institut für elektronische musik und akustik

IEM Report 12/03:
IEMLIB für PD

Funktionsbeschreibung der Pd-Objekte von iemlib1, iemlib2 und iemabs
(Zusatz-Bibliotheken für die Echtzeit-Multimedia-Plattform Pd)
Release 1.15

Verfasser:

Thomas Musil

Version 1.0, Stand 15.09.2003

IEM - INSTITUT FÜR ELEKTRONISCHE MUSIK UND AKUSTIK

Vorstand: O.Univ.-Prof. Mag. DI Dr. Robert HÖLDRICH

A-8010 Graz, Inffeldgasse 10/3, Tel.:+43/(0)316/389 – 3447, FAX:+43/(0)316/389 – 3171

musil@iem.at http://iem.at/

Zusammenfassung

Die iemlib, bestehend aus 4 Pd-External-Bibliotheken und einem Sammelordner für Pd-Subpatches, umfasst ca. 180 Objekte. Diese erweitern die Anwendbarkeit von Pd, einer graphischen Echtzeit-Multimedia-Programmiersprache, entwickelt von Miller Puckette. Hauptmerkmale der iemlib sind digitale Filter (FIR, IIR 1. bis 10. Ordnung), Parameter-Management und Parameter-Speicherung von Pd-Messages.

Abstract

Iemlib contains 4 pd-external-libraries and one directory for pd-subpatches, or appr. 180 Objects. It expands the usability of Pd, a realtime multimedia graphical programing language, developed by Miller Puckette. The main iussues of iemlib are digital filters (FIR, IIR 1. to 10. order) and parameter management of Pd messages.

INHALTSVERZEICHNIS

IEMLIB für PD

Funktionsbeschreibung der Pd-Objekte von iemlib1, iemlib2 und iemabs

Release 1.15

1 Einleitung

Die iemlib versteht sich einerseits als Ergänzung zu den internen Pd-Objekten (Internals) von Miller Puckette [1], andererseits als Ansammlung von Objekten, die für diverse Projekte speziell programmiert wurden, und doch allgemeinen Verwendungscharakter aufweisen. Die iemlib setzt sich zusammen aus 4 dynamisch linkbaren Bibliotheken (iemlib1 , iemlib2, iem_t3_lib , iem_mp3) mit der Endung *.dll für Windows, *.pd_linux für Linux und *.pd_darwin für Macintosh. Jede Bibliothek wiederum besteht aus mehreren, in C programmierten Objekten (Externals). Weiters gibt es einen Ordner „iemabs“ mit Pd-Makro-Objekten (Abstractions), die aus einem Verbund diverser Externals und Internals zusammen gebaut wurden. Dabei bilden iemlib1, iemlib2 und iemabs eine untrennbare Einheit; während iem_t3_lib und iem_mp3 Spezial-Bibliotheken darstellen. Schließlich gibt es noch „iemhelp“ , einen Ordner mit einer Sammlung von Help-Patches für alle Externals und Abstraktionen von iemlib.

Die aktuelle Version von iemlib kann man kostenlos beziehen unter http://pd.iem.at/iemlib http://iem.at/~musil/iemlib/.

2 Begriffsdefinitionen

Grund-Elemente (Einträge) einer PD-Message können sein:

- <*float*> 32-Bit-Gleitkommazahl bzw. 24-Bit Ganzzahl;
- <*symbol*> zusammengesetzte Struktur, bestehend aus einem null-terminierten 8-Bit-Buchstabenfeld, und einem Zeiger-Eintrag (welcher zB. auf einen reservierten Speicherbereich verweist).

Eine PD-Message besteht aus einem Selektor-Symbol {*xxx*} und einem anschließenden Feld von <*float*>- oder <*symbol*>-Einträgen. In Pd gibt es folgende bekannte Message-Typen:

- {*bang*} inhaltsloses Ereignis (das Feld enthält keine Einträge);
- {*float*} das Feld entält nur einen <*float*>-Eintrag (wird im allgemeinen ohne dem Selektor {*float*} dargestellt);
- {*symbol*} das Feld entält nur einen <*symbol*>-Eintrag;
- {*list*} das Feld enthält eine Reihe von <*float*>- und/oder <*symbol*>-Einträgen; (Listen, die mit einem <*float*>-Eintrag beginnen, werden im allgemeinen ohne dem Selektor {*list*} dargestellt).

Zusätzlich gibt es noch spezielle, vom Programmierer definierte Messages, zB:

- {*open*} <*symbol*> Dateiname;
- {*clear*} ohne Inhalt;
- {*set*} <*float*> Parameter 1 + <*float*> Parameter 2.

Diese werden im allgemeinen {*anything*}-Messages genannt. Hier kommt es zu einer gewissen Unschärfe in der Bezeichnung, da man auch anything für jegliche Message verwenden kann (also auch für die bekannten Grundtypen).

3 Signalverarbeitende Objekte

3.1 Filterobjekte

FIR~ : Filter mit endlicher Impuls Antwort

Funktionsweise:

FIR~ faltet die eingehende Sample-Sequenz mit den aus einem graphischen Array stammenden Koeffizienten. Wird das Array oder die Ordnung verändert, tritt eine Unstetigkeit im Ausgangssignal auf. Falls die Filter-Ordnung größer als die Array-Länge ist, wird nicht gefaltet (Null-Signal am Ausgang).

Übergabe-Parameter:

1.Argument: <*symbol*> Array-Name;

2.Argument: <*float*> Filter-Ordnung (<= Array-Länge).

Eingänge und Ausgänge:

1.Eingang: {*signal*} : Filter-Eingang;

oder: {*set*} + <*symbol*> Array-Name + <*float*> Filter-Ordnung;

1. Ausgang: {*signal*} : gefalteter Filter-Ausgang.

---

maverage~ : gleitendes Mittelwert-Filter

Funktionsweise:

Das Moving-Average-Filter wurde realisiert mit einem IIR-Filter und einem Delay; es faltet die eingehende Samplesequenz mit einem Rechteckfenster der Breite N und der Höhe 1/N. Ändert sich die Fensterbreite, wird das Ausgangssignal Breitendauer lang unterdrückt.

Übergabe-Parameter:

1.Argument: <*float*> maximale Fenster-Breite in ms;

2.Argument: <*float*> aktuelle Fenster-Breite (<= Maximale Fenster-Breite) (>= 1.45 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} aktuelle Fenster-Breite in ms.

1.Ausgang: {*signal*} arithmetisch gemittelter Filter-Ausgang.

Für die folgenden Filter gilt:

Die Berechnung der Koeffizienten wurde aus dem Analog-Bereich mittels Bilinear-Transformation in den z-Bereich bewerkstelligt.

Legende: f ... Frequenzvariable in Hz

f_0 ... Grenz- , oder Resonanz- , oder 90-Grad, oder 180-Grad-Frequenz in Hz

$P = jf / f_0$ ... komplexe normierte Frequenzvariable

SR ... Samplerate in Hz

z ... freiwählbare Variable im z-Bereich

$L = \cot(\pi * f_0/SR)$

bw ... Bandbreite eines Filters in Hz

$BW = bw / f_0$ ... normierte Bandbreite eines Filters

$Q = f_0 / bw$ ... Güte eines Filters

Bilinear-Transformation:

$$P = L*(1 - z^{-1})/(1 + z^{-1}) \qquad [2]$$

Die Übertragungs-Funktion $H(z^{-1})$ wird auf eine rekursive Differenzengleichung der Form:

$wn_0 = in[i] + b_1*wn_1 + b_2*wn_2$

$out[i] = a_0*wn_0 + a_1*wn_1 + a2*wn_2$

$wn_2 = wn_1$

$wn_1 = wn_0$ gebracht.

---

ap1~ : Allpass 1.Ordnung

hp1~ : Hochpass 1.Ordnung

lp1~ : Tiefpass 1.Ordnung

Funktionsweise:

Die Filter haben folgende Übertragungsfunktionen:

- Allpass ap1~ H(P) = (1 - P) / (1 + P);
- Hochpass hp1~ H(P) = P / (1 + P);
- Tiefpass lp1~ H(P) = 1 / (1 + P).

Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Zusätzlich wird bei jeder Neuberechnung der Koeffizienten die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz- (-3 dB) bzw. -90 Grad Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz- (-3 dB) bzw. -90 Grad Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

lp1_t~ : Tiefpass 1.Ordnung mit Zeitkonstanten-Eingang

Funktionsweise:

Wie lp1~ , nur statt Frequenz-Eingang ein Zeitkonstanten-Eingang (in welcher Zeit der Ausgang 63 % des Eingang-Sprungs erreicht). Bei Änderung der Zeitkonstanten wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Zeitkonstante auf die neue Zeitkonstante logarithmisch interpoliert. Zusätzlich wird bei jeder Neuberechnung der Koeffizienten die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Zeitkonstante in ms;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Zeitkonstante in ms;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

ap2~ : Allpass 2.Ordnung

hp2~ : Hochpass 2.Ordnung

lp2~ : Tiefpass 2.Ordnung

bpq2~ : Bandpass 2.Ordnung mit Güte-Eingang

bpw2~ : Bandpass 2.Ordnung mit Bandbreite-Eingang

bsq2~ : Bandsperre 2.Ordnung (notch) mit Güte-Eingang

bsw2~ : Bandsperre 2.Ordnung (notch) mit Bandbreite-Eingang

rbpq2~ : Resonanz-Bandpass 2.Ordnung mit Güte-Eingang

rbpw2~ : Resonanz-Bandpass 2.Ordnung mit Bandbreite-Eingang

Funktionsweise:

Die Filter haben folgende Übertragungsfunktionen:

- Allpass ap2~ $H(P) = (1 – P/Q + P^2) / (1 + P/Q + P^2)$;
- Hochpass hp2~ $H(P) = P^2 / (1 + P/Q + P^2)$;
- Tiefpass lp2~ $H(P) = 1 / (1 + P/Q + P^2)$;
- Bandpass mit Güte bpq2~ $H(P) = (P/Q) / (1 + P/Q + P^2)$;
- Bandpass mit Bandbreite bpw2~ $H(P) = (P*BW) / (1 + P*BW + P^2)$;
- Bandsperre mit Güte bsq2~ $H(P) = (1 + P^2) / (1 + P/Q + P^2)$;
- Bandsperre mit Bandbreite bsw2~ $H(P) = (1 + P^2) / (1 + P*BW + P^2)$;
- Resonanz-Bandpass mit Güte bpq2~ $H(P) = P / (1 + P/Q + P^2)$;
- Resonanz-Bandpass mit Bandbreite bpw2~ $H(P) = P / (1 + P*BW + P^2)$.

Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Gleiches gilt für den Parameter Güte bzw. Bandbreite. Für hp2~ und lp2~ gilt: Güte-Werte kleiner 0.5 ergeben eine nichtschwingende Sprungantwort, Güte von 0.707 ergibt eine Butterworth-Charakteristik im Betrags-Frequenzgang, Güte-Werte größer 0.707 ergeben eine Resonanz-Überhöhung im Betrags-Frequenzgang. Die Allpass-Bandbreite definiert sich als Frequenz-Bereich zwischen einer Ausgangs-Phasen-Verschiebung von -90 bis -270 Grad. Die beiden Bandpässe bpq2~ und bpw2~ besitzen bei ihrer Resonanz-Frequenz eine Verstärkung von 1, die Resonanz-Bandpässe rbpq2~ und rbpw2~ besitzen bei ihrer Resonanz-Frequenz eine Verstärkung von Q. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz- (-3 dB), Resonaz- bzw. -180 Grad Frequenz in Hz;

2.Argument: <*float*> Güte (ohne Einheit) bzw. Bandbreite in Hz;

3.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz- (-3 dB), Resonaz- bzw. -180 Grad Frequenz in Hz;

3.Eingang: {*float*} Güte (ohne Einheit) bzw. Bandbreite in Hz.

4.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

hml_shelf~ : Höhen-Mitten-Bässe Klangregler

Funktionsweise:

Bei Änderungen der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Gleiches gilt für den Parameter Gain, wobei hier zuerst die dB-Werte linear interpoliert werden und dann erst die Effektiv-Wert-Konvertierung statt findet. Weiters gilt: Übergangs-Frequenz Mitten zu Höhen muss größer sein als Übergangs-Frequenz Bässe zu Mitten. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Gain im Bassbereich in dB;
2.Argument: <*float*> Übergangs-Frequenz Bass zu Mitten in Hz;
3.Argument: <*float*> Gain im Mittenbereich in dB;
4.Argument: <*float*> Übergangs-Frequenz Mitten zu Höhen in Hz;
5.Argument: <*float*> Gain im Höhenbereich in dB;
6.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;
2.Eingang: {*float*} Gain im Bassbereich in dB;
3.Eingang: {*float*} Übergangs-Frequenz Bass zu Mitten in Hz;
4.Eingang: {*float*} Gain im Mittenbereich in dB;
5.Eingang: {*float*} Übergangs-Frequenz Mitten zu Höhen in Hz;
6.Eingang: {*float*} Gain im Höhenbereich in dB;
7.Eingang: {*float*} Interpolationszeit in ms.
1.Ausgang: {*signal*} Filter-Ausgang.

---

para_bp2~ : parametrischer Bandpass 2. Ordnung

Funktionsweise:

Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Gleiches gilt für die Parameter Güte und Gain, wobei für Gain hier zuerst die dB-Werte linear interpoliert werden und dann erst die Effektiv-Wert-Konvertierung

statt findet. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Resonanz-Frequenz in Hz;

2.Argument: <*float*> Güte (ohne Einheit);

3.Argument: <*float*> Gain, Resonanz-Überhöhung oder Abschwächung in dB;

4.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Resonanz-Frequenz in Hz;

3.Eingang: {*float*} Güte (ohne Einheit);

4.Eingang: {*float*} Gain, Resonanz-Überhöhung oder Abschwächung in dB;

5.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

Für die folgenden Filter gilt:

Die Koeffizienten der einzelnen Kaskaden-Stufen wurden entnommen aus [3].

---

hp2_butt~ , hp3_butt~ , hp4_butt~ , hp5_butt~ ;

hp6_butt~ , hp7_butt~ , hp8_butt~ , hp9_butt~ ;

hp10_butt~ : Hochpass 2. bis 10.Ordnung mit Butterworth-Kennlinie

Funktionsweise:

Butterworth-Charakteristik des Frequenzgangs bedeutet, dass die Kennlinie, von hohen Frequenzen ausgehend, möglichst lange bei 0 dB bleibt, und ab der Grenzfrequenz sich möglichst schnell der Kennlinie +20dB/(Ordnung*Decade) nähert. Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz-Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz-Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

hp2_cheb~ , hp3_cheb~ , hp4_cheb~ , hp5_cheb~ ;

hp6_cheb~ , hp7_cheb~ , hp8_cheb~ , hp9_cheb~ ;

hp10_cheb~ : Hochpass 2. bis 10.Ordnung mit Tschebyscheff-Kennlinie

Funktionsweise:

Tschebyscheff-Charakteristik Typ I des Frequenzgangs bedeutet, dass die Kennlinie, von hohen Frequenzen ausgehend, möglichst lange innerhalb des Welligkeits-Bereich bleibt, und ab der Grenzfrequenz sich möglichst steil absenkt, und erst dann sich der Kennlinie +20dB/(Ordnung*Decade) nähert. Bei geraden Ordnungen liegt der Welligkeits-Bereich zwischen 0 dB und +1 dB, bei ungeraden Ordnungen liegt er zwischen -1 dB und 0 dB. Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz-Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz-Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

hp2_bess~ , hp3_bess~ , hp4_bess~ , hp5_bess~ ;

hp6_bess~ , hp7_bess~ , hp8_bess~ , hp9_bess~ ;

hp10_bess~ : Hochpass 2. bis 10.Ordnung mit Bessel-Kennlinie

Funktionsweise:

Bessel-Charakteristik der Gruppenlaufzeit bedeutet, dass die Kennlinie, von hohen Frequenzen ausgehend, möglichst lange konstant gehalten wird, und erst ab der Grenzfrequenz sich verändern darf. Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz-Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz-Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

hp2_crit~ , hp3_crit~ , hp4_crit~ , hp5_crit~ ;

hp6_crit~ , hp7_crit~ , hp8_crit~ , hp9_crit~ ;

hp10_crit~ : Hochpass 2. bis 10.Ordnung mit kritischer Dämpfung

Funktionsweise:

Kritische Dämpfung bedeutet, dass die Sprungantwort keine Überhöhung bzw. kein Ausschwingverhalten aufweist. Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz-Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz-Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

lp2_butt~ , lp3_butt~ , lp4_butt~ , lp5_butt~ ;

lp6_butt~ , lp7_butt~ , lp8_butt~ , lp9_butt~ ;

lp10_butt~ : Tiefpass 2. bis 10.Ordnung mit Butterworth-Kennlinie

Funktionsweise:

Butterworth-Charakteristik des Frequenzgangs bedeutet, dass die Kennlinie, von tiefen Frequenzen ausgehend, möglichst lange bei 0 dB bleibt, und ab der Grenzfrequenz sich möglichst schnell der Kennlinie -20dB/(Ordnung*Decade) nähert. Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz-Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz-Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

lp2_cheb~ , lp3_cheb~ , lp4_cheb~ , lp5_cheb~ ;

lp6_cheb~ , lp7_cheb~ , lp8_cheb~ , lp9_cheb~ ;

lp10_cheb~ : Tiefpass 2. bis 10.Ordnung mit Tschebyscheff-Kennlinie

Funktionsweise:

Tschebyscheff-Charakteristik Typ I des Frequenzgangs bedeutet, dass die Kennlinie, von tiefen Frequenzen ausgehend, möglichst lange innerhalb des Welligkeits-Bereich bleibt, und ab der Grenzfrequenz sich möglichst steil absenkt, und erst dann sich der Kennlinie -20dB/(Ordnung*Decade) nähert. Bei geraden Ordnungen liegt der Welligkeits-Bereich zwischen 0 dB und +1 dB, bei ungeraden Ordnungen liegt er zwischen -1 dB und 0 dB. Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz-Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz-Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

lp2_bess~ , lp3_bess~ , lp4_bess~ , lp5_bess~ ;

lp6_bess~ , lp7_bess~ , lp8_bess~ , lp9_bess~ ;

lp10_bess~ : Tiefpass 2. bis 10.Ordnung mit Bessel-Kennlinie

Funktionsweise:

Bessel-Charakteristik der Gruppenlaufzeit bedeutet, dass die Kennlinie, von tiefen Frequenzen ausgehend, möglichst lange konstant gehalten wird, und erst ab der Grenzfrequenz sich verändern darf. Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die

bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz-Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz-Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

lp2_crit~ , lp3_crit~ , lp4_crit~ , lp5_crit~ ;

lp6_crit~ , lp7_crit~ , lp8_crit~ , lp9_crit~ ;

lp10_crit~ : Tiefpass 2. bis 10.Ordnung mit kritischer Dämpfung

Funktionsweise:

Kritische Dämpfung bedeutet, dass die Sprungantwort keine Überhöhung bzw. kein Ausschwingverhalten aufweist. Bei Änderung der Frequenz wird binnen der Interpolationszeit in 1.45 ms (bei Blocksize = 64; Samplerate = 44100 Hz) Schritten die bisherige Frequenz auf die neue Frequenz logarithmisch interpoliert. Bei jeder Neuberechnung der Koeffizienten wird die Stabilität des Filters überprüft.

Übergabe-Parameter:

1.Argument: <*float*> Grenz-Frequenz in Hz;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*float*} Grenz-Frequenz in Hz;

3.Eingang: {*float*} Interpolationszeit in ms.

1.Ausgang: {*signal*} Filter-Ausgang.

---

vcf_hp2~ , vcf_hp4~ , vcf_hp6~ ;

vcf_hp8~ : Hochpass 2. bis 8. Ordnung mit Frequenz- und Q-Signaleingängen

vcf_lp2~ , vcf_lp4~ , vcf_lp6~ ;

vcf_lp8~ : Tiefpass 2. bis 8.Ordnung mit Frequenz- und Q-Signaleingängen

Funktionsweise:

Die Vorsilbe vcf stammt aus Zeiten der Analog-Synthesizer und bedeutet Voltage Controlled Filter. Filter höherer Ordnung als 2 werden zusammengesetzt aus Filterstufen 2. Ordnung. Die effektive Güte dieser Filterstufen beträgt: Q radiziert durch N/2. Bei den als Signal vorliegenden Werten von Frequenz und/oder Güte werden nur alle 4 Samples neue Koeffizienten berechnet (4-fach Downsampling). Die Stabilität des Filters wird nicht überprüft, daher wird empfohlen, den Frequenzbereich von 10 bis 17000 Hz nicht zu verlassen.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;
2.Eingang: {*signal*} Grenz-Frequenz in Hz;
3.Eingang: {*signal*} Güte (ohne Einheit).
1.Ausgang: {*signal*} Filter-Ausgang.

---

vcf_bp2~ , vcf_bp4~ , vcf_bp6~ ;

vcf_bp8~ : Bandpass 2. bis 8.Ordnung mit Frequenz- und Q-Signaleingängen

vcf_rbp2~ , vcf_rbp4~ , vcf_rbp6~ , vcf_rbp8~ :

Resonanz-Bandpass 2. bis 8.Ordnung mit Frequenz- und Q-Signaleingängen

Funktionsweise:

Die Vorsilbe vcf stammt aus Zeiten der Analog-Synthesizer und bedeutet Voltage Controlled Filter. Die normalen Bandpässe weisen bei Resonanzfrequenz eine Verstärkung von 1 auf; während die Resonanz-Bandpässe eine Verstärkung proportional zu Q erzeugen. Filter höherer Ordnung als 2 werden zusammengesetzt aus Filterstufen 2. Ordnung. Die effektive Güte dieser Filterstufen wurde multiplikativ verringert, damit für das Gesamtfilter die Formel: Q = Resonanzfrequenz / Bandbreite erfüllt wird. Bei den als Signal vorliegenden Werten von

Frequenz und/oder Güte werden nur alle 4 Samples neue Koeffizienten berechnet (4-fach Downsampling). Die Stabilität des Filters wird nicht überprüft, daher wird empfohlen, den Frequenzbereich von 10 bis 17000 Hz nicht zu verlassen.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*signal*} Filter-Eingang;

2.Eingang: {*signal*} Grenz-Frequenz in Hz;

3.Eingang: {*signal*} Güte (ohne Einheit).

1.Ausgang: {*signal*} Filter-Ausgang.

3.2 Arithmetische Objekte

addl~ : Signal-Addition mit line~

divl~ : Signal-Divison mit line~

mull~ : Signal-Multiplikation mit line~

subl~ : Signal-Subtraktion mit line~

Funktionsweise:

Der linke Signal-Eingang wird mit rechten <*float*>-Message-Eingang, der intern mittels line~-Funktion zeitlich interpoliert wird, mit einer der 4 Grundrechnungsarten verknüpft, das Signal-Resultat wird am Ausgang ausgegeben.

Übergabe-Parameter:

1.Argument: <*float*> 2. Operand für die Grundrechnungsart;

2.Argument: <*float*> Interpolationszeit in ms (optional, sonst 0 ms).

Eingänge und Ausgänge:

1.Eingang: {signal} 1. Operand;

2.Eingang: {*float*} 2. Operand;

3.Eingang: {*float*} Interpolationszeit für 2. Operanden in ms.

1.Ausgang: {signal} Ergebnis der Grundrechnungsart.

3.3 „signal-to-float“ Objekte

prvu~ : Spitzenwert und Effektivwert VU-Meter Interface

Funktionsweise:

Dreierlei Werte werden berechnet und ausgegeben: der Effektivwert, der Spitzenwert und ein Zählerwert für die Anzahl an Übersteuerungen pro Metronom-Intervall. Der Effektivwert steigt und fällt mit der Zeitkonstante von Abkling-Zeitkonstante, der Spitzenwert steigt unmittelbar an, bleibt dann Spitzenwert-Einfrierzeit lang konstant und fällt mit Abkling-Zeitkonstante ab. Der Übersteuerungszähler wird inkrementiert, falls der zeitlich unverlängerte Spitzenwert binnen eines Metronom-Intervalls den Wert von Übersteuerungs-Schwellwert überschreitet.

Übergabe-Parameter:

1.Argument: <*float*> Ausgabe-Metronomzeit in ms (optinal, sonst: 300 ms);

2.Argument: <*float*> Spitzenwert-Einfrierzeit in ms (optinal, sonst: 1000 ms);

3.Argument: <*float*> Abkling-Zeitkonstante in ms (optinal, sonst: 300 ms);

4.Argument: <*float*> Übersteuerungs-Schwellwert in dB (optinal, sonst: -0.01 dB).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Eingangssignal;

oder {*t_metro*} + <*float*> Ausgabe-Metronomzeit in ms;

oder {*t_hold*} + <*float*> Spitzenwert-Einfrierzeit in ms;

oder {*t_release*} + <*float*> Abkling-Zeitkonstante in ms;

oder {*threshold*} + <*float*> Übersteuerungs-Schwellwert in dB;

oder {*reset*} setzt interne Werte von Effektiv- und Spitzenwert auf -99.9 dB, und den Übersteuerungszähler auf 0;

oder {*start*} startet Ausgabe Metronom;

oder {*stop*} stoppt Ausgabe Metronom;

oder {*float*} 0 stoppt, 1 startet Ausgabe Metronom;

1.Ausgang: {*list*}+ <*float*> Effektivwert + <*float*> Spitzenwert + + <*float*> Anzahl der Übersteuerungen.

pvu~ : Spitzenwert VU-Meter Interface

Funktionsweise:

Zweierlei Werte werden berechnet und ausgegeben: der Spitzenwert und ein Zählerwert für die Anzahl an Übersteuerungen pro Metronom-Intervall. Der Spitzenwert steigt unmittelbar an und fällt mit Abkling-Zeitkonstante ab. Der Übersteuerungszähler wird inkrementiert, falls der zeitlich unverlängerte Spitzenwert binnen eines Metronom-Intervalls den Wert von Übersteuerungs-Schwellwert überschreitet.

Übergabe-Parameter:

1.Argument: <*float*> Ausgabe-Metronomzeit in ms (optinal, sonst: 300 ms);

2.Argument: <*float*> Abkling-Zeitkonstante in ms (optinal, sonst: 300 ms);

3.Argument: <*float*> Übersteuerungs-Schwellwert in dB (optinal, sonst: -0.01 dB).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Eingangssignal;

oder {*t_metro*} + <*float*> Ausgabe-Metronomzeit in ms;

oder {*t_release*} + <*float*> Abkling-Zeitkonstante in ms;

oder {*threshold*} + <*float*> Übersteuerungs-Schwellwert in dB;

oder {*reset*} setzt internen Wert von Spitzenwert auf -199.9 dB,
und den Übersteuerungszähler auf 0;

oder {*start*} startet Ausgabe Metronom;

oder {*stop*} stoppt Ausgabe Metronom;

oder {*float*} 0 stoppt, 1 startet Ausgabe Metronom;

1.Ausgang: {*float*}+ Spitzenwert;

2.Ausgang: {*float*}+ Anzahl der Übersteuerungen.

---

rvu~ : Effektivwert VU-Meter Interface

Funktionsweise:

Der Effektivwert wird folgendermaßen ermittelt: blockweise wird die Energie berechnet und binnen der Metronomzeit aufaddiert, durch die Anzahl von Samples dividiert, einer Message-Tiefpass-Filterung unterzogen, in dB gewandelt und ausgegeben.

Übergabe-Parameter:

1.Argument: <*float*> Ausgabe-Metronomzeit in ms (optinal, sonst: 300 ms);

2.Argument: <*float*> Abkling-Zeitkonstante in ms (optinal, sonst: 300 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Eingangssignal;
oder {*t_metro*} + <*float*> Ausgabe-Metronomzeit in ms;
oder {*t_release*} + <*float*> Abkling-Zeitkonstante in ms;
oder {*reset*} setzt internen Wert von Effektivwert auf -99.9 dB;
oder {*start*} startet Ausgabe Metronom;
oder {*stop*} stoppt Ausgabe Metronom;
oder {*float*} 0 stoppt, 1 startet Ausgabe Metronom;
1.Ausgang: {*float*}+ Effektivwert.

---

unsig~ : Signal zu <*float*>-Message Konverter

Funktionsweise:

Im Abstand von Metronomzeit wird der erste Wert eines Signal-Blocks als {*float*}-Message ausgegeben.

Übergabe-Parameter:

1.Argument: <*float*> Ausgabe-Metronomzeit in ms (optinal, sonst: 300 ms).

Eingänge und Ausgänge:

1.Eingang: {*signal*} Eingangssignal.
1.Ausgang: {*float*}+ erster Wert eines Signal-Blocks.

3.4 Allgemeine Objekte

fade~ : Formgeber für Signal Ein-/Aus-Blendungskurven (benötigt **line~**)

Funktionsweise:

fade~ stellt eine Transformationsfunktion dar, mit einem Eingangswertebereich von 0 bis 1. Das Übergabeargument bzw. der Inhalt der {set}-Message sind 6 verschiedene Symbole:

_lin : Lineare Übertragungsfunktion;
_linsqrt : Linearer Übertragungsfunktion potenziert mit 3/4;
_sqrt : Linearer Übertragungsfunktion potenziert mit 1/2;
_sin : Viertelperiode einer Sinusfunktion;
_sinhann : Viertelperiode einer Sinusfunktion potenziert mit 3/2;
_hann : Viertelperiode einer Sinusfunktion potenziert mit 2.

Übergabe-Parameter:

1.Argument: <*symbol*> Name der Transformations-Kurve.

Eingänge und Ausgänge:

1.Eingang: {*signal*} : Rampen-Verlauf zwischen 0 und 1;

oder {*set*} + <*symbol*> Name der Transformations-Kurve;

1.Ausgang: {*signal*} : Verzerrte Rampe.

---

iem_blocksize~ : liefert die Signal-Block-Länge eines Fensters in Samples

Funktionsweise:

iem_blocksize~ muss man in einem Pd-Fenster an ein Objekt mit Signal-Ausgang anschliessen. Wenn man nun die DSP-Engine neu einschaltet, gibt dieses Objekt die jeweilige Größe der Signal-Block-Länge in Samples aus.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*signal*} Signal-Leitung, der die Information entnommen werden soll.

1.Ausgang: {*float*} Signal-Block-Länge in Samples.

---

iem_samplerate~ : liefert die Samplerate eines Fensters in Hz

Funktionsweise:

iem_samplerate~ muss man in einem Pd-Fenster an ein Objekt mit Signal-Ausgang anschliessen. Wenn man nun die DSP-Engine neu einschaltet, gibt dieses Objekt die jeweilige Samplerate in Hz aus.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*signal*} Signal-Leitung, der die Information entnommen werden soll.

1.Ausgang: {*float*} Samplerate in Hz.

LFO_noise~ : bandbegrenztes weisses Rauschen mit Grenzfrequenz

Funktionsweise:

Die Erzeugung dieses bandbegrenzten weissen Rauschens wird folgendermaßen bewerkstelligt: proportional zur Grenzfrequenz wird immer wieder ein neuer Zufallswert

zwischen -1 und +1 gebildet und mittels einer 2-Punkt-Interpolation die Zwischensamples berechnet und ausgegeben.

Übergabe-Parameter:

1.Argument: <*float*> Grenzfrequenz des weissen Rauschens.

Eingänge und Ausgänge:

1.Eingang: {*float*} + Grenzfrequenz des weissen Rauschens.

1.Ausgang: {*signal*} : bandbegrenztes weisses Rauschen.

---

peakenv~ : Spitzenwert-Hüllkurven-Folger

Funktionsweise:

Zuerst wird das Eingangssignal gleichgerichtet, das interne Signal mit einem Abklingwert (< 1) multipliziert und dann folgendes entschieden: ist das gleichgerichtete Signal größer als das interne, exponentiell abklingende Signal, so wird der Eingangswert unmittelbar auf den internen Wert übertragen und ausgegeben; ist es kleiner oder gleich, wird das interne abklingende Signal ausgegeben.

Übergabe-Parameter:

1.Argument: <*float*> Abkling-Zeitkonstante in ms.

Eingänge und Ausgänge:

1.Eingang: {*signal*} Eingangssignal;

oder {*reset*} setzt internen Wert und Ausgang auf null.

2.Eingang: {*float*} Abkling-Zeitkonstante in ms.

1.Ausgang: {*signal*} geglättetes Spitzenwert-Signal.

---

pink~ : rosa Rausch-Generator

Funktionsweise:

Weisses Rauschen wird mittels 6 high-shelving Filterstufen, die logarithmisch über den Hörbereich verteilt sind, gefiltert, wodurch sich ein Betrags-Frequenzgang von -3 dB pro Oktave ergibt (mit einer Welligkeit von weniger als +-1 dB). Zeitlicher Spitzenwert ist um -4 dB, Effektivwert ist um -16 dB.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Ausgang: {*signal*} rosa Rausch-Signal.

---

sin_phase~ : berechnet die Phasendifferenz zweier Sinus-Signale in Samples

Funktionsweise:

Unter der Annahme, dass beide eingehenden Signale sinusförmig sind, wird die Phasendifferenz bezüglich der Nulldurchgänge berrechnet und als Vielfache Ganzzahl der Abtastperiode ausgegeben.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*signal*} Phasenverschobenes Signal;

2.Eingang: {*signal*} Referenz-Sinus-Signal.

1.Ausgang: {*signal*} Phasendifferenz in Samples.

---

4 Message-verarbeitende Objekte

4.1 Gleitkomma-basierende Objekte (float)

1p1z : IIR-Filter 1.Ordnung für einen Parameter-Wert

Funktionsweise:

Dient zur Filterung von {*float*}-Messages. Es ist aufgebaut wie ein IIR-Filter 1. Ordnung mit einer Polstelle und einer Nullstelle und gehorcht folgender Differenzen-Gleichung:.

$out_0 = a_0*in_0 + a_1*in_1 + b_1*out_1$

$out_1 = out_0$

$in_1 = in_0$

Das Objekt besitzt selbst keine Abtastrate, es wird also von der Rate der eingehenden {*float*}-Messages getaktet.

Übergabe-Parameter:

1.Argument: <*float*> a_0 erster Zähler-Term;

2.Argument: <*float*> a_1 zweiter Zähler-Term;

3.Argument: <*float*> b_1 erster Nenner-Term.

Eingänge und Ausgänge:

1.Eingang: {*float*} + neuer Eingangswert in_0;

2.Eingang: {*float*} + a_0 zweiter Zähler-Term;

3.Eingang: {*float*} + a_1 erster Nenner-Term;

4.Eingang: {*float*} + b_1 erster Zähler-Term;

5.Eingang: {*float*} + in_1 alter Eingangswert;

6.Eingang: {*float*} + out_1 alter Ausgangswert.

1.Ausgang: {*float*} + gefilterter Wert out_0.

---

bpe : zeitliche Ablauf-Steuereinheit für einen Parameter

Funktionsweise:

Dieses Objekt in Kombination mit einem line- oder line~-Objekt generiert lineare Verläufe eines Parameters über die Zeit (break-point-envelope). Eine {*list*}-Message bestehend aus einer Anzahl N von <*float*>-Paaren (Wert + Zeit) bestimmt den Polygonzugverlauf. Diese Message muss immer mit dem Paar „Initialwert + 0 ms“ beginnen (die Zeit wird immer in ms relativ zum letzten Breakpoint gemessen), und wird in einem Arbeitsspeicher gespeichert. Aktiviert wird der Ablauf durch eine {*bang*}-Message. Gelangt innerhalb des Abarbeitens eine weitere {*list*}-Message ein, wird diese in einen Hilfsspeicher abgelegt und erst beim nächsten Start in den Arbeitsspeicher übertragen. Nachdem das letzte Wert-Zeit-Paar verarbeitet wurde (der Endwert in Zeit erreicht wurde), erscheint eine {*bang*}-Message am 3.Ausgang; zuvor wurde auf den Ausgängen 1 und 2 immer die Wert-in-Zeit-Paare ausgegeben. Eine {*bang*}-Message während eines Ablaufs, bricht den aktuellen Ablauf ab und startet ihn neu.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*list*} + N * (<*float*> Wert + <*float*> Zeit in ms);

oder {*bang*} startet den Ablauf;

oder {*stop*} stoppt den Ablauf.

1.Ausgang: {*float*} + Endwert der Rampe;

2.Ausgang: {*float*} + Dauer der Rampe;

3.Ausgang: {*bang*} zeigt Ende des Ablaufes an.

---

db2v : dB zu Effektivwert Konverter

Funktionsweise:

Wandelt technische dB in Effektivwert um (0 dB entspricht 1, -6 dB entspricht 0.5). Unterhalb von -199.9 dB wird 0 ausgegeben.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*float*} + technische dB.

1.Ausgang: {*float*} + Effektivwert.

---

v2db : Effektivwert zu dB Konverter

Funktionsweise:

Wandelt Effektivwert in technische dB um (1 entspricht 0 dB, 0.5 entspricht -6 dB). Unterhalb von 0 werden -199.9 dB ausgegeben.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*float*} + Effektivwert.

1.Ausgang: {*float*} + technische dB.

---

dbtofad : MIDI-dB zu Fader-Verlauf Konverter

Funktionsweise:

Wandelt MIDI-dB in eine geometrische Pixel-Auslenkung um. Diese Auslenkung entspricht einer VU-Meter-Skala bzw. einer Mischpult-Fader-Skala. Hier nachfolgend einige Entsprechungen:

120 MIDI-dB ~ 20 dB ~ 10.0 ~ 125 pixel
106 MIDI-dB ~ 6 dB ~ 2.0 ~ 108 pixel
100 MIDI-dB ~ 0 dB ~ 1.0 ~ 84 pixel
94 MIDI-dB ~ -6 dB ~ 0.5 ~ 60 pixel
80 MIDI-dB ~ -20 dB ~ 0.1 ~ 36 pixel
60 MIDI-dB ~ -40 dB ~ 0.01 ~ 18 pixel
40 MIDI-dB ~ -60 dB ~ 0.001 ~ 6 pixel
0 MIDI-dB ~ -inf dB ~ 0.0 ~ 0 pixel.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*float*} + MIDI-dB.

1.Ausgang: {*float*} + Fader-Skala-Auslenkung in pixel.

---

fadtodb : Fader-Verlauf zu MIDI-dB Konverter

Funktionsweise:

Wandelt eine geometrische Pixel-Auslenkung in MIDI-dB um. Siehe **dbtofad**.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*float*} + Fader-Skala-Auslenkung in pixel.

1.Ausgang: {*float*} + MIDI-dB.

---

fadtorms : Fader-Verlauf zu Effektivwert Konverter

Funktionsweise:

Wandelt eine geometrische Pixel-Auslenkung in Effektivwert um. Siehe **dbtofad**.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*float*} + Fader-Skala-Auslenkung in pixel.

1.Ausgang: {*float*} + Effektivwert.

---

rmstofad : Effektivwert zu Fader-Verlauf Konverter

Funktionsweise:

Wandelt Effektivwert in verzerrte MIDI-dB eines Mischpult-Faders um (100 dB entspricht 1, 94 dB entspricht 0.5). Unterhalb von 0 werden 0 dB ausgegeben.Siehe **dbtofad**.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*float*} + Effektivwert.

1.Ausgang: {*float*} + Fader-Skala-Auslenkung in pixel.

---

round_zero : rundet Zahlen im Bereich um null auf null

Funktionsweise:

Das Übergabeargument beschreibt die positive Grenze eines symmterischen Bereichs bezüglich 0. round_zero rundet eingehende Zahlen, die sich innerhalb dieses Bereichs befinden, exakt zu 0. Zahlen, die außerhalb dieses Gebiets befinden bleiben unbehelligt.

Übergabe-Parameter:

1.Argument: <*float*> positiver Grenzwert des Rundungs-Gebiets.

Eingänge und Ausgänge:

1.Eingang: {*float*} + die zu rundende Zahl.

1.Ausgang: {*float*} + die eventuell gerundete Zahl.

---

speedlim : reduziert die Daten-Durchlaufrate von <*float*>-Messages

Funktionsweise:

Das Übergabeargument bestimmt die Abtastperiodendauer in ms. Im Prinzip ist speedlim ein event-bedingtes Downsampling-Objekt, das höhere Event-Raten auf maximal eine {*float*}-Message pro Abtastperiodendauer reduziert, und zuletzt den aktuellsten Eingangswert ausgibt. Falls längere Zeit keine Eingangs-Message hereinkam, bleibt auch der Ausgang stumm.

Übergabe-Parameter:

1.Argument: <*float*> Abtastperiodendauer in ms.

Eingänge und Ausgänge:

1.Eingang: {*float*} + Eingangswert;

2.Eingang: {*float*} + Abtastperiodendauer in ms.

1.Ausgang: {*float*} + eventuell downgesampelter Eingangswert.

---

split3 : teilt eine {*float*}-Message einem von 3 Wertebereichen zu

Funktionsweise:

Falls Eingangswert größer gleich dem unteren Grenzwert und kleiner dem oberen Grenzwert ist, wird er auf dem ersten Ausgang ausgegeben. Ist der Eingangswert kleiner als der untere Grenzwert, wird er auf dem zweiten Ausgang ausgegeben, ist er größer gleich dem oberen Grenzwert, wird er auf dem dritten Ausgang ausgegeben. Die jeweils anderen 2 Ausgänge bleiben dabei inakiv.

Übergabe-Parameter:

1.Argument: <*float*> untere Grenze;

2.Argument: <*float*> obere Grenze.

Eingänge und Ausgänge:

1.Eingang: {*float*} + Eingangswert;

2.Eingang: {*float*} + untere Grenze;

3.Eingang: {*float*} + obere Grenze.

1.Ausgang: {*float*} + Eingangswert innerhalb beider Grenzen;

2.Ausgang: {*float*} + Eingangswert kleiner unterer Grenze;

3.Ausgang: {*float*} + Eingangswert größer gleich oberer Grenze.

---

split : teilt eine {*float*}-Message einem von 2 Wertebereichen zu (wie **moses**)

Funktionsweise:

Falls Eingangswert kleiner dem Grenzwert ist, wird er auf dem ersten Ausgang ausgegeben. Ist der Eingangswert größer gleich dem Grenzwert, wird er auf dem zweiten Ausgang ausgegeben. Der jeweils andere Ausgang bleibt dabei inakiv.

Übergabe-Parameter:

1.Argument: <*float*> Grenzwert.

Eingänge und Ausgänge:

1.Eingang: {*float*} + Eingangswert;

2.Eingang: {*float*} + Grenzwert.

1.Ausgang: {*float*} + Eingangswert kleiner als Grenzwert;

2.Ausgang: {*float*} + Eingangswert größer gleich als Grenzwert.

4.2 „symbol“-basierende Objekte

---

mergefilename : fügt Liste zu einer Symbol-Buchstabenkette zusammen

Funktionsweise:

fügt eine {*list*} von <*float*>- und/oder <*symbol*>-Elementen zu einem langen <*symbol*> zusammen. An den Verbindungungsstellen wird optional ein Trennzeichen eingefügt.

Übergabe-Parameter:

1.Argument: <*symbol*> Separier-Buchstabe = Trennzeichen (optional, sonst leer).

Eingänge und Ausgänge:

1.Eingang: {*list*} von <*float*>- und/oder <*symbol*>-Elementen;
{*separator*} + <*symbol*> Separier-Buchstabe = Trennzeichen.

1.Ausgang: {*symbol*} zu einer Buchstabenkette zusammengefügte Liste samt Trennzeichen.

---

splitfilename : trennt ein Symbol in 2 Teile auf

Funktionsweise:

trennt ein eingehendes <*symbol*> in 2 Teile auf. Als Trennzeichen gilt das erstmalige Auftreten des Separier-Buchstabens, beginnend von rechts (String-Ende). Der Teil rechts vom Trennzeichen wird am 2. Ausgang, der Teil links vom Trennzeichen wird am 1. Ausgang als {*symbol*} ausgegeben. Das Trennzeichen selbst wird nicht ausgegeben.

Übergabe-Parameter:

1.Argument: <*symbol*> Trennzeichen (optional, sonst „/“ = slash, gilt auch unter Windows als Trennzeichen diverser Ordner).

Eingänge und Ausgänge:

1.Eingang: {*symbol*} Buchstabenkette (absoluter Datei-Pfad + Name);
{*separator*} + <*symbol*> Trennzeichen = Separier-Buchstabe.

1.Ausgang: {*symbol*} linker Teil des eingehenden Symbols (Pfad);

2.Ausgang: {*symbol*} rechter Teil des eingehenden Symbols (Datei-Name).

---

stripfilename : entfernt Buchstaben aus einer Buchstabenkette

Funktionsweise:

wenn Parameter strip-index positiv ist, werden die ersten „strip-index“ Buchstaben vom eingehenden Symbol entfernt, wenn Parameter strip-index negativ ist, werden die letzten „strip-index“ Buchstaben vom eingehenden Symbol entfernt und anschließend der Rest als {*symbol*} ausgegeben.

Übergabe-Parameter:

1.Argument: <*float*> strip-index.

Eingänge und Ausgänge:

1.Eingang: {*symbol*} Buchstabenkette;
{*set*} + <*float*> strip-index.

1.Ausgang: {*symbol*} Rest der Buchstabenkette.

---

unsymbol : wandelt eine <*symbol*>- in eine <anything>-Message um

Funktionsweise:

wandelt ein eingehendes {*symbol*} in eine {*anything*}-Message um (Inhalt von Symbol wird zum Selektor-Symbol einer Message). Wird vor dem Objekt route manchmal gebraucht.

Keine Übergabe-Parameter.

Eingänge und Ausgänge:

1.Eingang: {*symbol*} Buchstabenfeld.

1.Ausgang: {*Buchstabenfeld*} + leere Liste = eingehendes Symbol als Selektor.

4.3 „anything“-basierende Objekte

---

any : universeller Speicher für jegliche Einzel-Message (wie **f** oder **symbol**)

Funktionsweise:

any hat 2 Eingänge, der linke heiße Eingang gibt jede eingehende Message zum Ausgang weiter (und speichert sie intern), der rechte kalte Eingang speichert nur die eingehende Message. Diese kann durch einen {*bang*} am linken Eingang wieder abgerufen werden.

Übergabe-Parameter:

N Argumente: Liste von <*symbol*>- oder <*float*>-Einträgen, die intern gespeichert werden.

Eingänge und Ausgänge:

1.Eingang: {*bang*} gibt die intern gespeicherte Message am Ausgang aus.
{*float*} gibt {*float*}-Message am Ausgang aus.
{*symbol*} gibt {*symbol*}-Message am Ausgang aus.
{*list*} gibt {*list*}-Message am Ausgang aus.
{*anything*} gibt {*anything*}-Message am Ausgang aus.

2.Eingang: {*bang*} löscht internen Speicher.
{*float*} speichert {*float*}-Message im internen Speicher.
{*symbol*} speichert {*symbol*}-Message im internen Speicher.
{*list*} speichert {*list*}-Message im internen Speicher.
{*anything*} speichert {*anything*}-Message im internen Speicher.
1.Ausgang: jegliche Message.

---

iem_append : fügt Message einen Nachspann hinzu (ehemals **merge_any**)

Funktionsweise:

iem_append hat 2 Eingänge, einer eingehenden Message am linken heißen Eingang wird die intern gespeicherte Message (appendix) angeheftet und zum Ausgang weitergeleitet, der rechte kalte Eingang speichert die eingehende Message (appendix) intern ab.

Übergabe-Parameter:

1.Argument: Liste von <*symbol*>- oder <*float*>-Einträgen,
die intern gespeichert werden (appendix).

Eingänge und Ausgänge:

1.Eingang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} Vorderteil der Message, die ausgegeben wird;
Ausgang wird aktiviert.
2.Eingang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} Hinterteil der Message, die ausgegeben wird;
Ausgang bleibt inaktiv.
1.Ausgang: jegliche Message; zusammengesetzt aus Vorderteil und Hinterteil.

---

iem_prepend : fügt Message einen Vorspann hinzu (Abbk. **pp** , **prepend**)

Funktionsweise:

iem_prepend hat 2 Eingänge, die intern gespeicherte Message (prependix) wird einer, am linken heißen Eingang, eingehenden Message vorangestellt und zum Ausgang weitergeleitet, der rechte kalte Eingang speichert die eingehende Message intern ab (prependix).

Übergabe-Parameter:

1.Argument: Liste von <*symbol*>- oder <*float*>-Einträgen,
die intern gespeichert werden (prependix).

Eingänge und Ausgänge:

1.Eingang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} Hinterteil der Message, die ausgegeben wird,
Ausgang wird aktiviert.

2.Eingang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} Vorderteil der Message, die ausgegeben wird,
Ausgang bleibt inaktiv.

1.Ausgang: jegliche Message; zusammengesetzt aus Vorderteil und Hinterteil.

4.4 Initialisierungsobjekte

dollarg : gibt Übergabeargumente des Eltern-Klasse-Objekts aus (Abbk. **$n**)

Funktionsweise:

Dieses Objekt dient dazu, die Übergabe-Argumente des Abstraktions-Patches, in dem dollarg verwendet wird, auszuwerten. Unter der Annahme, dass N <*float*>- und/oder <*symbol*>-Einträge übergeben wurden, kann man mittels einer {*float*}-Message (Index) folgendes bewerkstelligen: falls der Index positiv ist, wird der <*float*>- oder <*symbol*>-Eintrag an der Stelle Index auf dem 1.Ausgang ausgegeben, auf dem 2.Ausgang wird die Index-Zahl selbst

ausgegeben. Falls der Index negativ ist, wird der <*float*>- oder <*symbol*>-Eintrag an der von hinten her gezählten Stelle Index auf dem 1.Ausgang ausgegeben, auf dem 2.Ausgang wird die auf positiv korrigierte Index-Zahl ausgegeben. Falls der Index 0 ist, wird eine {*list*}-Message, gefolgt von allen <*float*>- oder <*symbol*>-Einträgen, am 1.Ausgang ausgegeben, auf Ausgang 2 wird die Gesamtanzahl der Einträge ausgegeben. Eine {*bang*}-Message bewirkt das gleiche wie ein Index 0.

Keine direkten Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*float*} + Index : gibt einen oder mehrere Einträge auf 1.Ausgang,
und am 2.Ausgang den korrigierten Index aus;
{*bang*}: gibt eine {*list*} mit allen <*float*>- oder <*symbol*>-Einträgen auf
1.Ausgang aus, auf 2.Ausgang wird die Anzahl der Einträge ausgegeben.

1.Ausgang: {*list*} + <*symbol*>- oder <*float*>-Eintrag: alle Übergabe-Argumente;
oder {*anything*} (je nach Eintrag) <*symbol*>-Übergabe-Argument
an der Stelle Index;
oder {*float*} (je nach Eintrag) <*float*>-Übergabe-Argument an der Stelle Index;

2.Ausgang: {*float*} Anzahl oder Index der Übergabe-Argumente.

---

dsp : schaltet Audio-Engine ein und berechnet die Auslastung (aka **dsp~**)

Funktionsweise:

Dieses Objekt schaltet die Audio-Engine (DSP) des Rechners EIN oder AUS und berechnet die DSP-Ausnutzung des Pd-Prozesses. Alle halben Sekunden wird der Mittelwert der Ausnutzung (5 Sekunden Glättung) am 1. Ausgang ausgegeben, am 2.Ausgang wird der Spitzenwert ausgegeben (und 4 Sekunden gehalten).

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*float*} + EIN/AUS : schaltet Audio-Engine EIN (1) und AUS (0).

1.Ausgang: {*float*} + Mittelwert-Last;

2.Ausgang: {*float*} + Spitzenwert-Last.

---

float24 : speichert eine <*float*>-Zahl 24 Bit genau

Funktionsweise:

Dient zum Speichern einer Gleitkommazahl mit 24 Bit genauer Mantisse innerhalb eines Pd-Patches. Da Pd eine Zahl auf 6 Dezimalstellen (entspricht ca. 20 Bit) rundet, wenn man einen Patch abspeichert, ist dies eine Möglichkeit, auf 7 bis 8 Dezimalstellen genaue Zahlen zu verwenden. Die gewünschte Zahl ist als Übergabeargument so zu unterteilen, dass nicht mehr als 6 signifikante Dezimalstellen pro Zahl bleiben (zB. 0.987654321 wird zu 0.987654 321). Eine {*bang*}-Message gibt die zusammengesetzten Übergabeargumente als Zahl aus. Eine {*float*}-Message überschreibt die interne Zahl und gibt die neue Zahl sofort aus.

Übergabe-Parameter:

N Argumente: <*float*>-Einträge von einer, in mehrere Stücke unterteilten, Gleitkommazahl.

Eingänge und Ausgänge:

1.Eingang: {*float*} + neuer Wert : überschreibt internen Wert und wird ausgegeben;
oder {*bang*} gibt zusammengesetzte Übergabe-Argumente aus.

1.Ausgang: {*float*} + zusammengesetzter Wert.

---

init : gibt Initial-Message aus (Abbk. **ii**)

Funktionsweise:

Dieses Objekt gibt seine Übergabe-Argumente nach dem vollständigen Laden des ihn nutzenden Pd-Patches aus. Unter der Annahme, dass N <*float*>- und/oder <*symbol*>-Einträge vorhanden sind, wird entsprechend der Anzahl N, bzw. des ersten Eintrags folgende Message ausgegeben: eine {*bang*}-Message, falls kein Eintrag; eine {*float*}-Message, falls nur ein <*float*>-Eintrag vorhanden; eine {*symbol*}-Message, falls die Einträge <*symbol*> und <xxx> vorhanden; eine {*list*}-Message: falls mehrere Einträge, beginndend mit einem <*float*>-Eintrag oder einem Symbol <list>; oder mit einer {*anything*}-Message, falls zuerst ein Symbol-Eintrag war, und noch mehrere Einträge anhängen. Der Inhalt kann später auch noch abgerufen werden mittels einer {*bang*}-Message.

Übergabe-Parameter:

N Argumente: Liste von <*symbol*>- oder <*float*>-Einträgen, die zur Initialisierung dienen.

Eingänge und Ausgänge:

1.Eingang: {*bang*}: gibt alle Einträge aus.

1.Ausgang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} je nach Anzahl und Art der Übergabe-Argumente.

---

once : lässt eine Message nur erstmalig passieren

Funktionsweise:

Once besitzt intern ein anfangs offenes Gatter, das jegliche Message in Richtung Ausgang passieren lässt. War dies der Fall, schließt sich das Gatter und blockiert jede weitere Message. Aufgehoben wird dies durch eine beliebige Message am 2. Eingang.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} einmalig durchlässiger Eingang;

2.Eingang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} Aufhebungs-Eingang der Sperre.

1.Ausgang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} Ausgang des Einmal-Gatters.

4.5 Zählerobjekte

exp_inc : exponentieller / inkrementeller Zähler (ausgelöst durch {*bang*})

Funktionsweise:

exp_inc ist ein Zähler mit sowohl additivem als auch multiplikativem Inkrement. Der Wert des multiplikativem Inkrement ist in Prozent angegeben. Der Ausgangsbereich des Zählers wird zwischen Untergrenze und Obergrenze begrenzt. Gesteuert wird dieser Ausgabe- und Zähl-Vorgang durch eine {*bang*}-Message auf folgende weise:

- Ausgabe von out_0;
- $out_0 = a + (1 + 0.01*m)*out_1$;
- $out_1 = out_0$.

Eine {*reset*}-Message setzt den Wert out_0 auf den Initialwert zurück.

Übergabe-Parameter:

1.Argument: <*float*> Initialwert;
2.Argument: <*float*> Multiplikative Erhöhung in Prozent;
3.Argument: <*float*> Additive Erhöhung;
4.Argument: <*float*> Untergrenze des Ausgangs;
5.Argument: <*float*> Obergrenze des Ausgangs.

Eingänge und Ausgänge:

1.Eingang: {*float*} + Initialwert;
oder {*bang*} gibt Zählerwert aus und inkrementiert ihn;
oder {*reset*} setzt Zählerwert auf Initialwert zurück;
2.Eingang: {*float*} + Multiplikative Erhöhung in Prozent;
3.Eingang: {*float*} + Additive Erhöhung;
4.Eingang: {*float*} + Untergrenze des Ausgangs;
5.Eingang: {*float*} + Obergrenze des Ausgangs.
1.Ausgang: {*float*} + Zählerwert out_0.

---

for++ : inkrementeller Zähler (gesteuert durch internes Metronom)

Funktionsweise:

for++ ist ein Schleifen-Zähler-Objekt mit Anfangswert, Endwert und der jeweiligen Dauer einer Inkrementierung des Zählers. Falls der Endwert kleiner als der Anfangswert ist, wird der

Zählerwert dekrementiert. Der Endwert wird immer inklusive erreicht und ausgegeben. Nachdem der Zähler gestartet wurde durch eine {*bang*}- oder {*start*}-Message wird unmittelbar danach der Anfangswert als Zählerstand ausgegeben. Dann wird zyklisch Metronom-Zeit lang gewartet, der Zählerstand um 1 erhöht oder erniedrigt und ausgegeben, bis dass der Endwert erreicht und ausgegeben wird. Zuletzt wird noch einmal Metronom-Zeit lang gewartet und der „fertig“-Bang ausgegeben.
Eine {*stop*}-Message beendet den Zählvorgang sofort, belässt aber den Zählerstand.

Übergabe-Parameter:

1.Argument: <*float*> auf Ganzzahl abgerundeter Anfangs-Zählerwert;
2.Argument: <*float*> auf Ganzzahl abgerundeter End-Zählerwert;
3.Argument: <*float*> Metronom-Zeit in ms.

Eingänge und Ausgänge:

1.Eingang: {*float*} + auf Ganzzahl abgerundeter Anfangs-Zählerwert;
oder {*bang*} startet den Zählvorgang;
oder {*start*} startet den Zählvorgang;
oder {*stop*} stoppt den Zählvorgang;
2.Eingang: {*float*} + auf Ganzzahl abgerundeter End-Zählerwert;
3.Eingang: {*float*} + Metronom-Zeit in ms.
1.Ausgang: {*float*} + Zählerwert;
2.Ausgang: {*bang*} zeigt, mit einer Verzögerung von einer Metro-Zeit, das Ende des gesamten Zählvorgangs an.

---

modulo_counter : Zirkulär-Zähler (ausgelöst durch {*bang*})

Funktionsweise:

modulo_counter ist ein Zähler mit einem Inkrement von +1, der immer im Kreis zählt. Der Ausgangsbereich des Zählers bewegt sich zwischen 0 und N-1. Zuerst wird der Zählerstand auf den Wert des 2.Übergabearguments bzw. des 2. Eingangs gesetzt, und zwischen 0 und N-1 begrenzt. Eine {*bang*}-Message gibt den aktuellen Zählerstand aus, inkrementiert ihn und fragt ab, ob er grösser-gleich N ist. Falls dies wahr ist, wird der Zählerstand auf 0 gesetzt. Dies kann man elegant formulieren als:

- Ausgabe von out_0;
- $out_0 = out_1 \% N$;
- $out_1 = out_0$.

Übergabe-Parameter:

1.Argument: <*float*> Modulozahl N, ist der positive Grenzwert,
auf den sich der Zählerstand bezieht;

2.Argument: <*float*> Initial-Zählerstand, wird begrenzt zwischen 0 und N-1.

Eingänge und Ausgänge:

1.Eingang: {*float*} + Modulozahl N, ist der positive Grenzwert,
auf den sich der Zählerstand bezieht;

oder {*bang*} gibt Zählerwert aus und inkrementiert ihn;

2.Eingang: {*float*} + Initial-Zählerstand , wird begrenzt zwischen 0 und N-1.

1.Ausgang: {*float*} + Zählerwert out_0.

4.6 Objekte für Parameter-Management

iem_pbank_csv : verwaltet Parameter-Banken im CSV-Format (wie **textfile**)

Funktionsweise:

iem_pbank_csv verwaltet Parameter-Banken und speichert diese Banken als Datei im CSV-Format (Comma Separated Values). Eine Parameter-Bank stellt eine Zeile der Arbeits-Matrix dar, die von 0 beginnend bis N-1 indiziert wird (N Zeilen). Die einzelnen Parameter werden mittels eines Spaltenindex indiziert (0 bis M-1). Die Elemente der Arbeits-Matrix können sowohl <*float*>- als auch <*symbol*>-Einträge sein. Eine {*list*}-Message am 1.Eingang, die mit dem Spaltenindex beginnt, und dem dann m Parameter folgen (maximal bis Zeilenende), schreibt die Parameterwerte in eine Arbeits-Buffer-Zeile, die wiederum durch eine {*store*}-Message in die Arbeits-Matrix übertragen werden kann (und zwar auf die Stelle, auf die der Zeilenindex-Wert vom 2.Eingang verweist). Eine {*recall*}-Message liest eine Zeile der Matrix aus, kopiert den Inhalt in die Arbeits-Buffer-Zeile und gibt Ihn am Ausgang aus. {*read*}- und {*write*}-Messages lesen bzw. speichern den Matrix-Inhalt aus einer Datei bzw. in eine Datei. Zum Dateinamen muss noch das CSV-Format in Form eines 3 Buchstaben-Symbols Übergeben werden. Die Bedeutung der 3 Buchstaben sind:

der 1.Buchstabe bestimmt das Eintrags-Trennzeichen:

- *b* .. Leerzeichen (blank)
- *c* .. Beistrich (colon)
- *s* .. Strich-Punkt (semicolon)
- *t* .. Tabulator (tab)

der 2.Buchstabe bestimmt das Zeilenende-Zeichen:

- *b* .. Leerzeichen + Zeilenumbruch (blank + return)
- *c* .. Beistrich + Zeilenumbruch (colon + return)
- *s* .. Strich-Punkt + Zeilenumbruch (semicolon + return)
- *r* .. nur Zeilenumbruch (return)

der 3.Buchstabe bestimmt die Kodierung des Zeilenumbruchs in Abhängigkeit des Betriebssystems:

- *l* .. Linux
- *w* .. Windows
- *m* .. Macintosh.

Übergabe-Parameter:

1.Argument: <*float*> maximale Anzahl der Parameter-Spalten M;

2.Argument: <*float*> maximale Anzahl der Preset-Zeilen N.

Eingänge und Ausgänge:

1.Eingang: {*bang*} : gibt den Inhalt der ganzen Arbeitsbuffer-Zeile aus;

oder {*list*} + <*float*> Spaltenindex + m mal <*float*>- oder <*symbol*>-Einträge;
speichert die m Einträge in die Arbeitsbuffer-Zeile ab der Stelle Spaltenindex.

oder {*recall*} : kopiert den Inhalt der ganzen Zeile mit Position Zeilenindex (2.Eingang) aus der Arbeits-Matrix in die Arbeitsbuffer-Zeile und gibt diese Zeile aus;

oder {*recall*} + <*float*> Spaltenindex + <*float*> Einträge-Anzahl;
diese Message kopiert nur einen Teil der Zeile aus der Arbeits-Matrix, beginnend von Spaltenindex bis Spaltenindex + Einträge-Anzahl – 1 (maximal bis M-1) in die Arbeitsbuffer-Zeile (gleiche Grenzen) und gibt diesen Teil auch aus;

oder {*store*} : speichert den Inhalt der ganzen Arbeitsbuffer-Zeile in die Zeile an der Stelle Zeilenindex (2.Eingang) der Arbeits-Matrix;

oder {*read*} + <*symbol*> Dateiname + <*symbol*> CSV-Format;
falls die Datei erfolgreich geöffnet und interpretiert werden konnte, kopiert diese Message den Inhalt dieser Datei in die gesamte Arbeits-Matrix; falls zu wenig Einträge vorhanden waren, werden diese mit <*float*> 0 Einträgen ergänzt, falls zu viele Einträge pro Zeile oder pro Spaltenindex vorhanden waren, werden diese ignoriert und übergangen;

oder {*write*} + <*symbol*> Dateiname + <*symbol*> CSV-Format;
falls die Datei erfolgreich angelegt werden konnte, kopiert diese Message den Inhalt der gesamten Arbeits-Matrix in diese Datei; dabei werden alle Einträge einer Zeile durch ein bestimmtes ASCII-Zeichen separiert, und die Zeilen wiederum durch ein eigenes Zeichen terminiert;
(bestimmt mittels CSV-Format)

2.Eingang: {*float*} + Zeilenindex; für {*store*}- oder {*recall*}-Message (beginnend von 0 bis N-1).

1.Ausgang: {*list*} + Einträge-Anzahl mal <*float*>- oder <*symbol*>-Einträge; Ausgabe der Liste von Einträgen einer ganzen Zeile oder eines Teils einer Zeile;

2.Ausgang: {*list*} + <*float*> Spaltenindex + Einträge-Anzahl mal <*float*>- oder <*symbol*>-Einträge; Ausgabe der der gleichen Liste wie auf 1.Ausgang, nur mit einem vorangestelltem <*float*>-Eintrag, der die Zahl Spaltenindex von der {*recall*}-Message übernimmt (falls die ganze Zeile ausgegeben wird, ist dieser Index 0);

---

list2send : wandelt {*list*}-Messages in send-Messages um

Funktionsweise:

list2send dient als Bindeglied zwischen iem_pbank_csv und receive-fähigen GUI-Objekten. Mittels einer {*add*}-Message werden Indexzahlen mit Send-Receive-Namen in Zusammenhang gebracht. Eingehende Messages werden dann anhand ihres Index in Richtung eines entsprechenden receive-Objekts weiter geleitet.

Übergabe-Parameter:

1.Argument: <*float*> maximale Anzahl an versendbaren Parametern M.

Eingänge und Ausgänge:

1.Eingang: {*list*} + <*float*> Index + <*float*>- oder <*symbol*>-Inhalt-Eintrag;
sendet den Inhalt-Eintrag an Receive-Objekte mit dem Symbol-Namen, auf den Index verweist;

oder {*set*} + <*float*> Index + <*float*>- oder <*symbol*>-Inhalt;
sendet eine {set}-Message mit Inhalt-Eintrag an Receive-Objekte mit dem Symbol-Namen, auf den Index verweist;

oder {*all*} + M mal <*float*>- oder <*symbol*>-Einträge;
sendet alle Einträge an die jeweiligen Receive-Objekte mit den Symbol-Namen, die mit den Indizes 0 bis M-1 assoziiert werden;

oder {*set_all*} + M mal <*float*>- oder <*symbol*>-Einträge; sendet M {set}-Messages mit den jeweiligen Einträgen an Receive-Objekte mit den Symbol-Namen, die mit den Indizes 0 bis M-1 assoziiert werden;

oder {*from*} + <*float*> Index + m mal <*float*>- oder <*symbol*>-Einträge; sendet die m Einträge an die jeweiligen Receive-Objekte mit den Symbol-Namen, die mit den Indizes Index bis Index+m-1 assoziiert werden;

oder {*set_from*} + <*float*> Index + m mal <*float*>- oder <*symbol*>-Einträge; sendet m {*set*}-Messages mit den jeweiligen Einträgen an Receive-Objekte mit den Symbol-Namen, die mit den Indizes Index bis Index+m-1 assoziiert werden;

oder {*add*} + <*float*> Index + <*symbol*> Parameter-Send-Name;
die Parameter-Send-Namen werden in einer internen Tabelle an der Stelle Index gespeichert; es wird eine Verbindung hergestellt mit diversen receive-Objekten;

oder {*clear*} : löscht die interne Namens-Tabelle.

Kein Ausgang:

---

receive2list : empfängt receive-Messages und gibt sie als Listen aus

Funktionsweise:

receive2list dient als Bindeglied zwischen send-fähigen GUI-Objekten und iem_pbank_csv. Mittels einer {*add*}-Message werden Indexzahlen mit Send-Receive-Namen in Zusammenhang gebracht. Empfangene Messages werden dann entsprechend der Receive-Namen mit einer Indexzahl versehen und als Liste ausgegeben.

Übergabe-Parameter:

Kein Argument: (in Zukunft: maximale Anzahl an empfangbaren Parametern M, momentan maximal 200)

Eingänge und Ausgänge:

1.Eingang: {*add*} + <*float*> Index + <*symbol*> Parameter-Send-Name;
die Parameter-Receive-Namen werden in einer internen Tabelle an der Stelle Index gespeichert; es wird eine Verbindung hergestellt mit diversen send-Objekten;

oder {*clear*} : löscht die interne Namens-Tabelle.

1.Ausgang: {*list*} + <*float*> Index + <*float*>- oder <*symbol*>-Wert;
falls nur ein einfacher <*float*>- oder <*symbol*>-Wert empfangen wurde;

oder {*list*} + <*float*> Index + <set> + <*float*>- oder <*symbol*>-Wert; falls eine {set}-Message mit <*float*>- oder <*symbol*>-Wert empfangen wurde.

4.7 Allgemeine Objekte

add2_comma : erzeugt Beistrich getrennte Messages in einer Messagebox

Funktionsweise:

fügt einer eingehenden Message am Anfang das Selektor-Symbol {*add2*} und als erstes Element ein <*symbol*> „,“ (Beistrich) voraus. Dies dient dazu, eine Messagebox mit diversen Messages zu bestücken, die dann gleichzeitig aktiviert werden können.

Keine Übergabe-Parameter.

Eingänge und Ausgänge:

1.Eingang: {*bang*} liefert am Ausgang die Message {*add2*} + <*symbol*> „,“;
{*float*} liefert am Ausgang die Message {add2} + <*symbol*> „,“ +
+ <*float*> eingehende Zahl.
{*symbol*} liefert {add2} + <*symbol*> „,“ + <*symbol*> eingehendes Symbol.
{*list*} liefert {add2} + <*symbol*> „,“ + <*symbol*>- oder <*float*>-Einträge der eingehenden Liste.
{*anything*} liefert {add2} + <*symbol*> „,“ + <anything> +
+ <*symbol*>- oder <*float*>-Einträge der eingehenden Message.

1.Ausgang: {*add2*} + <*symbol*> „,“ + eingehende Message.

f2note : Frequenz zu Midi+Cents+Note Konverter

Funktionsweise:

f2note wandelt einen eingehenden Frequenzwert (Hz) um in 3 Ausgangsgrößen: in einen MIDI-Wert am ersten Ausgang, in ein Noten-Symbol (mittels einer {set}-Message) am zweiten Ausgang und in die Cent-Abweichung von der Note am dritten Ausgang. Die Umwandlung findet bezüglich des Kammerton a1 statt, der als Frequenz in Hz übergeben werden muss (sonst 440 Hz). Die Noten einer Oktave sind : C, #C, D, #D, E, F, #F, G, #G, A, #A und H; die Indizierung der Oktaven an Hand des Tons A von tief nach hoch sind: A3, A2, A1, A, a, a1, a2, a3, a4, a5, a6 usw. Wobei a1 dem Übergabeargument bzw. 440 Hz entspricht.

Übergabe-Parameter:

1.Argument: <*float*> Frequenz des Kammerton a1 in Hz (MIDI 69) (sonst 440 Hz).

Eingänge und Ausgänge:

1.Eingang: {*float*} + Eingangsfrequenz.

1.Ausgang: {*float*} + in MIDI-Wert gewandelte Eingangsfrequenz;

2.Ausgang: {*set*} + <*symbol*> Note.

3.Ausgang: {*float*} + Abweichung von Note in cents.

gate : steuerbares Message-Gatter (wie **spigot**)

Funktionsweise:

gate läßt in Abhängigkeit von seinem Steuerparameter eine beliebige Message durch oder blockiert deren Durchgang. Falls der Steuerparameter ungleich 0 ist, wird jegliche Message vom ersten Eingang auf den Ausgang weitergereicht. Falls der Steuerparameter 0 ist, wird diese Message blockiert. Der Steuerparameter kann auch mittels Übergabeargument initialisiert werden.

Übergabe-Parameter:

1.Argument: <*float*> Steuerparameter (0 = Blockade, sonst Durchlass).

Eingänge und Ausgänge:

1.Eingang: {*bang*}
oder {*float*}
oder {*symbol*}
oder {*list*}
oder {*anything*} Eingangs-Message;
2.Eingang: {*float*} + Steuerparameter (0 = Blockade, sonst Durchlass).
1.Ausgang: {*bang*}
oder {*float*}
oder {*symbol*}
oder {*list*}
oder {*anything*} Ausgang für optional durchgelassene Eingangs-Message.

---

iem_i_route : Variation von route-Objekt (Abbk. **iiroute**)

Funktionsweise:

iem_i_route ist eine spezielle Variante des route-Objekts, das geeignet ist für Anwendungen in Abstraktionen. Bedingung dafür ist, dass die Eingangs-Message eine Liste mit einem <*float*>-Eintrag an erster Stelle ist. Die 3 Übergabeargumente bedeuten: Anfangs-Index, End-Index und Offset-Index. Intern wird dann der Offset-Index jeweils zum Anfangs-Index bzw. zum End-Index hinzuaddiert. Die Differenz von End-Index minus Anfangs-Index plus 1 ergibt die Anzahl der Ausgänge mit Abzweig-Eigenschaften, hinzu kommt noch ein weiterer Ausgang (ganz rechts), der die Eingangs-Message abweist, falls der erste <*float*>-Eintrag der Message nicht übereingestimmte mit einem der Indizes, die zwischen Anfang und Ende lagen. Aus den Übergabeargumenten ergeben sich damit folgende Größen:

- Effektiver Anfangs-Index = Anfangs-Index + Offset-Index;
- effektiver End-Index = End-Index + Offset-Index;
- Anzahl der Abzweig-Ausgänge N = End-Index - Anfangs-Index + 1.

Eine abgezweigte Message besitzt nur mehr M Einträge (um den ersten <*float*>-Eintrag verringert). Falls M gleich 0 ist, kommt nur mehr eine {*bang*}-Message heraus, falls M eins ist, kommt eine {*float*}-Message oder eine {*anything*}-Message heraus, falls M größer 1 ist, kommt eine {*list*}-Message oder eine {*anything*}-Message heraus. (Wenn der nunmehr erste Eintrag der Liste ein <*symbol*>-Eintrag ist, wandelt sie sich in eine {*anything*}-Message um)

Übergabe-Parameter:

1.Argument: <*float*> Anfangs-Index;

2.Argument: <*float*> End-Index;

3.Argument: <*float*> Offset-Index, der den beiden vorangegangenen Indizes hinzuaddiert wird (optional, sonst 0).

Eingänge und Ausgänge:

1.Eingang:	{*list*} + <*float*>-Eintrag + beliebiger Rest (M * <*float*> oder <*symbol*>).
1. bis N. Ausgang:	{*bang*}
oder	{*float*}
oder	{*list*}
oder	{*anything*} um den ersten Eintrag verkürzte Eingangs-Message.
N+1.Ausgang:	{*list*} Eingangs-Message, deren erster Eintrag nicht passte, und damit abgewiesen wurde.

---

iem_route : verzweigt Listen in Abhängigkeit ihres ersten Eintrags

Funktionsweise:

iem_route funktioniert fast gleich wie das route-Objekt, bis auf folgenden Unterschied: falls eine {*float*}-Message mit einem der <*float*>-Übergabeargumente übereinstimmt, kommt beim route-Objekt eine Liste mit null Einträgen heraus (was manche Objekte nicht verstehen) und bei iem_route kommt eine {*bang*}-Message heraus. Gleiches gilt, falls eine {*anything*}-Message mit einem der <*symbol*>-Übergabeargumente übereinstimmt.

Übergabe-Parameter:

N Argumente: <*float*>- oder <*symbol*>-Einträge.

Eingänge und Ausgänge:

1.Eingang:	{*float*}
oder	{*symbol*}
oder	{*list*}
oder	{*anything*} Eingangs-Message
1. bis N. Ausgang:	{*bang*}
oder	{*float*}
oder	{*symbol*}
oder	{*list*}
oder	{*anything*} um den ersten Eintrag verkürzte Eingangs-Message.

N+1.Ausgang: {*float*}
oder {*symbol*}
oder {*list*}
oder {*anything*} Eingangs-Message, deren erster Eintrag nicht passte, und damit abgewiesen wurde.

---

iem_receive : Receive-Objekt mit Eingang für Namen (Abbk. **iem_r**)

Funktionsweise:

iem_receive funktioniert fast gleich wie das receive-Objekt, es hat jedoch einen Eingang, mittels dem man den Receive-Namen verändern kann ({*set*} + <*symbol*> Name). Mit einer {*clear*}-Message unterbricht man die bisherige Verbindung.

Übergabe-Parameter:

1.Argument: <*symbol*> Receive-Name.

Eingänge und Ausgänge:

1.Eingang: {*set*} + <*symbol*> Receive-Name; stellt eine Verbindung zu unter „Receive-Name“ sendenden Objekten her;
oder {*clear*} : löscht die bisherige Verbindung;
1.Ausgang: {*bang*}
oder {*float*}
oder {*symbol*}
oder {*list*}
oder {*anything*} gibt die empfangene Message aus.

---

iem_sel_any : ermöglicht Messagebox mit Mehrfach-Inhalt

Funktionsweise:

Mittels einer {*add*}-Message wird ein Symbol-Feld mit Einträgen an der Stelle Index belegt. Zuvor stand an dieser Stelle das symbol „no_entry“. Das Symbol-Feld hat die Größe vom 1.Übergabe-Argument. Mit einer {*clear*}-Message löscht man die Einträge. Eine {*float*}-Message ruft die Einträge aus dem Symbol-Feld ab und gibt zuerst am 2. Ausgang das Symbol als {*anything*}-Message aus, und in einem 2. Schritt das Symbol mit vorangestelltem

{*set*}-Selektor am 1.Ausgang aus. Dieser 1.Ausgang soll mit einer Messagebox verbunden werden. Man hat also mittels dem float-Eingang wahlfreien Zugriff auf das Symbol-Feld.

Übergabe-Parameter:

1.Argumente: <*float*> maximale Anzahl an Symbol-Einträgen.

Eingänge und Ausgänge:

1.Eingang: {*float*} + Index : gibt am 2.Ausgang die {*anything*}-Message aus;
gibt am 1.Ausgang die {set} + <*symbol*>-Message aus;

oder {*add*} + <*float*> Index + <*symbol*> Eintrag : speichert den Symbol-Eintrag in den Symbol-Buffer an der Stelle Index;

{*clear*} löscht alle Einträge des Symbol-Buffers.

1.Ausgang: {*set*} + <*symbol*> Symbol-Eintrag;

2.Ausgang: {*Symbol-Eintrag*}-Message.

---

iem_send : Send-Objekt mit extra Eingang für Namen (Abbk. **iem_s**)

Funktionsweise:

iem_send funktioniert fast gleich wie das send-Objekt, es hat jedoch noch einen 2. Eingang, mittels dem man den Send-Namen verändern kann ({*set*} + <*symbol*> Name). Mit einer {*clear*}-Message am 2.Eingang unterbricht man die bisherige Verbindung.

Übergabe-Parameter:

1.Argument: <*symbol*> Send-Name.

Eingänge und Ausgänge:

1.Eingang: {*bang*}

oder {*float*}

oder {*symbol*}

oder {*list*}

oder {*anything*} versendet jegliche Message, falls ein empfangbereites Objekt zur Verfügung steht.

2.Eingang: {*set*} + <*symbol*> Receive-Name; stellt eine Verbindung zu unter „Send-Name“ empfangenden Objekten her;

oder {*clear*} : löscht die bisherige Verbindung;

Kein Ausgang:

---

pre_inlet : schickt eine Identifikations-Message jeder Message voraus

Funktionsweise:

pre_inlet dient der Erstellung eines Makro-Externals mit mehreren Eingängen, die {*anything*}-Messages verarbeiten. Die erste Übergabeargument-Zahl wird als ASCII-Nummer interpretiert und in ein <*symbol*> gewandelt, die zweite Übergabeargument-Zahl bleibt <*float*>; beide werden intern gespeichert. Das erste Argument dient als Selektor-Symbol, das bei Verwendung von ASCII-Zahlen von 1 bis 8 keinem Buchstaben-Äquivalent gleichkommt, das zweite Argument dient als Eingangs-ID (zB. 1 als erster Eingang, 2 als zweiter Eingang, usw.). Jede eingehende Message verursacht zuerst die Ausgabe einer ID-Message, und darauf folgend die Ausgabe der eigentlichen Message.

Übergabe-Parameter:

1.Argument: <*float*> ID-Selektor-Symbol als ASCII-Zahl;

2.Argument: <*float*> Eingangs-ID.

Eingänge und Ausgänge:

1.Eingang: {*bang*}
{*float*}
{*symbol*}
{*list*}
{*anything*} Jede Art von Message, die nach Ausgabe der ID-Message ausgegeben wird.

1.Ausgang: {*anything*} ID-Message-Ausgang + {*anything*} eingehender Message-Ausgang.

---

soundfile_info : liefert Header-Information einer wav-Datei

Funktionsweise:

Eine {read}-Message mit einem nachfolgenden <*symbol*>-Eintrag Dateiname bewirkt die Ausgabe einer Liste mit 7 Einträgen, falls die Datei erfolgreich geöffnet und interpretiert werden konnte. Die 7 Einträge sind:

- 1.Eintrag: <*float*> Samplerate in Hz (zB. 44100);
- 2.Eintrag: <*symbol*> Dateiname (zB. /sounds/ping.wav);
- 3.Eintrag: <*float*> Sound-Datenlänge in Samples (zB. 88200);
- 4.Eintrag: <*float*> Fileheader-Länge in Bytes (zB. 44);
- 5.Eintrag: <*float*> Anzahl der Kanäle (zB. 2);

- 6.Eintrag: <*float*> Einzel-Sample-Daten-Größe in Bytes (zB. 2);
- 7.Eintrag: <*symbol*> „l“ für little, „b“ für big endian;

litlle endian wird als natürliche Bit-Anordnungs-Reihenfolge für „wav“-Dateien auf Intel-Rechnern verwendet, big endian wird als natürliche Bit-Anordnungs-Reihenfolge für „aiff“-Dateien auf Macintosh-Rechnern verwendet.

Keine Übergabe-Parameter:

Eingänge und Ausgänge:

1.Eingang: {read} + <*symbol*> Pfad_plus_Dateiname.

1.Ausgang: {*list*} + 7 Einträge wie oben erklärt.

---

toggle_mess : ermöglicht Messagebox mit Mehrfach-Inhalt (Abbk. **tm**)

Funktionsweise:

Die N Übergabe-Argumente werden in einem zirkulären Element-Buffer gespeichert, der interne Zirkulär-Index zeigt anfangs auf den letzten Eintrag. Erreicht den Eingang nun eine <*float*>- oder {*anything*}-Message, springt der Zähler auf 0, der 3. Ausgang gibt diesen Wert aus; der 2. Ausgang gibt entweder einen <*float*>-Eintrag als {*float*}-Message, oder einen <*symbol*>-Eintrag als {*anything*}-Message aus. Der 1. Ausgang gibt den Eintrag als {*set*}-Message, gefolgt von einem <*float*>- oder <*symbol*>-Eintrag, aus. Weiters kann der Anfangszustand initialisiert werden mittels einer {set}-Message und einer nachfolgenden <*float*>-Indexzahl (Index von 0 bis N-1) und einer anschließenden {*bang*}-Message.

Übergabe-Parameter:

N.Argumente: <*float*>- oder <*symbol*>-Einträge, wird in den Zirkulär-Buffer kopiert.

Eingänge und Ausgänge:

1.Eingang: {*set*} + <*float*> Index : setzt den internen Zähler auf Index und verändert nur den 1.Ausgang;
{*bang*}: ändert auch 3. und 2. Ausgang;
{*anything*} oder {*float*} : inkrementiert internen Zähler (modulo) und ändert 3. , 2. und 1. Ausgang;

1.Ausgang: {*set*} + <*symbol*> oder <*float*> (je nach Eintrag) Richtung Messagebox;

2.Ausgang: {*anything*} oder {*float*} gibt bei <*symbol*>-Eintrag eine {*anything*}-Message aus, bei <*float*>-Eintrag eine {*float*}-Message aus;

3.Ausgang: {*float*} Index der am 2.Ausgang ausgegebenen {*anything*}-Message.

---

5 Literaturverzeichnis

[1] Miller Puckette, "*Pd Documentation Rel-0.37*", Dokumentation von Pd im Netz: "http://crca.ucsd.edu/~msp/Pd_documentation/"

[2] U. Tietze, Ch. Schenk, "*Kap 24 – Digitale Filter*" in: "*Halbleiter-Schaltungstechnik 9. Auflage*", Springer-Verlag 1991, pp. 834 – 841

[3] U. Tietze, Ch. Schenk, "*Kap 14 – Aktive Filter*" in: "*Halbleiter-Schaltungstechnik 9. Auflage*", Springer-Verlag 1991, pp. 391 – 414