Sound Synthesis with Periodic Linear Time-Varying Filters

Antonio GOULART
Marcelo QUEIROZ
Computer Science Department
Sonology Research Center
University of São Paulo
São Paulo - Brazil
{ag, mqz}@ime.usp.br

Joseph TIMONEY
Victor LAZZARINI
Sound and Digital Music Technology Group
National University of Ireland - Maynooth
Maynooth - Co. Kildare - Ireland
{joseph.timoney, victor.lazzarini}@nuim.ie

Abstract

In this survey we explore implementations of recently proposed distortion synthesis techniques, namely the Feedback Amplitude Modulation (1st and 2nd order cases) and Allpass filter coefficient modulation. These techniques are based on Periodic Linear Time-Varying systems, in which we operate by finding a suitable modulation function to obtain a desired spectrum. In order to illustrate this survey and encourage exploration of these new techniques we present examples in the Csound language.

Keywords

allpass filter coefficient modulation, feedback amplitude modulation, periodic linear time-varying systems, distortion synthesis

1 Introduction

Knowledge of a good set of synthesis techniques is a must for electronic / electroacoustic / computer musicians, sound designers and synthesiser builders. Audio effects plugins developers also benefit from dealing with synthesis techniques that can also be used in an audio effect context, such as the ones that will be presented.

In a previous survey, classic and recent distortion synthesis techniques along with their implementations were presented at the LAC [Lazzarini, 2009]. Waveshaping [LeBrun, 1979], phaseshaping [Ishibashi, 1987], and summation formulae [Moorer, 1976], some of the classic approaches to distortion synthesis, were analysed mathematically and demonstrated in Csound code. More recent or unusual approaches like asymmetrical FM [Palamin et al., 1988], Phase Aligned Formants [Puckette, 1995], and Modified FM synthesis [Lazzarini and Timoney, 2010] were also exposed. Other, more common distortion techniques, are also widely presented in computer music textbooks [Dodge and Jerse, 1997] [Moore, 1990] [Puckette, 2007].

In this paper, we address some recent techniques which extend the distortion synthesis family, namely techniques based on Periodic Linear Time-Varying (PLTV) systems. With these new approaches we operate by modulating the coefficients of a filter, and it was shown [Pekonen, 2008] that it results in a kind of dynamic version of phase distortion [Ishibashi, 1987]. A thorough study of time-varying systems applied to these techniques [Cherniakov, 2003] [Timoney et al., 2014] discloses the appropriate tools for properly understanding and using these new synthesis processes. These systems are intended to operate differently than audio effects such as time-varying delay lines used in flanging [Zolzer, 2011] and allpass filters used to obtain variable fractional delays [Pekonen et al., 2010]. Additionally, they do not replace another type of variation within musical systems where users manually change (e.g. with a knob) parameter values to achieve a sonic effect, such as mentioned in [Wishnick, 2014].

Our motivation for presenting this survey comes from the fact that most audio programmers' work is supported by classic Linear Time-Invariant (LTI) systems theory [Oppenheim and Schafer, 1975], but Linear Time Varying (LTV) systems theory is less well covered in the literature [Huang and Aggarwal, 1982]. Also, time-varying tools were indeed considered a long time ago [Layzer, 1971] [Risset, 1969] for audio applications within our context, but only recently are being tackled in a more comprehensive fashion. A good understanding of LTV theory can bring us new kinds of synthesis/effects techniques and different ways for implementing established ones (which can bring nice variations or reduce computational costs).

We will dedicate each of the next sections to one technique and then conclude the text. The aim of this paper is not to delve into the theory of LTV systems; our intention, instead, is to present another look at some techniques and also provide code[1] for their implementation.

[1] `http://www.ime.usp.br/~ag/dl/lac15-code.zip`

2 Allpass coefficient modulation

An allpass filter contains poles and zeros at reciprocal distances from the origin, so their effects on all frequencies are balanced [Moore, 1990]. A typical 1st order invariant allpass would have a fixed coefficient, but a time-varying one has the transfer function

$$H(z,t) = \frac{-m(t) + z^{-1}}{1 - m(t)z^{-1}}, \quad (1)$$

with $m(t)$ as the modulating function which drives the coefficient. Using this filter, a time-varying phase distortion given by [Timoney et al., 2009] [Laakso et al., 1996]

$$\phi(\omega,t) = -\omega + 2\tan^{-1}\left(\frac{-m(t)\sin(\omega)}{1 - m(t)\cos(\omega)}\right) \quad (2)$$

is introduced in the signal. If we know how we want the phase to be distorted – in other words, if we know $\phi(\omega,t)$ – we can use the approximation $\tan(x) \approx x$ [Timoney et al., 2009] and determine the modulating signal as

$$m(t) = \frac{-(\phi(\omega,t) + \omega)}{2\sin(\omega) - (\phi(\omega,t) + \omega)\cos(\omega)}. \quad (3)$$

The allpass filtering described can be implemented with the difference equation [Lazzarini et al., 2009b]

$$y(n) = x(n-1) - m(n)(x(n) - y(n-1)). \quad (4)$$

The condition $|m(n)| < 1, \forall n$, assures stability [Cherniakov, 2003] and a DC offset

$$DC(n) = \frac{1 - m(n)}{1 + m(n)} \quad (5)$$

is introduced in the signal [Pekonen, 2008].

2.1 Classic phase distortion emulation

In [Pekonen, 2008] [Lazzarini et al., 2009b] [Timoney et al., 2014], we find as an example for this technique the emulation of the classic phase distortion [Ishibashi, 1987]. The phase distortion algorithm consists of reading a cosine table in an unusual way. Instead of getting indexes from a regular phase generator (which goes from 0 to 1 in a period related to the chosen fundamental frequency) multiplied by the table size, we add another function to the phase generator values and then read the cosine wavetable. In order to get an approximation of a sawtooth from a cosine, we must read the rising part of the cosine in a shorter time, and take more time to read the decaying portion. The function we must add to the phase generator, in this case, is drawn on the upper panel of Figure 1 and is given by [Lazzarini et al., 2009b]

$$g(x) = \begin{cases} (\frac{1}{2} - d)\frac{x}{d}, & x < d \\ (\frac{1}{2} - d)\frac{1-x}{1-d}, & x \geq d, \end{cases} \quad (6)$$

where d is how long it takes to read from the start of the cosine table up to its maximum, and the smaller it is the more abrupt the rising ramp will be and the more distortion we will obtain.

If we want to implement $g(x)$ using a modulated allpass filter we will get better results if $\phi(\omega,t)$ goes from $-\omega$ to $-\pi$ [Lazzarini et al., 2009b], so we make

$$\phi(\omega,t) = \frac{g(t)((1-2d)\pi - \omega)}{(1-2d)\pi} - (1-2d)\pi - \omega. \quad (7)$$

The resulting modulation function is shown on the lower panel of Figure 1. On Figures 2 and 3 we can see the waveforms and spectra of the sawtooths generated with both the original technique and the modulated allpass based one. Notice that the missing components of the classic phase distortion technique are actually present in the spectrum of the modulated allpass output. We can also see and hear that the latter spectrum is richer. The code for implementing this example is given in Listing 1, where we can see two instruments; the first one works as a synthesis instrument, distorting the sinusoid and producing richer spectra; the second one applies the same technique as an audio effect, distorting the sound of a pre-recorded flute (any sound file can be used, or even a microphone input).

Listing 1: Classic phase distortion emulation as synthesis (instrument 1) and technique used as an audio effect (instrument 2)

```
<CsoundSynthesizer>

<CsOptions>
-o dac
</CsOptions>

<CsInstruments>

0dbfs=1

```

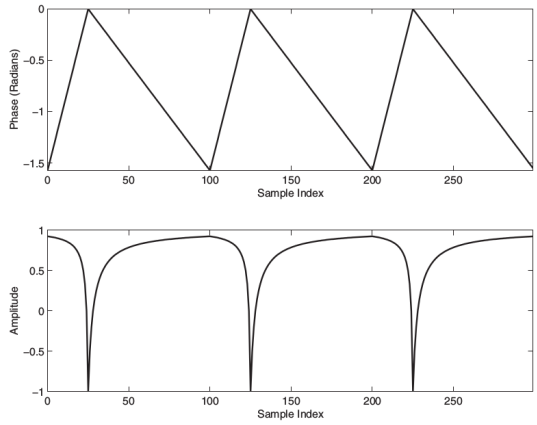

Figure 1: Phase distortion function (upper panel) and coefficient modulation function (lower panel). Source: [Timoney et al., 2014]

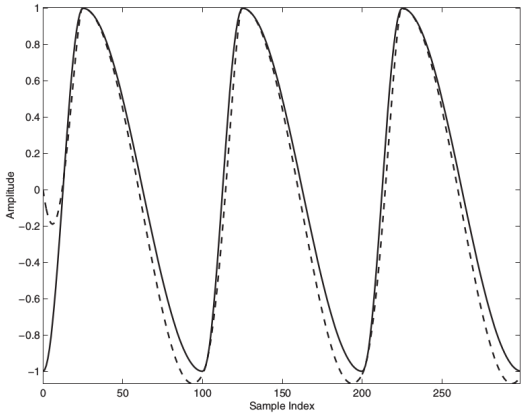

Figure 2: Waveforms generated with the classic technique (solid line) and the modulated allpass (dashed line). Source: [Timoney et al., 2014]

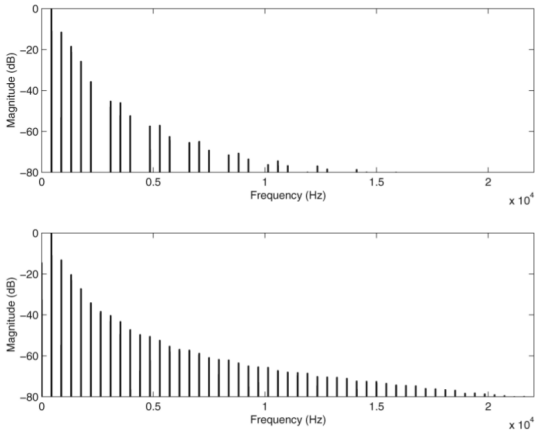

Figure 3: Spectra obtained with the classic technique (upper panel) and with modulated allpass (lower panel). Source: [Timoney et al., 2014]

```
11 /* audio-rate coeff allpass */
12 opcode Allpass,a,aa
13   adel init 0
14   setksmps 1
15   audio,acoef xin
16   aw = audio + acoef*adel
17   ay = -aw*acoef + adel
18   adel = aw
19   xout ay
20 endop
21
22
23 /* PD function */
24 /* inflection point */
25 gip = 0.1
26
27 /* ftgen producing line segments  */
28 ipdfun ftgen 1,0,16384,7,0,16384*gip
       ,1,16384*(1-gip),0
29
30 /*
31  instr 1:
32  PD synthesis
33  using sine wave input
34 */
35
36 instr 1
37   ifr = p5
38   iamp = p4
39
40   /* regular phase generator */
41   aph phasor ifr
42   /* phase distortion signal */
43   apd tablei aph, 1, 1, 0, 1
44
45   /* scaling of pd signal (eq.7)*/
46   iw = 2*$M_PI*ifr/sr /*omega value*/
47   ia = (1 - 2*gip)*$M_PI
48   apd = apd*ia
49   apd = apd*(ia - iw)/ia - ia - iw
50
51   /* coefficient modulation function
52      obtained from phase distortion
53      signal (equation 3) */
54   /* envelope for modulation */
55   kmod linseg 0,1,1,p3-2,1,1,0
56   amod = -kmod*(apd + iw)/(2*sin(iw)
       - (apd+iw)*cos(iw))
57
58   /* sine input */
59   asin tablei aph,-1,1,0,1
60
61   /* allpass */
62   asig Allpass asin,amod
63
64   /* envelope */
65   aout linenr asig * iamp, 0.01, 0.1,
         0.01
66
67   outs aout, aout
68 endin
69
70 /*
71  instr 2:
72  PD adaptive synthesis
73  using a monophonic instr input
```

```
74 */
75
76 instr 2
77   iamp = p4
78   /* input signal */
79   afl diskin2 "flutec3.wav",1,0,1
80
81   /* pitch tracking */
82   kfr,kamp ptrack afl,2048
83   kfr port kfr,0.01
84
85   /* master phase signal */
86   aph phasor kfr
87   /* phase distortion signal */
88   apd tablei aph,1,1,0,1
89
90   /* scaling of pd signal */
91   kw = 2*$M_PI*kfr/sr
92   ia = (1 - 2*gip)*$M_PI
93
94   apd = apd*ia
95   apd = apd*(ia - kw)/ia - ia - kw
96
97   /* envelope for modulation */
98   kmod linseg 0,1,1,p3-2,1,1,0
99
100  amod = -kmod*(apd + kw)/(2*sin(kw)
         - (apd+kw)*cos(kw))
101
102  asig Allpass afl,amod
103  aout linenr asig * iamp, 0.01, 0.1,
         0.01
104
105  outs aout, aout
106 endin
107
108 </CsInstruments>
109
110
111 <CsScore>
112
113 /* uncomment lines to
114    run instruments */
115 i 1 0 10 0.5 440
116 ;i 2 0 10 0.5
117
118
119 </CsScore>
120
121
122 </CsoundSynthesizer>
```

2.2 Finding a distortion function

Now we present a new example with the allpass coefficient modulation technique to bring more insight. In this example, instead of choosing a desired waveform and then finding out how to modulate the allpass coefficient, we choose an arbitrary signal to distort the phase. Keeping in mind that we should generate a signal within the appropriate range, that is [-1,1], any signal a priori can be considered for the process.

So we embarked on an experiment that involved the summing of partials in order to find a function to distort the phase of an input sinusoid and thus get a more musically interesting timbre. The expression we used to create our example is given by

$$y(n) = 0.4\cos(f_0) + 0.4\cos\left(2f_0 - \frac{\pi}{3}\right) + 0.35\cos\left(3f_0 + \frac{\pi}{7}\right) + 0.3\cos\left(4f_0 + \frac{4\pi}{3}\right) \quad (8)$$

In order to shift the output of Equation 8 to the appropriate range we use the scaling

$$y_s(n) = -\frac{\pi}{2}\frac{(y(n)+1)}{2}. \quad (9)$$

Listing 2 presents Csound code for implementing this new example. The upper panel of Figure 4 shows the distortion function for this example, while the lower panel shows the allpass modulation function resulting after applying Equation 3. Figure 5 shows the waveform and spectrum obtained. Albeit this was a naive example, it demonstrates in an intuitive way how a phase function could be easily generated using additive synthesis and then used to drive the allpass filter coefficient after applying Equations 9 and then 3, highlighting another musical possibility for using the allpass filter.

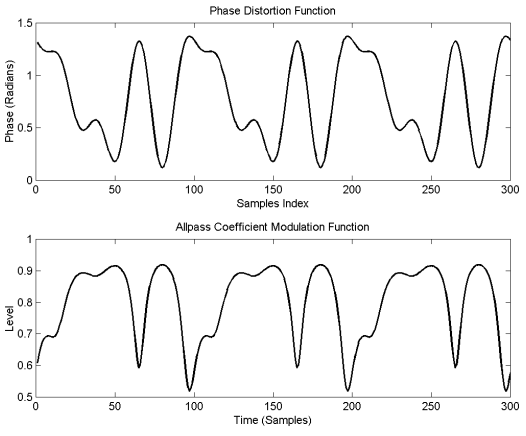

Figure 4: Phase distortion (upper panel) and resultant modulation (lower panel) functions.

Listing 2: Implementation of an arbitrary function as a phase distorter.

```
1 <CsoundSynthesizer>
2
3 <CsOptions>
```

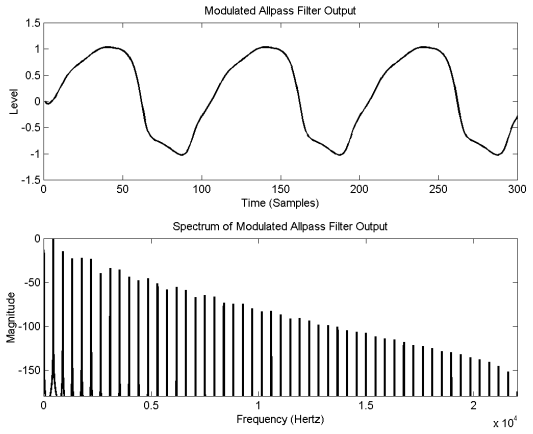

Figure 5: Waveform and spectrum obtained with arbitrary function

```
 4 -o dac
 5 </CsOptions>
 6
 7 <CsInstruments>
 8 0dbfs=1
 9
10 /* audio-rate coeff allpass */
11 opcode Allpass,a,aa
12   adel init 0
13   setksmps 1
14   audio,acoef xin
15   aw = audio + acoef*adel
16   ay = -aw*acoef + adel
17   adel = aw
18   xout ay
19 endop
20
21 /* PD function */
22 ipdfun ftgen 1, 0, 16384, 9, 1, 0.4,
       -90, 2, 0.4, -150, 3, 0.35,
       180/7-90, 4, 0.3, 4*180/3-90
23
24 instr 1
25
26   ifr = p5
27   iamp = p4
28
29   iw = 2*$M_PI*ifr/sr ;omega
30
31   aph phasor ifr  ;regular phase
32   /* phase distortion signal */
33   apd tablei aph, 1, 1, 0, 1
34
35   /* coef mod signal (eq.3)*/
36   apd = -0.5*$M_PI*(apd+1)/2 ;scaling
37   amod = -1*(apd + iw)/(2*sin(iw) - (
         apd+iw)*cos(iw))
38
39   /* sine input */
40   asin tablei aph,-1,1,0,1
41   /* allpass */
42   asig Allpass asin,amod
43   /* envelope */
44   aout linenr asig*iamp,0.01,0.1,0.01
45
46   outs aout, aout
47 endin
48
49 </CsInstruments>
50
51 <CsScore>
52 i 1 0 10 0.25 440
53 </CsScore>
54
55 </CsoundSynthesizer>
```

3 Feedback amplitude modulation

The Feedback Amplitude Modulation was mentioned by Layzer (1971) and described and implemented by Risset (1969) in his catalogue example #510, but a rigorous mathematical analysis of the technique was lacking. Since 2009 there was a renewed research interest in exploiting its musical possibilities [Lazzarini et al., 2009a], [Kleimola et al., 2011], [Lazzarini et al., 2011].

First of all we will review its 1st order case. The basic idea is to modulate the amplitude of an oscillator using its previous output, as in

$$y(n) = \cos(\omega_0 n)[1 + y(n-1)], \qquad (10)$$

with $\omega_0 = 2\pi f_0$ and the initial condition $y(n) = 0$ for $n \leq 0$.

A first analysis [Kleimola et al., 2011] is made expanding Equation 10 as

$$\begin{aligned} y(n) = \cos(\omega_0 n) + \\ \cos(\omega_0 n)\cos(\omega_0[n-1]) + \\ \cos(\omega_0 n)\cos(\omega_0[n-1])\cos(\omega_0[n-2]) + \\ \ldots \end{aligned} \qquad (11)$$

$$y(n) = \sum_{k=0}^{\infty} \prod_{m=0}^{k} \cos(\omega_0[n-m]), \qquad (12)$$

showing that the resultant signal is composed by harmonics of the fundamental f_0. A more interesting case is when we can control the amount of feedback in the system, so we introduce the feedback parameter β and Equation 10 becomes

$$y(n) = \cos(\omega_0 n)[1 + \beta y(n-1)]. \qquad (13)$$

The feedback parameter can be interpreted to be similar to modulation index of conventional FM synthesis, and its influence on FBAM's

spectral evolution is shown in Figure 6, borrowed from [Kleimola et al., 2011].

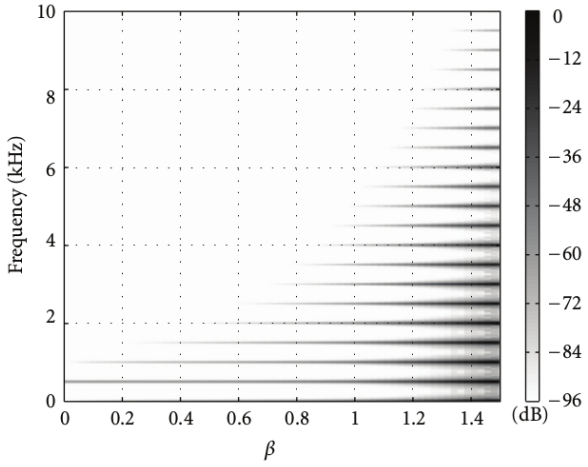

Figure 6: β influence on FBAM spectrum. Source: [Kleimola et al., 2011]

The interpretation of the system as a time-varying filter enables a better analysis. The structure for the 1st order case is

$$y(n) = x(n) + a(n)y(n-1), \tag{14}$$

with

$$x(n) = a(n) = \cos(\omega_0 n), \tag{15}$$

so we have a coefficient modulated IIR filter. As in the previous section, the modulation will create the dynamic phase distortion, generating new partials, and in this case the filter is not allpass, so it has a time-varying non-flat magnitude response.

Applying the equations for stability analysis by Cherniakov (2003) a condition for stability of this system was derived as

$$\left| \beta \prod_{m=1}^{N} \cos(\omega_0 m) \right| < 1. \tag{16}$$

An expression for the maximum value of beta was proposed by Kleimola et al. (2011) as

$$\beta_{max} \approx 1.9986 - 0.00003532(f_0 - 27.5), \tag{17}$$

and they also showed that as the value of β is increased, the presence of significant components in the output due to aliasing can be observed before the system becomes unstable. Listing 3 presents code for the 1st order FBAM implementation.

Listing 3: 1st order FBAM

```
<CsoundSynthesizer>

<CsOptions>
-o dac
</CsOptions>

<CsInstruments>

0dbfs= 1

opcode FBAM, a, kkki
  ; set vector size to 1 sample
  setksmps 1
  ay init 0 ; y[0] = 0
  ka, kf, kb, ifn   xin
  /* current sample +
  weighted previous sample */
  ay oscili ka + kb*ay, kf, ifn
  xout ay
endop

instr 1
        ; amp, freq, beta
  a1 FBAM 0.5, 440, 0.7, 1
  out a1
endin

</CsInstruments>

<CsScore>
f1 0 16384 10 1
i1 0 5
</CsScore>

</CsoundSynthesizer>
```

The 2nd order FBAM is obtained using two previous outputs in the modulation, each with its own feedback parameter. The system equation is then given by

$$y(n) = \cos(\omega_0 n)[1 + \beta_1 y(n-1) + \beta_2 y(n-2)]. \tag{18}$$

It was shown [Lazzarini et al., 2011] that with this system we can get a narrower pulse, and thus a richer spectral output for the FBAM system, as shown in Figure 7, borrowed from [Lazzarini et al., 2011]. Code for implementing the 2nd order FBAM is presented in Listing 4.

Listing 4: 2nd order FBAM

```
<CsoundSynthesizer>

<CsOptions>
-o dac
</CsOptions>

<CsInstruments>
ksmps = 10
0dbfs = 1
nchnls = 2
```

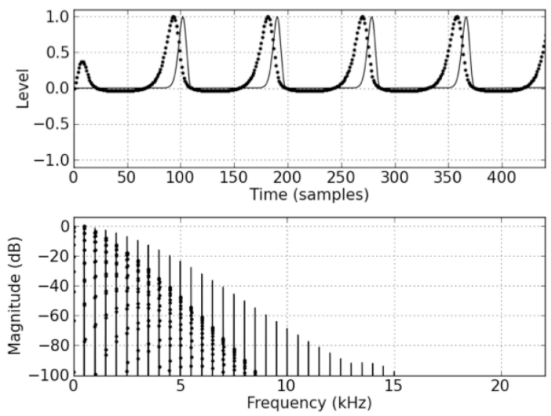

Figure 7: Comparison between 1st (in dots) and 2nd order (solid line) FBAM for 500 Hz fundamental. Source: [Lazzarini et al., 2011]

```
11
12 opcode Fbam2,a,akk
13   setksmps 1
14   asigm1 init 0
15   asigm2 init 0
16   ain,kb1,kb2 xin
17   asig =  ain*(1 + kb1*asigm1 + kb2*
         asigm2)
18   asigm2 = asigm1
19   asigm1 = asig
20   xout asig
21 endop
22
23 instr 1
24   kb1 = 0.7
25   kb2 = 0.7
26   ; sinusoidal input
27   ain oscili 1, 440, -1, 0.25
28   asig Fbam2 ain,kb1,kb2
29   asig balance asig, ain
30   outs asig,asig
31 endin
32
33 </CsInstruments>
34
35 <CsScore>
36 i 1 0 5
37 </CsScore>
38
39 </CsoundSynthesizer>
```

For a deeper analysis we can understand the system as a 2nd order PLTV system by rewriting the system equation as

$$y(n) = x(n)+\beta_1 a_1(n)y(n-1)+\beta_2 a_2(n)y(n-2), \quad (19)$$

with $x(n) = a_1(n) = a_2(n) = \cos(\omega_0 n)$. This is the simplest example but the analysis equations are complicated for this second order system. However we can also uncouple the input and modulation signals as independent streams, and treat the whole system as the combination of two first order units to reduce the difficulty of the analysis.

Despite the initial results about the 2nd order FBAM that were already reported, we plan to proceed with more thorough investigations especially regarding its stability and to determine any consequent restrictions it might have on the coefficient modulation waveform.

4 Conclusions

In this survey we presented some techniques for sound synthesis derived from the concept of phase distortion. Results can be used to generate approximations of classic sawtooth oscillators and more timbrally-involved FM-like spectra. Implementations in Csound, which are easily translated to other languages, were presented in order to promote the techniques among our community and audio tools developers.

The application of time-varying systems is not new in general signal processing, but only recently is the theory behind these systems being explored for time-varying digital audio filter systems. This paper shows there is a significant potential for a number of novel techniques based on PLTV systems. The field is still open to investigation, in particular with regards to 2nd and higher order systems.

We hope that this brief survey invites more musicians and technicians to explore the nice retro motivated sounds obtained with these systems, maintaining the interest in distortion synthesis/effects techniques. We hope to encourage the sharing of ideas and principles around these techniques, such as, for instance, nice distortion functions that could be used either in synthesis or acoustic instruments processing.

5 Acknowledgements

The research leading to this paper was partially supported by CAPES (proc. number 8868-14-0) and Science Foundation Ireland (ISCA-Brazil).

References

Mickhail Cherniakov. 2003. *An introduction to parametric digital filters and oscillators.* John Wiley & Sons Ltd, England.

Charles Dodge and Thomas Jerse. 1997. *Computer Music: Synthesis, composition and performance.* Schirmer Books, New York, NY, USA, segunda edition.

N.C. Huang and J.K. Aggarwal. 1982. Time varying digital processing: a review. In *Proceedings IEEE Int. Symp. Cas.*, pages 659–662, Rome, Italy.

Masanori Ishibashi. 1987. Electronic musical instrument (patent us 4658691), April.

Jari Kleimola, Victor Lazzarini, Vesa Valimaki, and Joseph Timoney. 2011. Feedback amplitude modulation synthesis. *EURASIP Journal on Advances in Signal Processing*, 2011(434378).

T. I. Laakso, V. Valimaki, M. Karjalainen, and U. Laine. 1996. Splitting the unit delay - tools for fractional delay filter design. *IEEE Signal Processing Magazine*, 13(1):30–60.

A. Layzer. 1971. Some idiosyncratic aspects of computer synthesized sound. In *Proceedings of the Annual Conference American Society of University Composers*, pages 27–39.

Victor Lazzarini and Joseph Timoney. 2010. Theory and practice of modified frequency modulation synthesis. *Journal of the Audio Engineering Society*, 58(6):459–471.

Victor Lazzarini, Joseph Timoney, Jari Kleimola, and Vesa Valimaki. 2009a. Five variations on a feedback theme. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-09)*, Como, Italy.

Victor Lazzarini, Joseph Timoney, Jussi Pekonem, and Vesa Valimaki. 2009b. Adaptive phase distortion synthesis. In *Proceedings of the International Conference of Digital Audio Effects*, pages 28–35.

Victor Lazzarini, Jari Kleimola, Joseph Timoney, and Vesa Valimaki. 2011. Aspects of second-order feedback am synthesis. In *Proceedings of the International Computer Music Conference*, University of Huddersfield, UK.

Victor Lazzarini. 2009. A distortion synthesis tutorial. In *Proceedings of the Linux Audio Conference*, pages 12–20.

Marc LeBrun. 1979. Digital waveshaping synthesis. *Journal of the Audio Engineering Society*, 27(4):250–266.

F. Richard Moore. 1990. *Elements of computer music*. Prentice Hall, Englewood Cliffs, New Jersey, USA.

James A. Moorer. 1976. The synthesis of complex audio spectra by means of discrete summation formulas. *Journal of the Audio Engineering Society*, 24(9):717–727.

A.V. Oppenheim and R.W. Schafer. 1975. *Digital signal processing*. Prentice Hall, New Jersey, USA.

Jean-Pierre Palamin, Philippe Palamin, and André Ronveaux. 1988. A method of generating and controlling musical asymmetrical spectra. *Journal of the Audio Engineering Society*, 36(9):671–685.

J. Pekonen, V. Valimaki, J. Nam, J.O. Smith, and J.S. Abel. 2010. Variable fractional delay filters in bandlimited oscillator algorithms for music synthesis. In *International Conference on Green Circuits and Systems (ICGCS)*, pages 148–153, June.

J. Pekonen. 2008. Coefficient modulated first-order allpass filter as a distortion effect. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-08)*, pages 83–87, Espoo, Finland.

Miller Puckette. 1995. Formant-based audio synthesis using non-linear distortion. *Journal of the Audio Engineering Society*, 43(1):40–47.

Miller Puckette. 2007. *The theory and technique of electronic music*. World Scientific Press, River Edge, NJ.

Jean Claude Risset. 1969. *An introductory catalogue of computer synthesized sounds*. Bell Laboratories, Murray Hill, New Jersey.

J. Timoney, V. Lazzarini, J. Pekonen, and V. Valimaki. 2009. Spectrally rich phase distortion sound synthesis using an allpass filter. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICCASP-09)*, pages 293–296, Taipei, Taiwan.

Joseph Timoney, Jussi Pekonen, Victor Lazzarini, and Vesa Valimaki. 2014. Dynamic signal phase distortion using coefficient-modulated allpass filters. *Journal of the Audio Engineering Society*, 62(9).

Aaron Wishnick. 2014. Time-varying filters for musical applications. In *Proceedings of the 17th International Conference on Digital Audio Effects (DAFx-14)*, pages 69–76, Erlangen, Germany.

Udo Zolzer, editor. 2011. *DAFX: Digital Audio Effects*. John Wiley & Sons, UK.