Revista Vórtex | Vortex Music Journal | ISSN 2317–9937 | http://vortex.unespar.edu.br/
D.O.I.: https://doi.org/10.33871/23179937.2021.9.2.24

Elastic Rhythm in Signal-Synchronised Sequencing Objects for Pure Data

Edward Kelly

Synchroma Audio Engineering | United Kingdom

Abstract: this paper presents a family of objects for manipulating polyrhythmic sequences and isorhythmic relationships, in both the signal and event domains. These work together and are tightly synchronised to an audio phase signal, so that relative temporal relationships can be tempo-manipulated in a linear fashion. Many permutations of polyrhythmic sequences including incomplete tuplets, scrambled elements, interleaved tuplets and any complex franctional relation can be realised. Similarly, these many be driven with controllable isorhythmic generators derived from a single driver, so that sequences of different fractionally related lengths may be combined and synchronised. It is possible to use signals to drive audio playback that are directly generated, so that disparate sound files may be combined into sequences. A set of sequenced parameters are included to facilitate this process.

Keywords: polyrhythm, polymeter, sequence, isorhythm, synchronised

Received on July 1st, 2021. Approved on: November 1st, 2021. Available online on: 30/11/2021. Guest Editor: Dr. Alexandre Torres Porres.

While the use of multiple metro objects in Pure Data (Pd) is for many purposes completely adequate for creating complex rhythmic relationships between events (polyrhythms), another method can be derived from an audio signal (phasor~ object) by defining an event as the moment at which a signal crosses a threshold. There are benefits to this technique that become apparent when manipulating playback speed, in that the signal driver may be manipulated to generate playback drivers for tabread4~ objects, and hence when speeding up and slowing down the speed of the driver, it has the effect similar to speeding up and slowing down a vinyl record or magnetic tape. This is accomplished by the polyMath~ object[1]. Pitch relationships in a sequence manipulated this way are preserved, and signal drivers for sequenced events may be normalised (0 to 1) or not (the original phase value)[2]. It is relatively simple to switch to another model where pitch is irrelevant of the rhythmic timing of the sequence using the rhythmic events generated. Although the method for achieving this is event-based (rather than a direct signal) it is still derived from threshold-crossing of the signal input (from a phasor~ object in Pd) and so the timing of these events is as accurate as it can be within the signal/event paradigm of a Pure Data object, within about 1 ½ milliseconds[3].

Any sequence of fractional polyrhythm may be used, including the use of incomplete tuplets such as 4/5ths followed by 2/3rds, although such sequences could include completion elements later on i.e. (in this case) 1/5 and 1/3 (KELLY, 2005, p.88-98).

Such sequences may be scrambled within the object, and novel serial polyrhythmic elements organised into sequences may be generated on-the-fly. Various methods are used to navigate between sequences either instantaneously or in sequential order.

It is important to note the distinction and difference between polyrhythm (different numbers of regularly spaced events within a given, fixed time frame) and polymeter (different integer groups of events at a specific tempo). Hence we will be using fractional notation to describe polyrhythmic relationships (e.g. 6/12 is equal to 6 triplets of 8th notes (quavers)) and ratio notation

[1] <https://github.com/SynchromaAE/polyMath>

[2] A whole bar (of 4/4 time) is considered to be 1, so a crotchet (1/4 note) = 0.25, a quaver (1/8 note) is 0.125. A quintuplet crotchet (1/5 note is equal to 0.2 (1 divided by 5).

[3] 1.45125ms is the maximum delay at 44.1KHz sampling rate, block size of 64. 1.33333 at 48 Khz. Lower values can be achieved with higher sampling rates, and hence greater temporal accuracy.

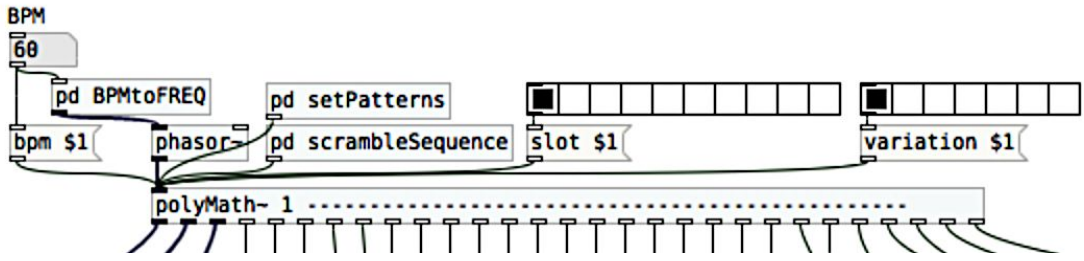
to describe polymetric relationships (e.g. 5:4 means there are two independent loops, one 5 beats long and another 4 beats long, so that by the time the first plays the 6th beat of its time signature, the other is already playing the second beat, and the loops re-synchronise at 20 beats (5x4).

At first, the notation appears to be counter to that which is normally used in music notation: a time signature of 5/4 against one of 4/4 produces the polymetric result and within that there may be triplets, quintuplets and so on – the mathematics of working out how to calculate a linear phase value from a polyrhythm are much more coherent with the fractional approach, and polymeter is best expressed in terms of ratios. Fundamentally there are two objects presented here: polyMath~ which deals with polyrhythmic sequences and playback, and isoWrap~ which deals with polymetric ratios. The two may be combined to create novel systems of rhythmic organisation, with extra features provided such that a number of parameters may be sequenced simultaneously.

1. polyMath~

The polyMath~ object has 1 inlet and 29 outlets, 3 of which are signal outlets. Many of the outlets represent features of the grouping system, some are for sequencing extra parameters coupled to rhythmic events, and some are designed for early instigation of event parameters so that (for example) the correct sample array will be played when the signal outlets are used for this purpose.

FIGURE 1 – The polyMath~ object

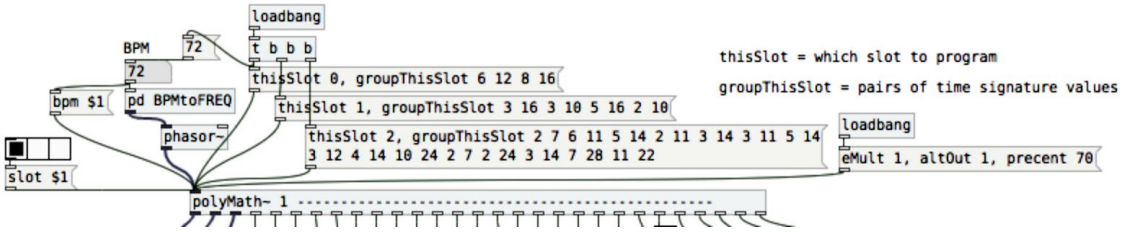


Rhythmic events are organised into groups, where each group is a continuous sequence of events with the same time denominator (rhythmic note value). Using the fractional notation system for polyrhythmic sequences, we can use this format to instigate rhythmic sequences, using thisSlot and groupThisSlot messages as in figure 2. Note that in the first instance we are programming complete

tuples (6/12, 8/16) and this is more like conventional music notation, whereas the second instance has broken tuplets (e.g. 3/10). Each sequence still adds up to 1.0 though, if each pair of values is taken as a fraction. 6/12 + 8/16 = 1.0. In the second sequence it is clear how complete sequences of polyrhythmic tuplets can be made from disparate groups of rhythmic elements i.e. we can have 3 quintuplets on their own (3/20) if we follow later on with two more quintuplets (making 5/20 which adds up to a single ¼ note) completing the tuple[4].

1.1. Slots and variations

FIGURE 2 – Programming Slots (Rhythmic Sequences)



Internally, each sequence is a slot, and each slot has a limited number of variations, with variation 0 as the original programmed sequence.

A sequence is programmed as a series of numerator / denominator pairs – time signatures, and each event of a time signature occupies a static location in memory[5]. So the first sequence above would program the rhythms to be 6 / 12th notes followed by 8 / 16th notes. It doesn't matter which order the slots are programmed in, since you specify the slot to program before you program it[6]. Each slot has a number of variations that are generated by scrambling a sequence.

There are currently two versions of this object, polyMath~ and polyMathLite~, and each has a limited number each of slots, variations, groups-per-slot and elements per slot or variation. Since

[4] This is one of a number of techniques for classifying polyrhythmic events. A more thorough evaluation of 20th-century rhythmic notation is found in Gardner Read (1978) pp. 22-83. Note that he uses ratios for any tuple that does not add up to 4 (e.g. 9:8 meaning 9 in the time of 8), but the fractional approach simply treats 9 / 8 as 9 events of 1 ÷ 9 .

[5] So a 3 / 16 pair will have three 1/16th elements (0.0625 phase value) but a 1 / 5.33333 pair will have one event of 3 / 16th notes in length (1 / (0.0625 * 3) = 5.33333 – but the phase value for this event will be 3 * 0.0625 = 0.1875.

[6] The reason to have to specify the slot to program is that the programming routines may be kept separate from the internal phase generator and sequencers, só that the object may be programmed on-the-fly – while the object is running.

the buffers are a fixed size in this object, they use a predictable amount of memory, and the polyMath~ object can use a significant of memory if a number of the objects are used, the other configuration of the object included for use in lower memory applications. These objects were always intended for use in both desktop and mobile applications, so a knowledge of memory usage (and fixed memory usage) may be important in such scenarios. If the user is familiar with programming and building objects from source code there are four global variables in the source code at the top of the .c files that alter the first four of these limitations. Extra sequence pairs are fixed for each object[7].

Currently the configurations are as follows:

TABLE 1 – Object Configuration

	polyMath~	**polyMathLite~**
Slots	128	64
Elements-per-slot/variation	2048	512
Groups-per-slot/variation	256	128
Variations per-slot	6	2
Extra sequence pairs	8 x 2	4 x 2

Within each slot or variation, a maximum number of groups is allowed. The groups themselves are collections of elements with identical fractional denominators, so 4 / 12ths would be a group but 3 / 16ths + 2 / 20ths would be two groups. Note that incomplete tuplets (4 /12ths meaning 4 x 8th-note triplets) are acceptable. Within a complete phase cycle, they can be completed by the same rhythmic type, or related (binary subdivision) later on in the sequence: 4/12 + 4/8 + 4/24 = 1. Hence the time signatures are used as fractions in order to calculate the phase of each event.

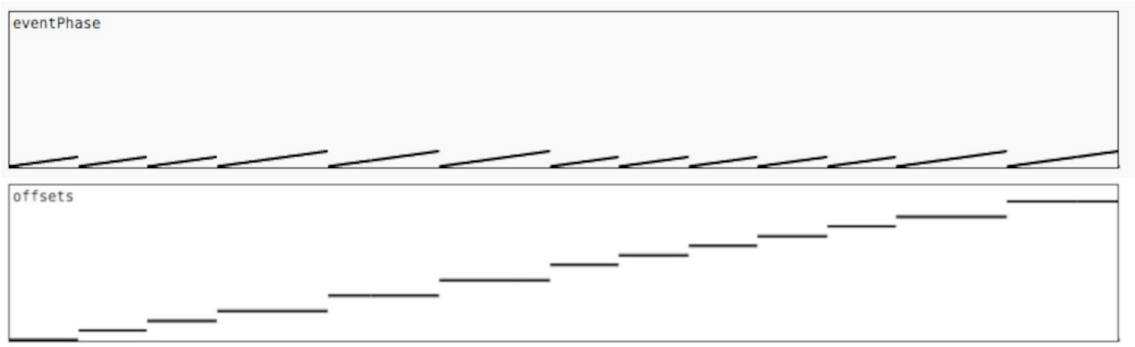
2. Event Ramps

Each element, as it is played, generates a number of data points as Pd events, but also signals that may be used to drive the playback of one audio file, or multiple audio files where sequence

[7] When this object was conceived, mobile devices had much less memory than they do now. The polyMathLite~ object is a legacy from then, but there seems little incentive to continue to support this.

pairs are used. The simplest form of this is that we use a set of ramps to play back one (or more) audio files in sequence[8] and the first signal outlet will generate in it's most basic settings a graph such as that shown in figure 4, which has a rhythmic sequence of 3 / 16 + 3 / 10 + 5 / 16 + 2 / 10.

FIGURE 3 – Raw Ramp Output, and offsets from Signal Outlet 1, with a Sequence of 3/16, 3/10, 5/16, 2/10, not normalised (raw phase value).



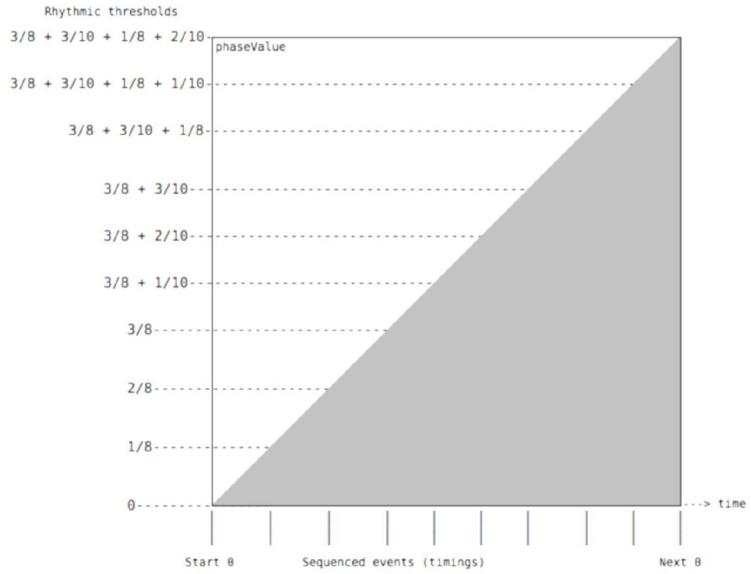
If we were to play a single audio file as it was originally recorded using this, the third signal outlet would give the offset, and added to the amplitude of each ramp segment a smooth line from 0 to 1 would result[9].

The polyMath~ object stores the phase value of each event, and the cumulative phase value crosses a threshold at the start of each event as shown in figure 4.

[8] Sequence pairs may be used to set an audio file. Later is shown a method for generating an early sequence pair and using alternate signal paths will be preferable.

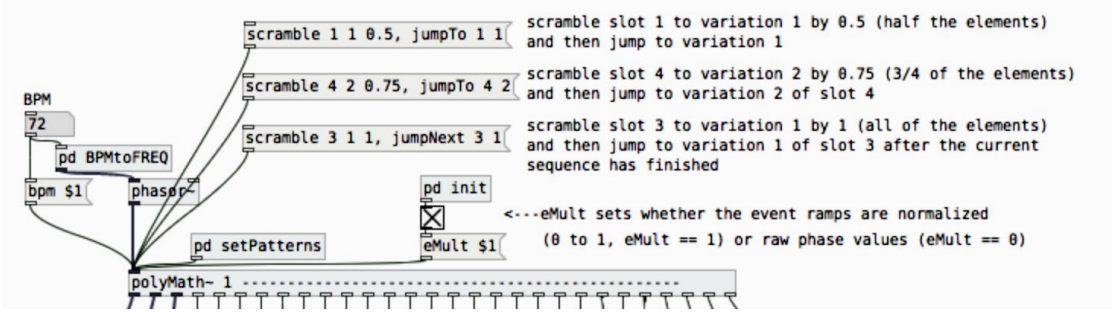
[9] Note that we are only considering a single "page" of sequence, i.e. a single phase signal from 0 to 1. The object supports sequences that span more than one cycle or page.

FIGURE 4 – The relationship between the phase input, and rhythmic thresholds which when crossed trigger the next event.



3. Scrambled Variations and Alternating Play Heads

FIGURE 5 – The **scramble** message creates a randomly re-ordered variation of the rhythms and offsets. This also shows **jumpTo** and **jumpNext** commands for switching between slots / variations. Sending the object [**eMult 1(** causes the signal ramps to output normalized ramps (0-1) for each event.

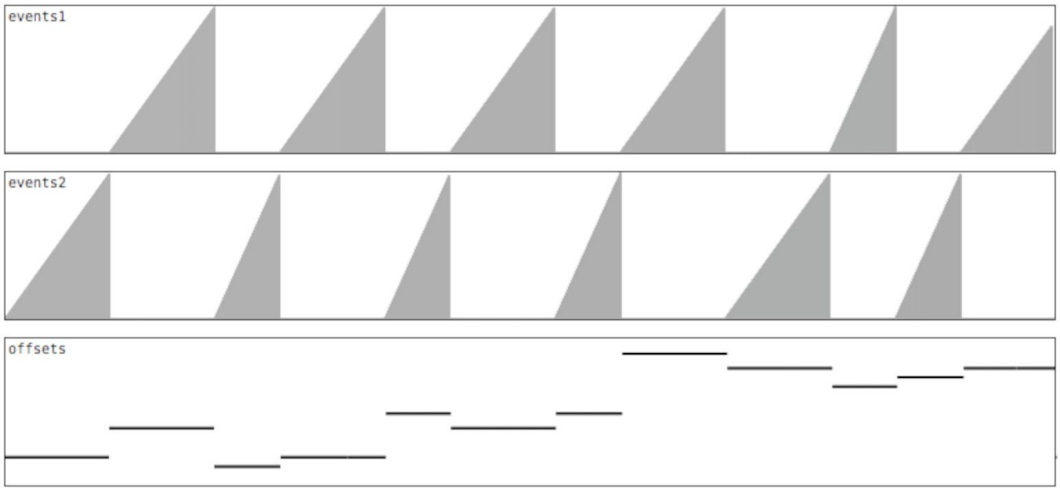


In the case of playing back a single soundfile using the scheme developed above, there is no reason to do this. The result of adding the ramps to the offsets is a signal ramp that goes seamlessly from 0 to 1, so it is the same as the phasor~ output that feeds the object, multiplied by the sample array length (and other factors) to play an array at any speed.

As is shown later, this information can be used to drive various forms of concatenative (or other) synthesis using normalized ramps. Within the context of an audio loop however (or any other event-based use of the sequencer) a scramble message can be used to create a variation of the sequence.

The variation in figure 6 shows three new aspects. The first is that the same ramps and offsets are aligned, but have been re-ordered. The second is that the ramps are now normalized from 0 to 1, and the third is that there are now alternate signal ramp outputs from the first and second outlets of the object. If we didn't normalize the ramps like we did here, we could read a scrambled variation of a single audio file by adding all three signals together, and it would have the same rhythmic values of the original sequence but in a different order. But the audio ramps alternating and normalized could be used to read *any* set of audio files, or sections of audio in any set of files, and re-order this sequence (or even use the original sequence to play a new set of files). If so, then it is necessary to find a way of setting audio tables from within the object, perhaps even setting the length of a section of an audio file per-section, and even generating any arbitrary parameters for this sequence to occur smoothly.

FIGURE 6 – A sequence that has been scrambled to generate a variation, with alternating "play head" signal outlets and normalized event ramps (filled in grey for clarity) and offsets from the original sequence.



4. Sequence Parameters and Early Messages

Along with the event sequence, there are a set of sequenced parameter pairs that are output as the sequence unfolds. They are organised into pairs so that they may be routed to diverse parameters downstream of the object in a Pd patch. These may be used to set the alternate event ramps to each play a different audio file, or sections of audio in different files, as well as controlling other parameters of signal processing and generation.

FIGURE 7 – Sequence parameter pairs.

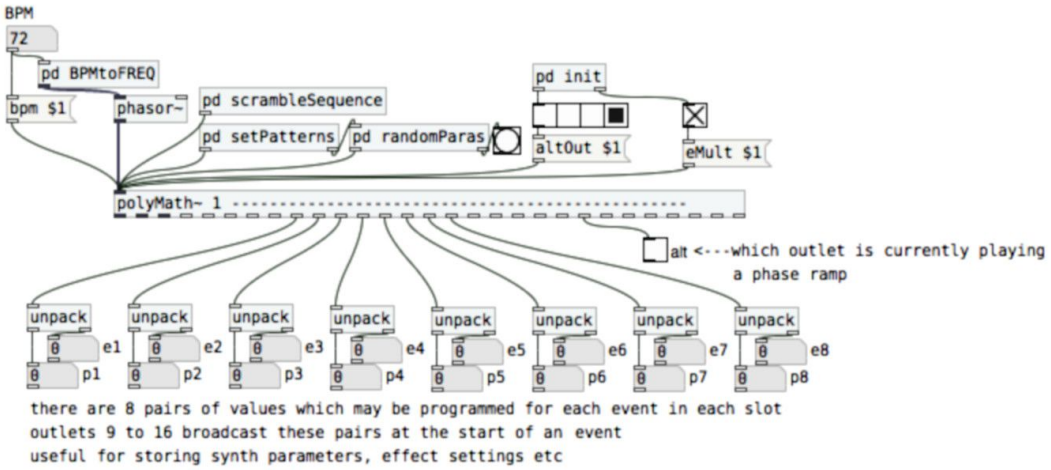
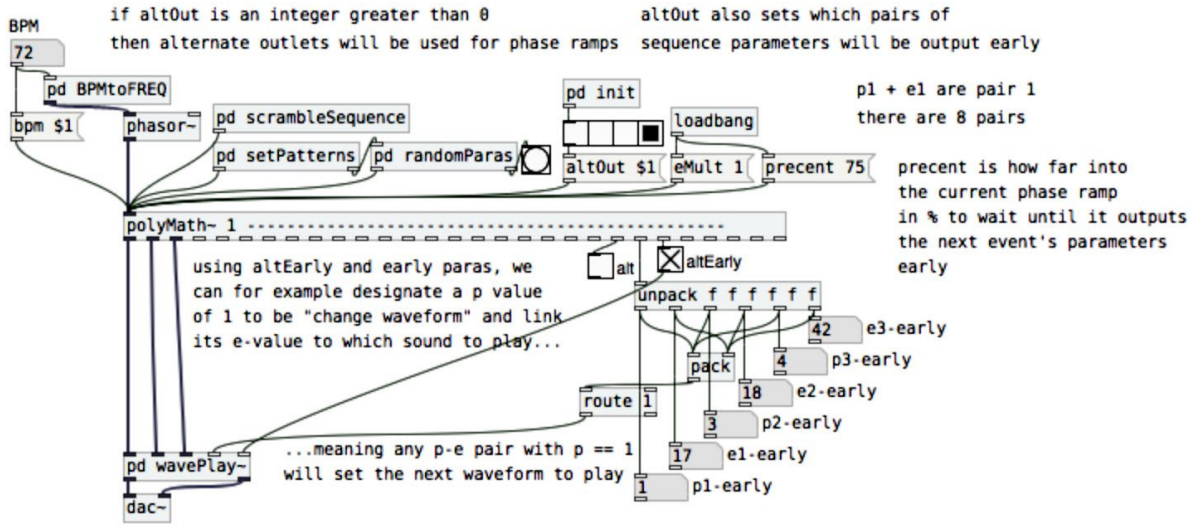


FIGURE 8 – The precent parameter is the percentage of the current event phase to wait before triggering early, parameters for the next event. An altEarly toggle indicates which will be the next phase ramp outlet~ (1 or 2) and then a number of pairs of parameters for the next event in the sequence come from outlet 23



Using alternate event-playback ramps would be challenging if the trigger to change an audio table (such as a message to the tabread4~ object) was generated at the end of the DSP block[10] (usually 64 samples). Clicks would result, as the phase ramp would already have begun by the time the table is changed (most of the time).

[10] Digital Signal Processing block size – by default in Pd this is 64 samples, and events generated within this time are transmitted at the end of the block, whereas a change in the audio signal may happen at any time within the block.

If a sequence parameter held information as to which sample table was *about* to be played by the next ramp to start, it could be initiated before the ramp starts. Since we are concerned with pre-programmed sequences, this is possible and once again is triggered by a signal crossing a threshold. This time, it is the current event ramp signal which generates the early parameter output – the signal of the current ramp crosses a point, e.g. 75% of the normalised ramp, and the early parameter output can be used to determine what the *next* playback instance (soundfile, wavetable etc) will be. The *altEarly* toggle happens before this, and so the early messages can be sent to different paths using spigots, to instruct one of two playback objects which file to play (for example), while the alternate ramp is still going (it is the signal outlet presently ramping that initiates the early signal to generate this event) and so the signal to change sample tables on one ramp is sent by passing a threshold (termed the "precent" value) on the other ramp. Once again, a threshold-based system based on a signal, can be used to sequence array-based sequences with an audio resolution equivalent to vinyl[11], but with the flexibility of sample-based concatenative synthesis. The number of sequence parameters that are output in this way is controlled by the number sent to the altOut message, and so there is potential for initialization of several parameters on each channel, ready before playback commences.

5. All of the Information at Once

If all of this complex rhythmic information is held in a monolithic object, and there is only one method to retrieve it (i.e. waveform playback using sample tables), it could be functional but it also points towards a didactic approach to its use. Considering the amount of information about each sequence stored within the object, it therefore makes sense for the object to make all of this information available to be specific, each sequence is stored in a variety of forms specific to the structural information required to process and interpret it.

[11] This has not been tested where the event – the "precent" of the event is less than the blocksize. Other methods are available for dividing up audio files and sequences in millisecond scales.

FIGURE 9 – polyMath~ outputs information about the current sequence, and each event, in many forms

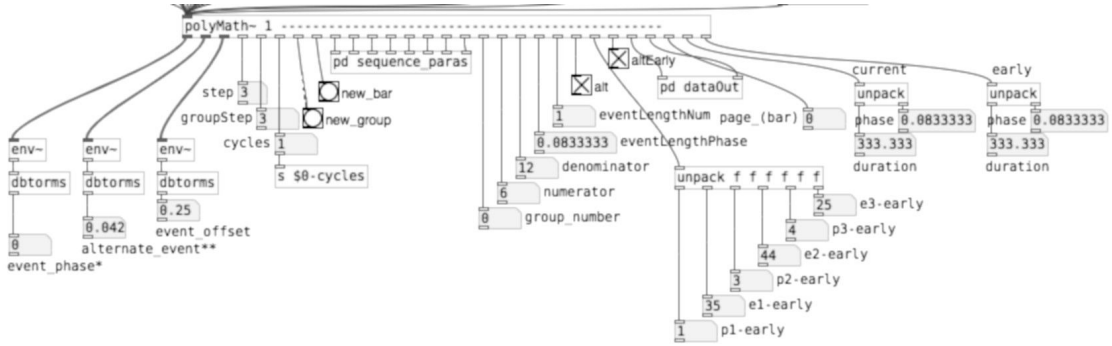


TABLE 2 – Non-signal Outlets

Outlet	Name	Description
4	step	Which sequential event is this?
5	groupStep	Which event is this in the current group (e.g. 2 out of 5, starting at 0)
6	cycles	How many cycles of the phasor~ does this sequence take?
7	new group	Bang when a new group begins
8	new bar	Bang when a new bar (or cycle, or page[12]) begins
9-16	sequence parameters	8 pairs of floating-point numbers, to route to any parameter
17	group number	Which group is this?
18	numerator	How many rhythmic intervals (e.g. 6 if the group is 6/8)
19	denominator	The rhythmic unit (e.g. 12 if the group is 5/12)
20	eventLengthPhase	Phase value (between 0 and 1) for the current event
21	eventLengthNum	How many multiples of the denominator
22	alt	Which signal ramp outlet is currently firing
23	Early sequence parameters	What happens next?
24	altEarly	Which event ramp signal is the next to fire
25	DataOut	When outputting data, it appears here[13]
26	DataType	Which type of data are output
27	Page	Which cycle (page, bar etc) are we on? (polyMath~ patterns can span multiple cycles of phasor~ output)
28	durFirst	Duration / phase pair of current event
29	durAlt	Duration / phase pair of next event (early)

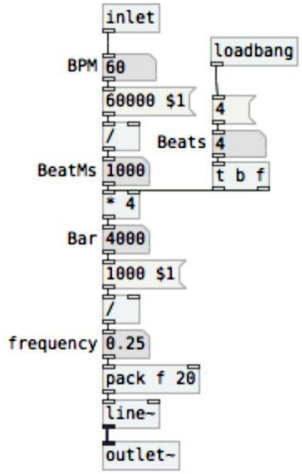
[12] Bar, cycle or page? – this terminology has yet to be finalised. They are all relevant, but cycle is more relevant since bar implies 4/4 time signatures due to the way phase signals are divided up. The idea of a "page" is useful since it does not imply completion.

[13] There is a yet un-documented feature to dump the data. Work needs to be carried out to develop the Pd infrastructure around this.

Taking a look at the full output from the object, it is clear that there are many approaches to using this rhythmic coherence of phase values, numeric musical representations of rhythm, and all of the data that resides within the object as it unfolds as a sequence. It may of course be used in a simpler way, perhaps by triggering events from the change of the step outlet (outlet 4), or the raw phase values could be used to set fast (or slow) loop points within array playback, so the purpose of the 5 year-development process has not been to provide a monolithic way of controlling sequences of recorded audio samples. It has been always in the service of complex rhythm, allowing multiple uses.

Without re-writing the object, it already has many approaches to rhythm in its 27 event outlets. Note that there are statistics as to how the rhythmic elements are organised within the sequence. The denominator (e.g. 8th notes, or perhaps 17th notes) is shown. Every event message is triggered before the "step" outlet so if all is needed is a polyrhythmic modulating rhythmic pulse then this is fine, but there are numerous ways in which the analytical data (and sequence data) from the other outlets could be used. Regardless of variations and scrambling permutations, this object allows for the storage of complex polyrhythmic patterns that will be tightly synchronised to audio-rate events.

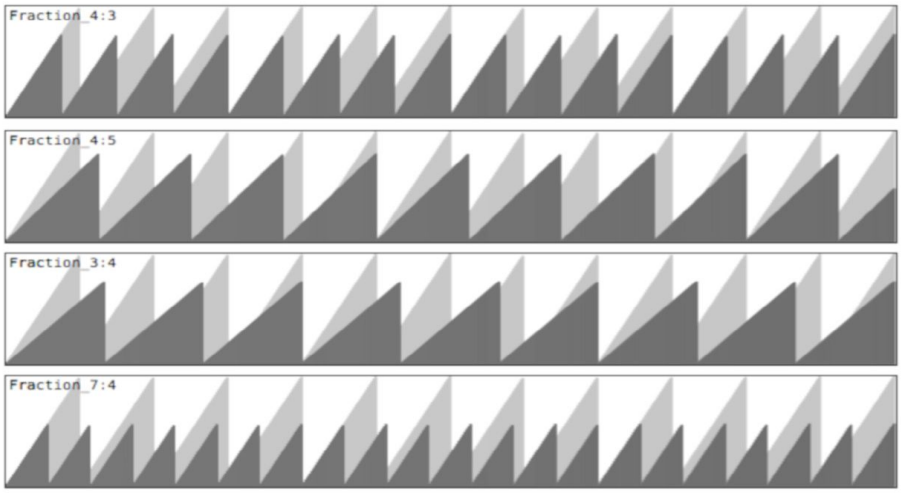
FIGURE 10 – Basic conversion of musical time to frequency, for phasor~ input to polyMath~[14]



[14] Note that the bpm message only applies to the outputs of outlets 28 and 29, where duration is stated along with phase.

6. Polymeter and Macro-Structure with isoWrap~

FIGURE 11 – Some potential phase relationships between raw phasor~ input (light grey) and isoWrap~ output (dark grey) at different ratios.



While it is entirely possible to create any sequence of polyrhythm and polymeter within a single sequence using polyMath~, to generate sequences of different lengths and synchronise them using more than one instance of polyMath~ requires more attention to the phasor~ signal that is used to drive the object.

Consider the situation where two sequences, one of five beats length, and another of four, are made to play synchronously in one of the following scenarios:

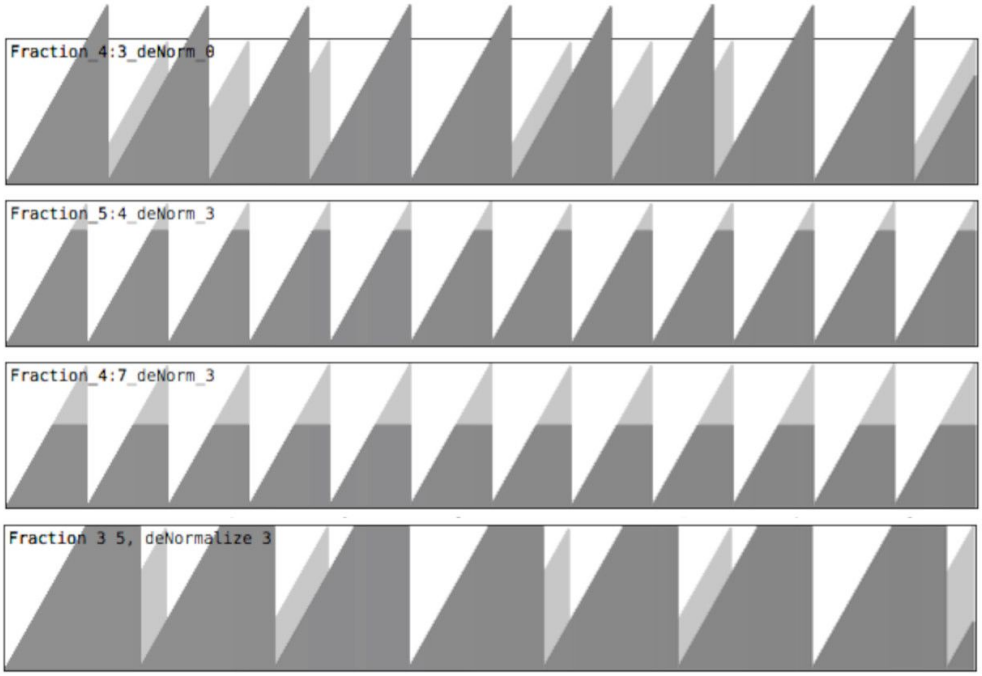
1. The four-beat rhythm is slowed down so that it takes as much time as the five-beat rhythm
2. The shortest rhythm plays to the end and then stops
3. The shortest rhythm is played synchronously with the longest, but looping at the end of each bar causing an isorhythmic effect

isoWrap~ takes the phasor~ signal and applies ratio-based temporal wrapping – it generates sympathetic phasor~ signals based on ratios of bar lengths and normalization values, so that two polyMath~ objects can be operating on different but related rhythmic bases without falling into or out of synchronisation. There are also several normalization functions so that the signals driving the polyMath~ objects can adopt different pulse-relationships on-the-fly, slip in-and-out of

synchronous operation, and translate the rhythmic relationship to a different fractional relationship.

Some examples of phase relationships can be seen below. It is worth noting that polyMath~ sequences can go on for multiple "bars" - i.e. 0-to-1 phase ramps from phasor~, and the reset from a high value to a low one tells the polyMath~ object when to switch to a new "bar" or page[15]. With isoWrap~ the order in which the fractional relationships between the polyMath~ objects is permutable in real time, and their polymetric, indeed isorhythmic relationships may be developed. Since there are many relationships between the phase input and the isoWrap~ signal, nested relationships between polyrhythmic sequences and transformations through isoWrap~ may be realised[16].

FIGURE 12 – Different methods (deNormalize) for clipping, normalizing and scaling the input result in various configurations. There are many more possibilities to be discovered.



[15] Since the notion of a bar is arbitrary here, and not necessarily the entire passage of a phase signal from 0 to 1, perhaps we should just use a literary metaphor instead, as in some other branches of computer science, the notion of a "page" of data. The isoWrap~ object presents the possibility that not all patterns occupy a complete "page", which in any case is set by default to a single bar of 4/4 (or a phase value of 4 ÷ 4 = 1)

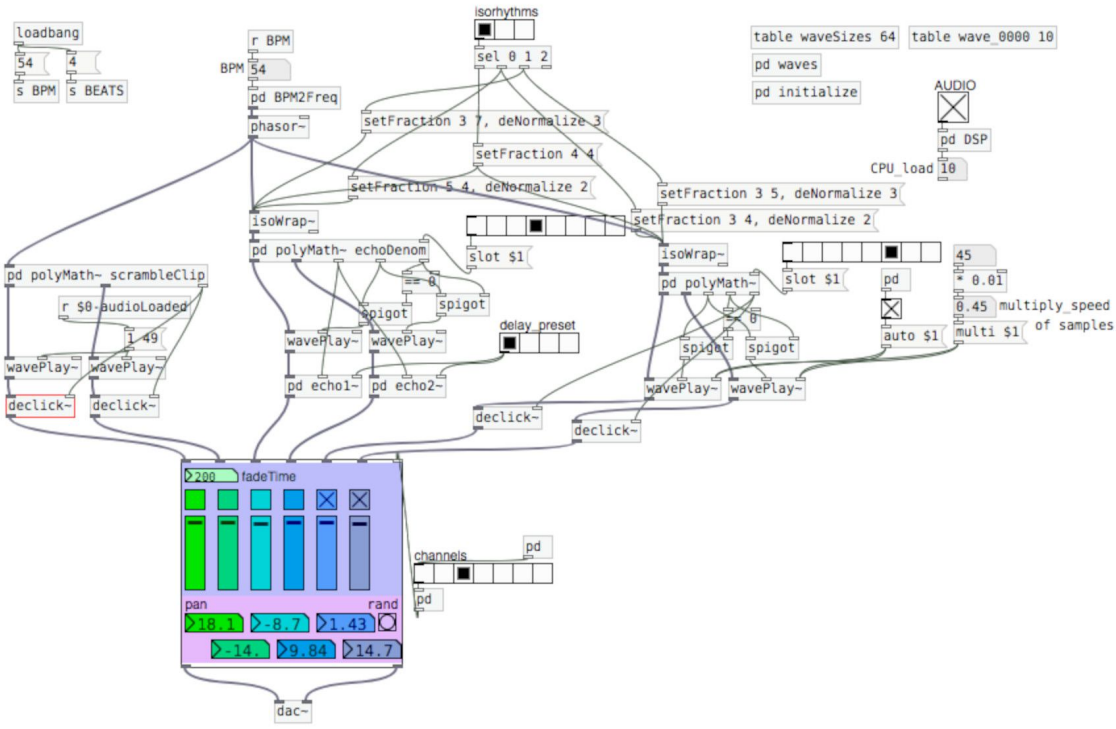
[16] And since isoWrap~ can generate any ratio between driver signals, there is the possibility for larger beat sizes for the driver (figure 10 – perhaps 6, or 9 beats) and the isoWrap~ signal can be reduced or extended according to **setFraction** and **deNormalize** messages sent to it.

These relationships are controlled within sections of the process prior to the injection into any polyMath~ object, and so macro- polymetric and polyrhythmic relationships are relatively simple to establish using this tool kit of objects.

It is important to realise that these signals are the drivers for polyMath~ objects, and within each of these ramps these signals can regulate polymetric or polyrhythmic structures at a microscopic scale, since if each phase signal controls a polyMath~ object, there is a flexible approach to rhythmic relationships on both the macro- and micro-scopic scale.

An example of the isoWrap~ objects combined with polyMath~ objects with a simple sample player is shown in figure 13. There are a number of assumptions – e.g. the sample arrays will be indicated by a number, and the sample loading system with its automatic naming system: from wave_0001 to wave_9999 makes this easily possible using the "early" sequenced parameters.

FIGURE 13 – Example sample player patch with isoWrap~ and polyMath~ objects.



This is the basic demo/TrioDemo.pd exposition in the source code. More implementations are to follow.

Conclusion

Polyrhythm and polymeter are concepts that have had some traction in modern musical practise, particularly in the period after World War 2. Any one example of these techniques in a piece of work is just a part of a network of rhythmic concepts – sets of conceptual ideas and techniques that encompass everything we do already know and use, but present a landscape where available tools become the means by which new musical territories can be explored. While in the 1950s onwards, instrumental performers would have to practise quite extreme exercises in order to perform the works of Boulez and Stockhausen for example, modern technology allows a freedom with principles of modern rhythmic expression in digital form.

The polyMath~ set of objects is a toolkit for exploring these realms. It is presently experimental and features are developing, but it offers a way to use rhythm as a continuum of events, rather than a singular unassailable clock.

There are many ways in which this piece of software can be used, and there are also many ways in which it could be modified to form a family of objects that are more agile in aims toward the goals for which they are intended. This is a project to open rhythmic flexibility as a practical construct to be modulated in expressive, musical ways, as a tool not just for composition but also for on-the-fly improvisation with complex rhythmic phenomena.

ACKNOWLEDGMENT

Much of the code for this was written during sessions of the Lambeth Music Service while my son had drum lessons. https://www.lambethmusic.co.uk/lambeth-sounds-partners.html. I am indebted to Simon Limbrick, who is always ready to engage with complex material. https://www.marimbo.com. Joao Pais has a wonderful approach to polyrhythmic metronomy, made for live performances - https://jmmmp.github.io/clicktracker/. The graphics in this piece were made using the Purr Data distribution by Jonathan Wilkes. https://agraef.github.io/purr-data/

REFERENCES

KELLY, Edward. *Time in Music: Strategies for Engagement*, PhD thesis, University of East Anglia, July 2005, pp88-98. http://sharktracks.co.uk/ed/epk_thesis.pdf

KRAMER, Jonathan D. *The Time of Music, New Meanings, New Temporalities, New Listening Strategies*. Macmillan,1988.

MANOURY, Phillipe. The Arrow of Time, *Contemporary Music Review*, 1:1 pp. 131-145. 1984

PUCKETTE, Miller. Pure Data, In: Proceedings of *International Computer Music Conference*, Ed. 20, San Francisco, 1996

READ, Gardner. *Modern Rhythmic Notation*, Bloomington: Indiana University Press, 1978, pp. 22 onwards.

STOCKHAUSEN, Karlheinz. ...How Time Passes... In: *Die Reihe,* 3. Massachusetts: Theodore Presser co., English language version, translated by Cornelius Cardew, 1959

ABOUT THE AUTHOR

Edward Kelly studied Electronic Music for his first degree, and developed a passion for complex rhythmic phenomena. His subsequent PhD explored complex harmonic and rhythmic systems, poly rhythms and modal serialist harmony. Ideas which he has continued to pursue through his independent research in computer music. He has explored diverse aspects of synthesis (wave manipulation) and sequencing, but has also developed a hybrid system for the manipulation of broken polyrhythmic groups and incomplete tuplets. ORCID: https://orcid.org/0000-0003-1921-7325. E-mail: synchroma@gmail.com

ANNEX

The full suite of polyMath objects can be downloaded from:

https://github.com/SynchromaAE/polyMath