Joachim Heintz

Views on Sine Waves

Online Workshop at TIEMF 2021

Part III:
Problems and Possibilities

Please **download the example files now** at:
www.joachimheintz.de/workshops/tehran_2021
-> examples_III.zip

Unpack the examples_III.zip in your working directory.
(I suggest to create a new folder.)

Looking back

What we did so far:
- Created percussive sines
- Learned about envelopes
- How to apply volume and pitch
- How to work with arrays to
- Create different spectra

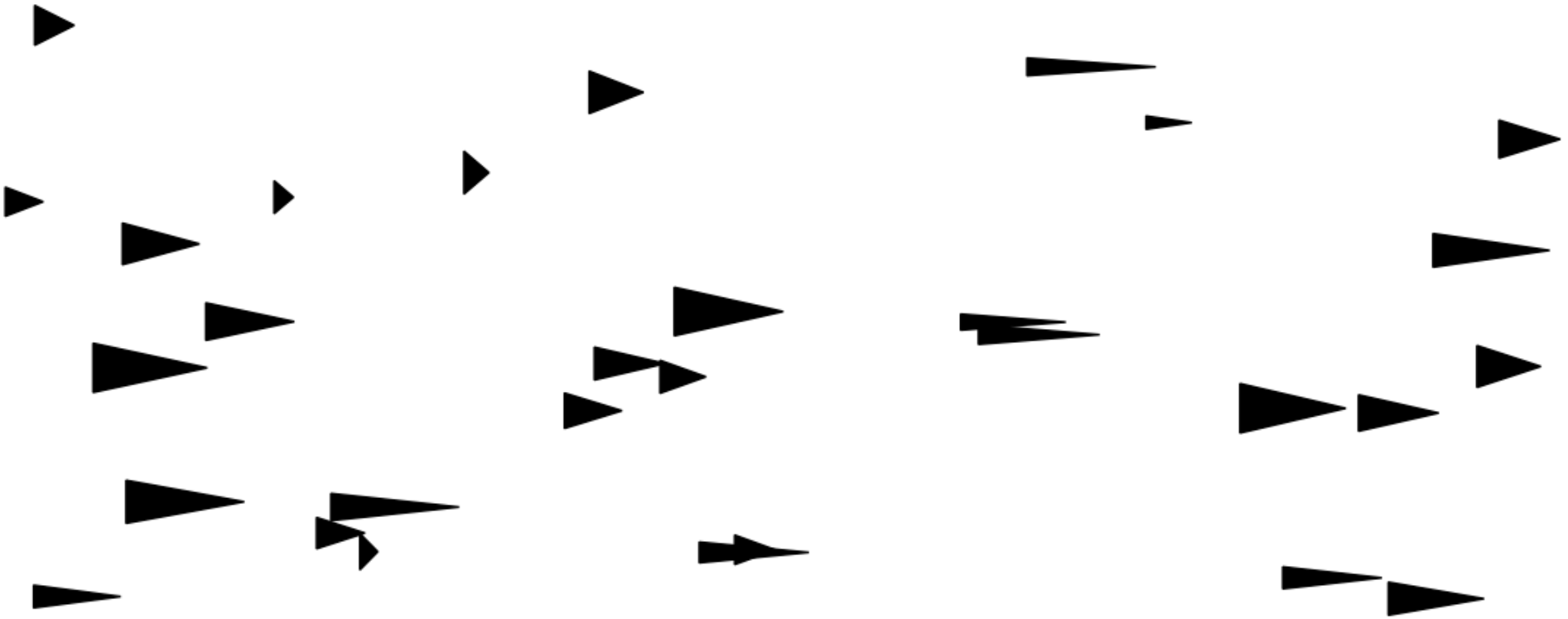
And now?

Place for music and composition but:

What is the idea behind it?

In which way to go on with the Cloud?

(Why actually a cloud?)



The problem

The question, at this point, can only be answered by any of you, individually.

It needs time to get to an idea. (Time and dedication.) And only if the idea is there, programming can work in a serious way (beyond "anything goes").

Not a solution but

I will focus here on rhythm and explore some possibilities how to work on rhythmic structures using arrays.

(You can work in a similar way with arrays for pitches.)

1 Use Csound's internal clock

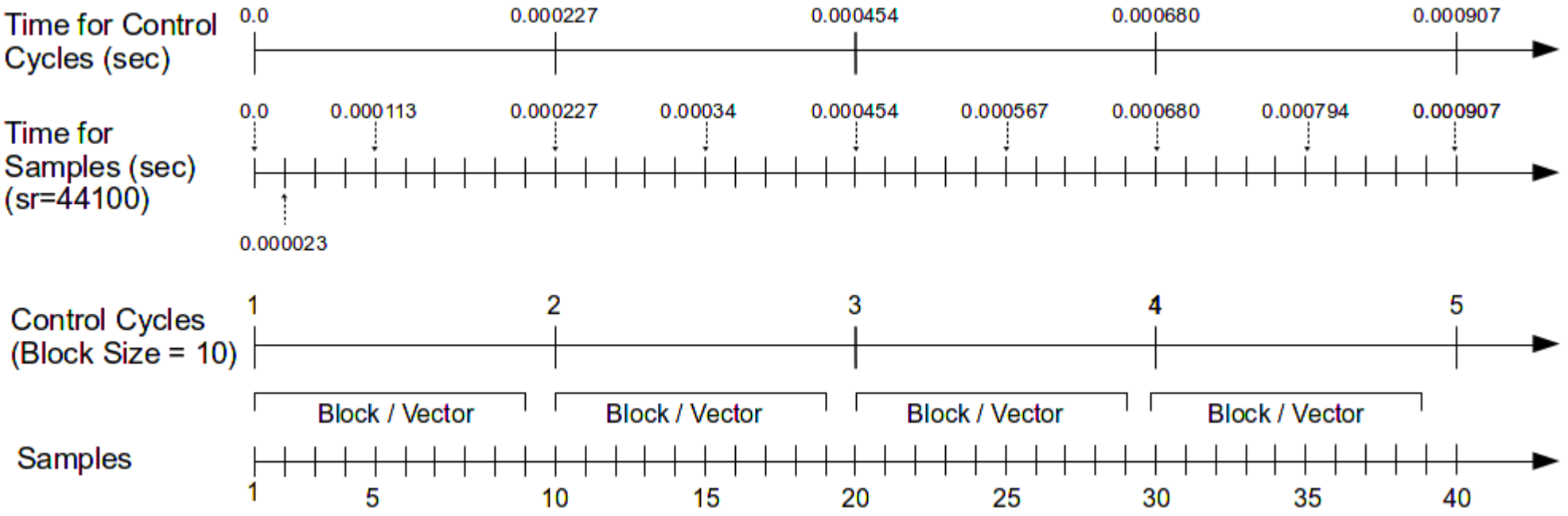
```
sr = 44100
ksmps = 10
nchnls = 2
0dbfs = 1
```

number of samples per second

number of samples per **control cycle**

Csound's internal clock for

```
sr = 44100
ksmps = 10
```



(find the error in this figure ...)

Enjoy some calculations

Symbol	Meaning	Unit	Example
`sr`	number of samples per second	Hertz (Hz)	`44100`
`1/sr`	time for one sample	Seconds (sec)	`1/44100 = 0.000022675... sec`
`ksmps`	number of samples in a control period	Number (always integer)	`10` (Csound old default) `32` (better because power-of-two)

`sr/ksmps`	number of control periods in one second	Number	for ksmps=10:
			`44100/10=4410`
			for ksmps=32:
			`44100/32=1378.125`

Good news

`sr/ksmps` is known as **`kr`** in Csound
so we can write:

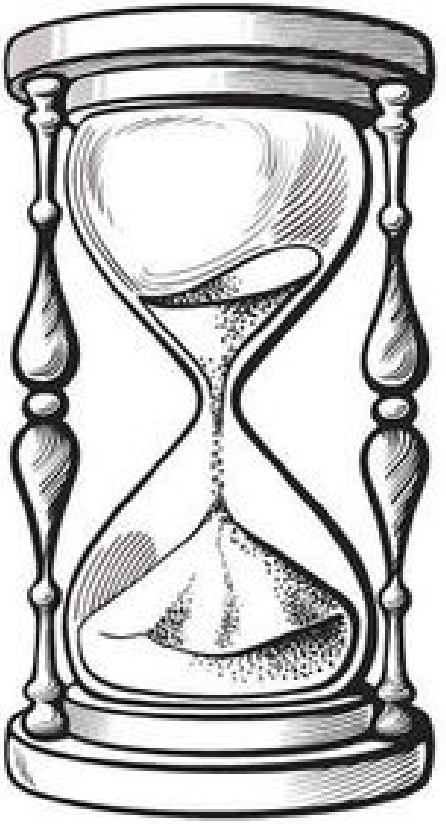
Symbol	Meaning	Unit	Example

`kr` (= `sr/ksmps`)	number of control periods in one second	Number	for ksmps=10:
			`44100/10=4410`
			for ksmps=32:
			`44100/32=1378.125`

`1/kr`	**time for one control period**	Seconds (sec)	for ksmps=10:
			`10/44100 = 0.00022675...`
			for ksmps=32:
			`32/44100 = 0.00072562...`

In other words

The `1/kr` time is always ticking in any running Csound instrument and we can use it like the sand grains in an hour glass



Finally start coding

III_01.csd — measure time by subtracting 1/kr
(the hour glass mechanism)

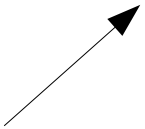
2 Repeat the process (create your own metronom ...)

III_02.csd — turn the hour glass once all time units have been fallen down ...

3 Read an array as rhythmic sequence

III_03.csd — stepping through an array

DATA	1	1	3/2	1/2	1	1/2	1/2
INDEX	0	1	2	3	4	5	6



start with index 0 ...

+1 +1 +1 +1 +1 +1

... then continue ...

Typical Csound code for incrementing the array index:

```
kIndx += 1
```

"set the next value of `kIndx` to the current value plus one"

But what to do here?

DATA	1	1	3/2	1/2	1	1/2	1/2
INDEX	0	1	2	3	4	5	6

start with index 0 ...

+1 =1 +1 =2 +1 =3 +1 =4 +1 =5 +1 =6 ... then continue ...

+1 =7 ... and ?!?

Usually we want to wrap around

DATA	1	1	3/2	1/2	1	1/2	1/2
INDEX	0	1	2	3	4	5	6

+1 +1 +1 +1 +1 +1 +1

=1 =2 =3 =4 =5 =6 =7

start with index 0 ...

... then continue ...

... and ...

...start with index 0 again

Possible code to do it

DATA	1	1	3/2	1/2	1	1/2	1/2
INDEX	0	1	2	3	4	5	6

+1 +1 +1 +1 +1 +1 +1

=1 =2 =3 =4 =5 =6 =7

start with index 0 ...

... then continue ...

... and ...

...start with index 0 again

```
if kIndx == 7 then
  kIndx = 0
endif
```

Or more elegant using the modulo

Length of the Array (7 Elements)
LENARRAY

DATA	1	1	3/2	1/2	1	1/2	1/2	**NO!**	**NO!**	**NO!**
INDEX	0	1	2	3	4	5	6	7	8	9
MODULO as INDEX % LENARRAY	0	1	2	3	4	5	6	0	1	2

Csound Code:

```
kArray fillarray 1, 1, 3/2, 1/2, 1, 1/2, 1/2
kIndx init 0
if [time is ok] then
 kIndx = (kIndx+1) % lenarray(kArray)
endif
```

4 More comfort ...

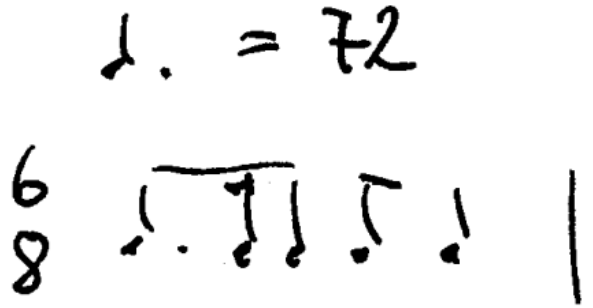
III_04.csd — setting an own time unit
III_05.csd — time unit can vary over time **DIY**

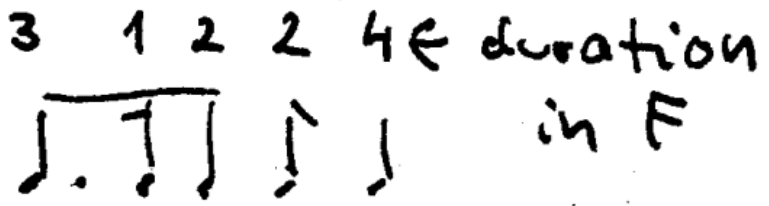
DIY = Do It Yourself!

5 Modifying a rhythmic sequence

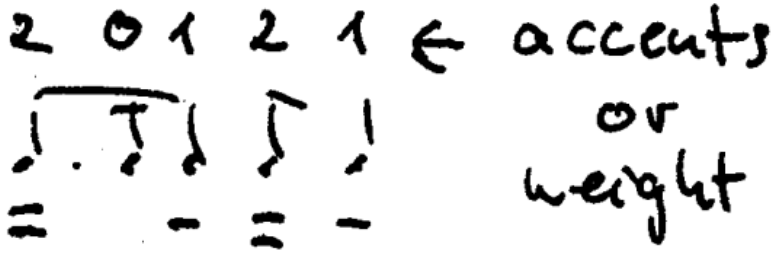
III_06.csd — simple looping a real (persian) rhythm

DIY

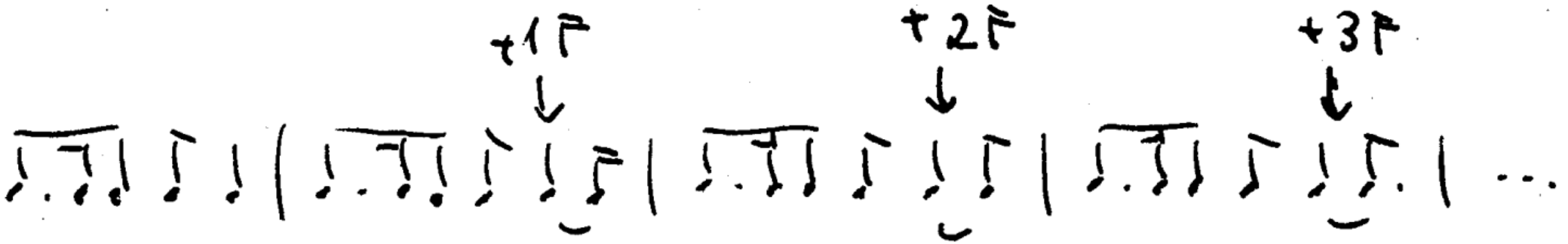




III_07.csd — applying accents



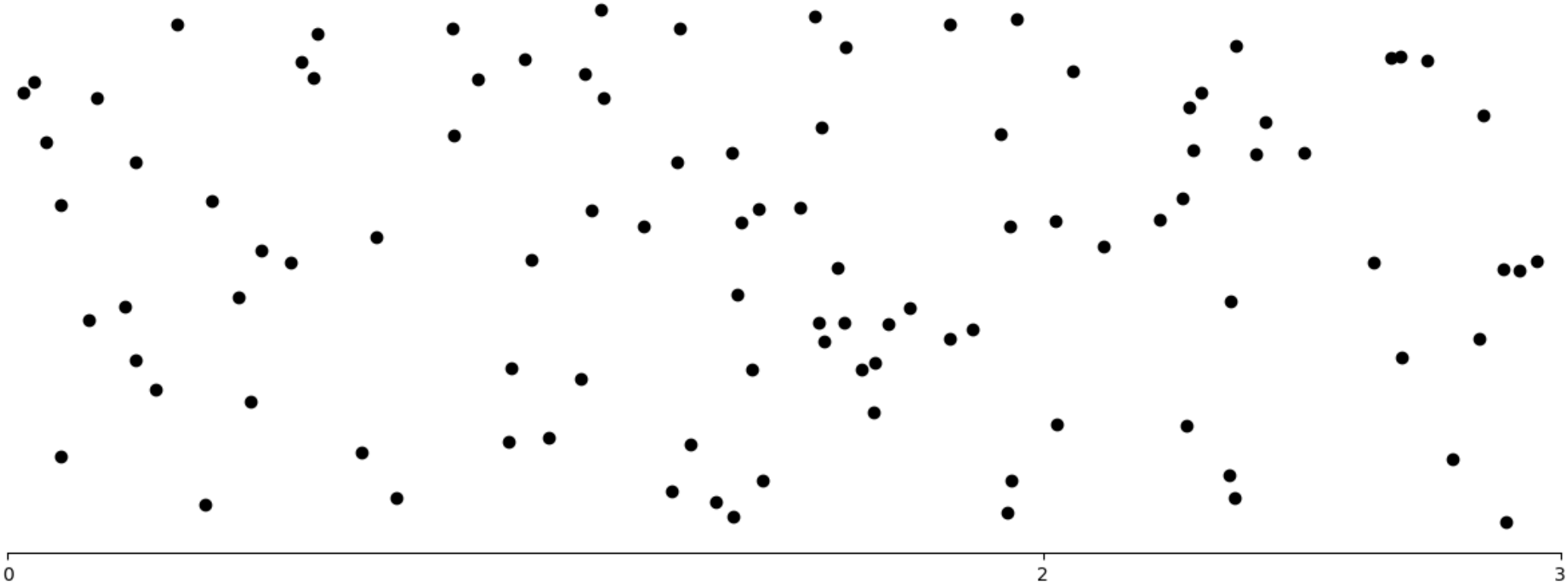
III_08.csd — change one value again and again



III_09.csd — change always a different element **DIY**

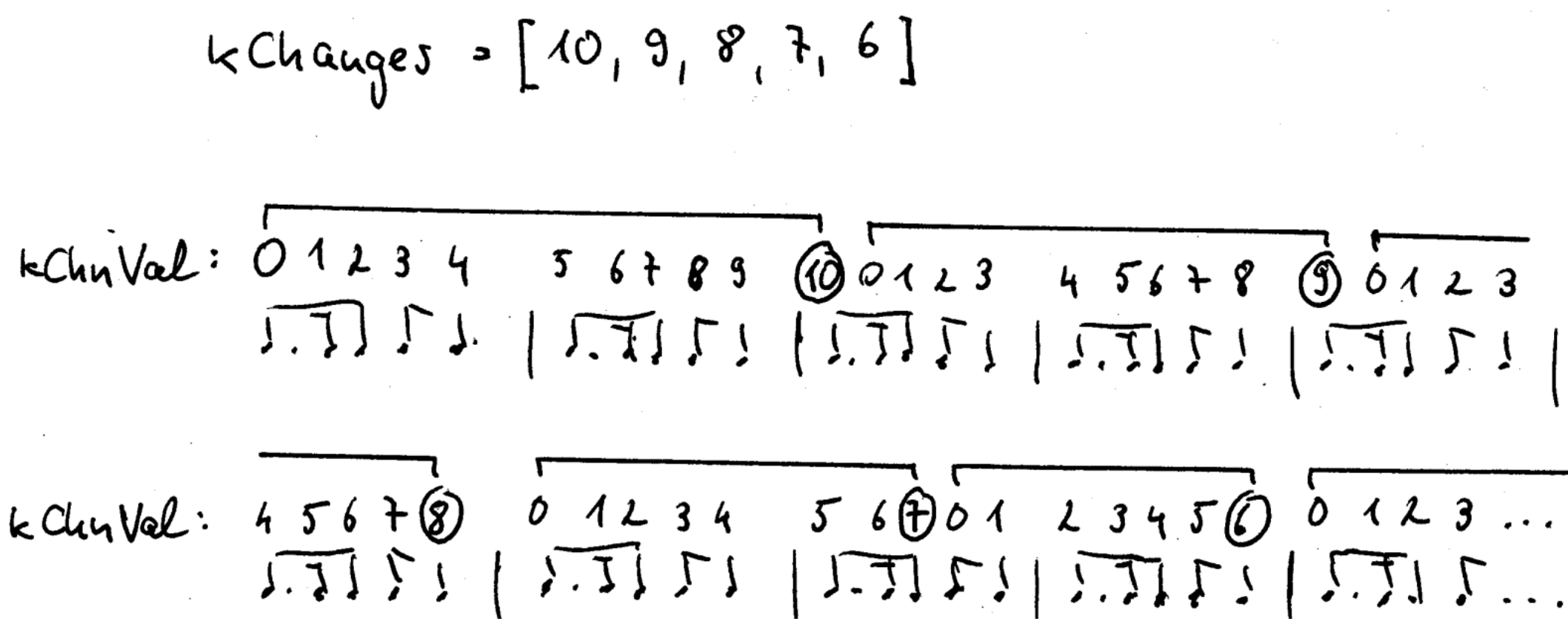
6 More modifications

III_10.csd — add or subtract in probability 2:1
III_11.csd — start the process later



7 Possibilities

III_12.csd — own series for the rhythm changes



III_13.csd — new pitch for every change **DIY**

III_14.csd — implement Sisyphos pitch movement

DIY: 1. move the pitch boundaries

2. change the values in kChanges[] during performance