Preliminary study for a *chorus* opcode

Daniele Cucchi and Stefano Cucchi

I.T.B. Project Studio
d_cucchi_1976@yahoo.it
s.cucchi@itbprojectstudio.com

Abstract. In this paper we submit the hypothesis of a new "chorus"opcode in *Csound*. We wrote a simple program in Octave language witch generate random values read by Csound in a further step. These values are used to modify the original playback speed of a audio file. Good choice is to use white sequence of uniform distribuited samples filtered by 1-pole system to obtain low-pass behaviour. Some considerations about amplitude distribution of results and proposals to manage it will be done.

Keywords: Chorus, New opcode, Asynchronous playback, Variable delay, Random speed, Random, Noise.

1 Introduction

One of the main features of computer music is the "perfection"of synthetized sound and music in terms of intonation, timing, waveforms, etc... Avoiding any kind of aesthetic judgement, there are many cases in which we want add some imperfection to the sound in order to make it more "real"and, maybe, pleasant to the ears. There are many ways to obtain the same results, tipically using through noise. The idea of this paper is to analyze some aspects of the noise "opcodes" and of introduce an alternative form of it.

2 The "noise"opcode in *Csound*

In *Csound* there is an effective "noise"opcode with a IIR lowpass filter.

$$Y_n = \sqrt{(1-\beta^2)} * X_n + \beta Y_{(n-1)} \tag{1}$$

The β (*kbeta* value in the score) determines the filter's cutoff frequency. According to the differents values of *kbeta* the behavior of the noise can vary from oscillation around a value to a kind of "drift"similar to the loss of intonation typical of analogic instruments. The *a-rate* noise can be used in association with the couple *phasor - tablei* in order to have subtle changes in speed, or downsampled to a *k-rate* signal to achieve little fluctuation of volume or intonation.

Asimptoticaly, the values of Y are distribuited like a gaussian, so it means that not all possible values are extracted with the same probability. Moreover the maximum values reached depends by the length of the sequence: the longer the sequence then more possibility you have to extract higher values. This is not a real problem with the most part of applications but sometimes could be useful to have a major control of amplitude distrubution.

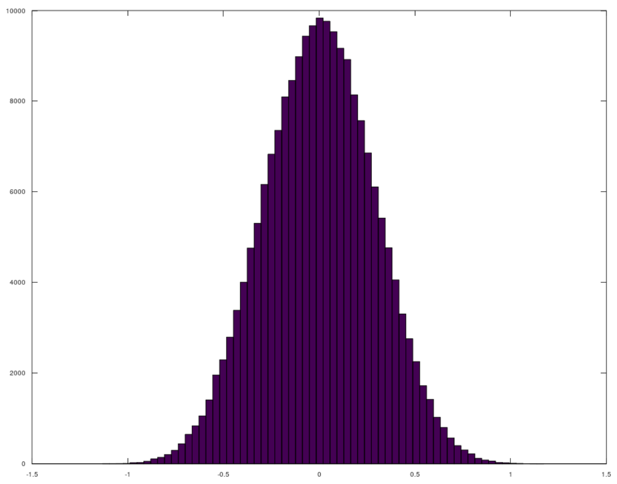

Fig. 1. original opcode

The figure 1 is the histogram distribution obtained by invoking noise opcode with parameter β of value 0.9.

3 The first modified noise generator

We define T (threshold) as the maximum admitted value for the sequence. So we modify the original equation as below:

$$W = \sqrt{(1-\beta^2)} * X_n + \beta Y_{(n-1)} \tag{2}$$

$$Y_n = min(|W|, T) * sign(W) \tag{3}$$

With this form of recursive formula we have the assurance that no value has module greater than T.

The figure 2 is the o obtained histogram distribution with T=0.7. It's clear that this simple algorithm is not the optimal one cause it create artificial and unwanted excess of extracted samples with value T.

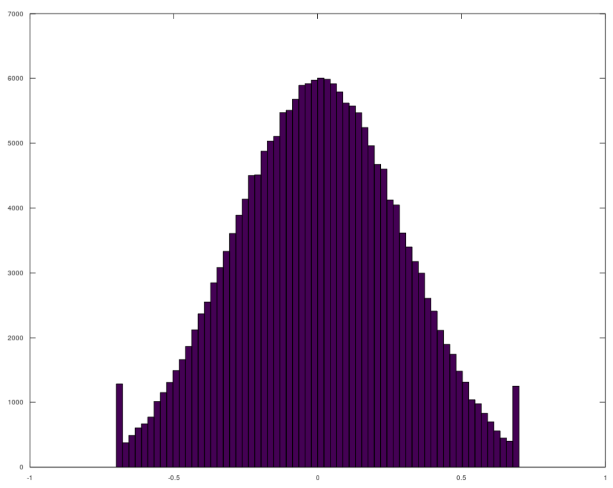

Fig. 2. 0.7 threshold

4 The second modified noise generator

The kernel of evolution is the well kown equation.

$$W = \sqrt{(1-\beta^2)} * X_n + \beta Y_{(n-1)} \tag{4}$$

If abs(W) is smaller than T nothing happens, otherwise some bounce back from threshold is implemented. The Octave code describe one of possible implementation of this bounce.

As expected there are no peaks of distribution around T visible in figure 3.

4.1 Octave Code

Here Octave code used to generate random sequences

Octave code

```
clear
close all

rand("seed",32);

standard_T = 0;
wrapped_T  = 1;
T = 0.7;
```

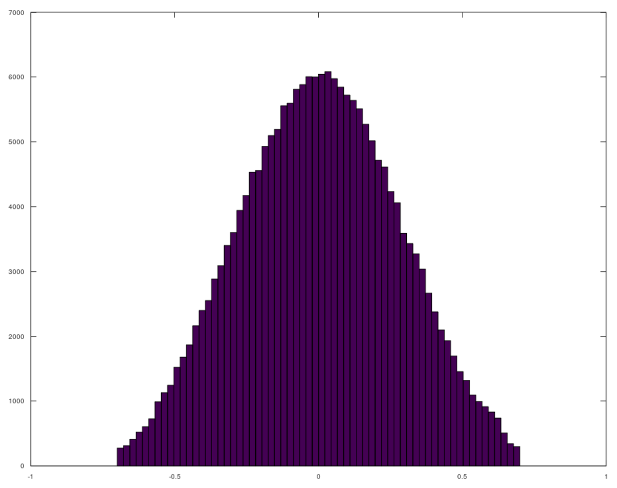

Fig. 3. 0.7 advanced threshold

```
nomefile="random_y";
L = 200000;
beta = 0.9;

x = rand(1,L);
x = x-0.5;

base = 0;

for k =1:L
  passo = x(k)*sqrt(1-beta*beta);
  old_base = base;
  base = base*beta+passo;

     if wrapped_T == 1
        if base > T
           passo = passo - (T-old_base);
           base = T - passo;
        end;
        if base < -T
           passo = passo - (-T-old_base);
           base = -T - passo;
        end;
     end;
     if standard_T == 1
        if base > T
           base = T;
        end;
        if base < -T
           base = -T;
        end;
     end;


  y(k) = base;
endfor;
```

4.2 *Csound* code

Here the *Csound* instrument used to test some sequences. This code use the input sequence to vary the playback speed of an audio file, but it's only one of the possible utilizations.

Csound code

```
<CsOptions>
</CsOptions>
<CsInstruments>
sr =  44100
kr =  4410
ksmps   =  0
nchnls =  2
0dbfs = 1
strset 1, "sinusoide.aif"
instr 1
iformat1 = 7
iprd1 = 0.08
kveldev1 readk "random_y", iformat1, iprd1
Sfile strget p4
aoriginal diskin2 Sfile, 1
achorus1 diskin2 Sfile, 1 + (kveldev1*p5)
outch 1, aoriginal
outch 2, achorus1
endin
</CsInstruments>
<CsScore>
i1 0 10 1 0.99
e
</CsScore>
```

```
</CsoundSynthesizer>
```

5 Conclusions

We described two variation of original noise opcode which give us an adjoint parameter controlling the dynamic of the system and can be useful when it's important to limit the maximum module of the sequence. Probably to really arrive to define a new opcode would be necessary a more depth study about control of amplitude distribution. It should be interesting investigate also the combination of different effects varying β and T parameters.

References

1. ffitch, J.: A look at Random Numbers, Noise, and Chaos with Csound. In: R. Boulanger (ed.) The Csound Book, pp. 321–338. MIT Press, Cambridge (2000)
2. Csound Github site, `http://csound.github.io`