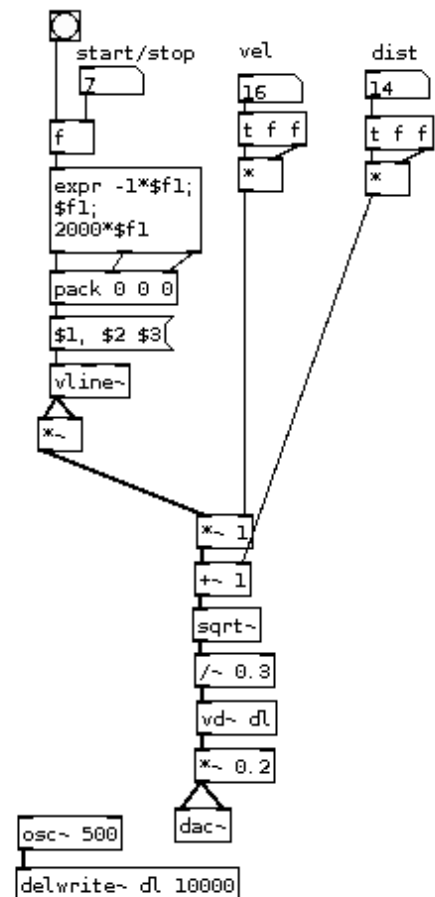


2e7.pd: Synthese eines Cembaloklages durch Modellierung des auf der "Saite" wandernden Einheitspulses (nach Puckettes Beispiel G.04). Die veränderte Blockgröße im Unterfenster erlaubt kurze Verzögerungen.

Verzögerungsleitungen werden mit **[delwrite~ name länge]** definiert und beschrieben, *länge* (in ms) bestimmt die Größe des reservierten Speichers und dadurch die maximale Verzögerungszeit. Mit **[delread~ name zeit]** wird aus der benannten Verzögerungsleitung gelesen; der Eingang erwartet einen Zahlenwert zur Veränderung der initialen Verzögerung. Mehrere [delread~] können mit unterschiedlichen Verzögerungen auf die selbe Verzögerungsleitung zugreifen. [delread~] interpoliert nicht; die Verzögerung ist immer ein Vielfaches der Abtastperiode.

[vd~ name] liest aus einer mit [delwrite~ name] beschriebenen Leitung, der Eingang erwartet die variable Verzögerungszeit in ms als Signal, die Werte werden interpoliert.

Für Verzögerungen unterhalb der Blockgröße muss sichergestellt werden, dass [delwrite~] vor [delread~] ausgeführt wird. Dazu kann man die lesenden Objekte in ein Unterfenster des Fensters mit den schreibenden Objekten setzen; die Unterfenster werden dann beim Öffnen des Programms auf jeden Fall nach den übergeordneten einsortiert. Wird das verzögerte Signal wieder in die Verzögerungsleitung eingespeist, ergibt sich die minimale Verzögerungszeit aus der Blockgröße; eine Auslagerung von [delread~] und [delwrite~] in ein Unterfenster mit kleinerer Blockgröße (sogar [block~ 1]) ist möglich.



2e8.pd: Dopplereffekt: Schallquelle mit 16 m/s im Abstand von 14 m vom Hörer von -7 sbis 7s zu hören.

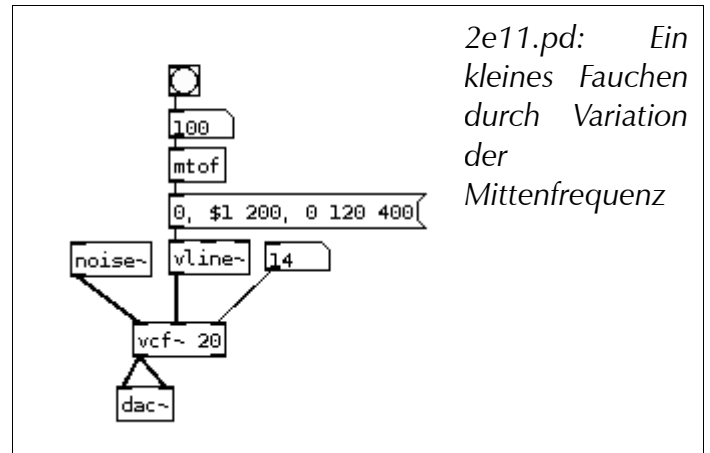
7. Filter

Der pd-Kern stellt zwei Arten von Filtern bereit: einfache, aber flexible rekursive Filter, mittels derer sich bei entsprechender Kenntnis der Theorie fast beliebige Filter aufbauen lassen und anwenderfreundliche Versionen für die häufigsten Zwecke.

Zur ersten Gruppe gehören die komplexwertigen Objekte `[cpole~]`, `[czero~]`, `[czero_rev~]` und die reellwertigen Analoga `[rpole~]`, `[rzero~]`, `[rzero_rev~]` sowie `[biquad~]`. Die Filterparameter werden als Argumente oder als Liste übergeben, sie haben keinen einfachen Zusammenhang mit dem sich ergebenden Frequenzgang.

Zur zweiten Gruppe gehören:

- **[lop~ freq]**, Tiefpassfilter: Verstärkung 1 bei Frequenz 0, -3dB bei *freq*, Abfall 6dB/Oktave,
- **[hip~ freq]**, Hochpassfilter. Auch zum Entfernen von "Gleichspannung" (etwa $\text{freq}=3$),
- **[bp~ freq Q]**, Bandpassfilter mit Mittenfrequenz *freq* und Güte *Q*,
- **[vcf~]**, Bandpassfilter mit Signaleingang zur schnellen, kontinuierlichen Änderung der Mittenfrequenz, die Filtergüte wird als Zahl gegeben.

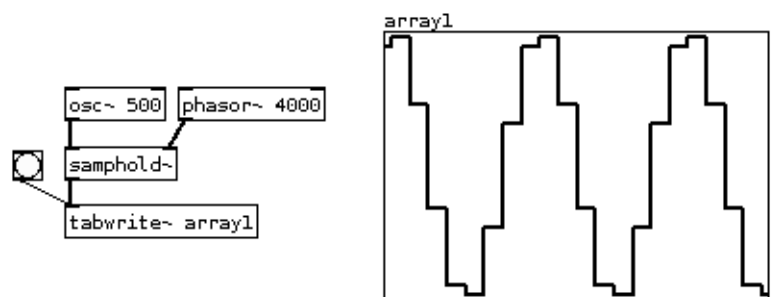


2e11.pd: Ein
kleines Fauchen
durch Variation
der
Mittenfrequenz

Leicht verwendbare höhere Filter finden sich in den Erweiterungsbibliotheken, v.a. in iemlib.

8. Analyse

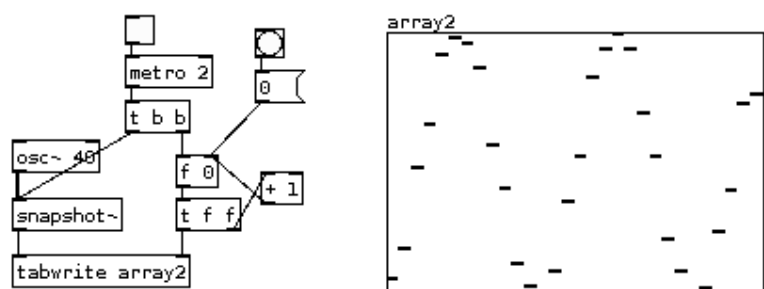
[samphold~] vergleicht zwei Eingangssignale. Das Eingangssignal (links) wird nur dann weitergeleitet, wenn das Kontrollsignal (rechts) abnimmt; bei ansteigendem Kontrollsignal wird der letzte Wert des Ausgangssignals gehalten. Verwendet man als Kontrollsignal einen [phasor~]-Ausgang, wird das Eingangssignal nur an der fallenden Flanke des Kontrollsignals abgetastet und gehalten.



2e12: Bei der fallenden Flanke des `phasor~` entnimmt `samphold~` den Wert des einkommenden Signals und hält diesen.

[snapshot~] liest aus einem Eingangssignal einen Zahlenwert aus, wenn es einen bang erhält.

[threshold~ t t_r r_t] vergleicht ein Eingangssignal mit als Argument oder als Liste angegebenen Werten. Steigt der Eingang über den Wert t , sendet der linke Ausgang einen bang. Fällt das Signal unter r , sendet der rechte Ausgang einen bang. Nach

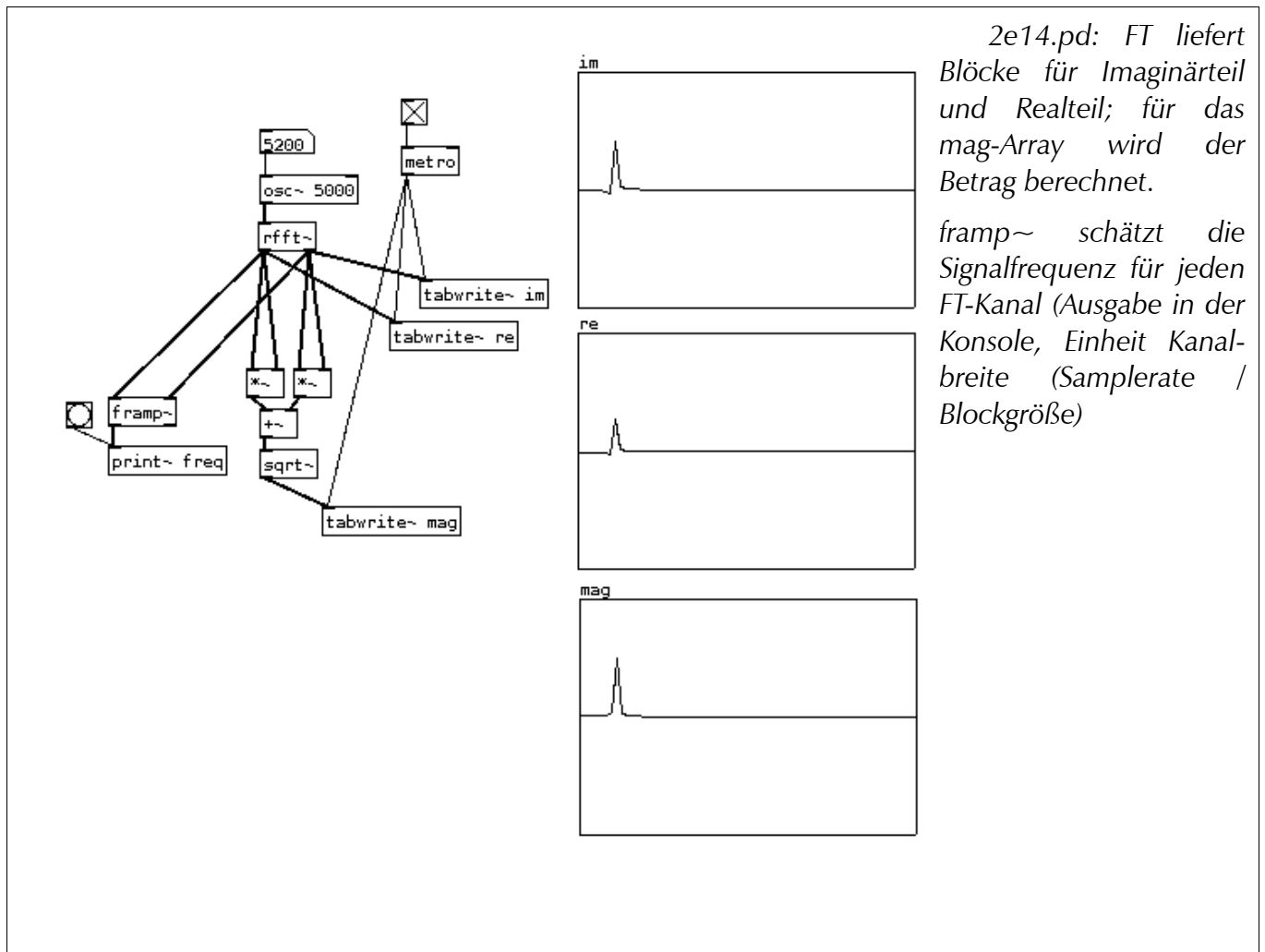


2e13.pd: Sampling alle 2 ms, diesmal nicht im Signalbereich.

einem bang ist das Ausgang für die Zeit t_t bzw. r_t gesperrt.

[env~ blocksize] analysiert einen Block der Größe *blocksize* (Zweierpotenz) und gibt dessen RMS-Amplitude aus. (100dB für Amplitude 1). Die analysierten Blöcke werden mit Faktor 2 überlappt, [env~] meldet also zweimal pro Block einen Wert.

Fourier-Analysen und -Resynthesen werden von **[fft~]** (Eingang komplex), **[ifft~]** (inverse Transformation, Ausgang komplex), **[rfft~]** (Eingang reell), **[rifft~]** (inverse Transformation, Ausgang reell) durchgeführt. Die FT verarbeitet je einen Block, je nach gewünschter Zeit-/Frequenzauflösung muss die Blockgröße mittels **[block~]** verändert werden.



[framp~] analysiert Fourier-transformierte Blöcke und liefert Blöcke, die für jeden FFT-Slot geschätzte Frequenz und Amplitude des ihn speisenden Signals angeben.

Im *Extra-Verzeichnis* von Pd befinden sich einige größere Objekte mit Verweisen auf die zugehörige Theorie:

[bonk~] sucht nach Attacken, d.h. plötzlich Änderungen des Spektrums vorzugsweise bei kleinen Schlagzeuginstrumenten,

[fiddle~] liefert Schätzungen von Tonhöhe und Amplitude aus komplexen Tönen,

[pique] liefert eine Peakliste aus FT-transformierten Blöcken.