SOUND SYNTHESIS BY ITERATED NONLINEAR FUNCTIONS

by Agostino Di Scipio

1. INTRODUCTION

The iteration of nonlinear functions is part of the branch of mathematics called "chaos theory". In 1991, I set out to investigate the possibility of using it in the field of digital sound synthesis. It was soon clear that the approach was to open up a world of unique sounds of its own, many with turbulent and noisy components (aperiodicity), and some with more familiar harmonic sound spectra (periodicity).

This present overview is a survey of iterated function synthesis based on Csound examples that the reader can analyze and modify for him/herself (readers willing to go into the theoretical and mathematical details are referred to the references). If you try to extend the examples provided here, it will be evident that even slight changes in the synthesis parameters give rise to dramatic differences in the sound, such that often the audible result can hardly be foreseen at the outset. Which suggests that the best way to deal with iterated nonlinear functions is with a kind of empirical, explorative and open-ended attitude. Indeed, the sound synthesis approach described here is a "non-standard" approach. It is not grounded on any model of scientifically proven and verified acoustical relevance.

Rather, it represents a somewhat arbitrary, procedural model which is properly understood as an interesting way of creating sequences of numbers such that, by sending these numbers to a digital-to-analog converter, sounds may eventually emerge having properties that are useful for music and sound-design.

2. GENERAL DESCRIPTION

To iterate a function is to apply some transformation, *f*, to a datum, *x(0)*, and to apply the transformation again to the first resulting value, and then again to the second resulting value, ... and so on, *n* times again:

x(n) = f (x(n-1))

We call *x(n)* the *n*th "iterate" obtained by applying *f* to *x(0)*. If *f* is nonlinear (e.g. a sine, a line broken in several segments, or any high-order polynomial) the process will determine different sequences of results. The particular sequence is dependent on the initial datum, *x(0)* and on the parameters of the function *f*. In most cases, it is impossible to predict the output series of values.

To translate this general procedure into a digital sound synthesis method, we can follow these steps:

a - initialize *x(0)* and *f*' parameters
b - take the *n*th iterate, *x(n)*, and save it as the current digital sample
c - update *x(0)* and *f*' parameters
d - repeat b and c as many times as the samples required.

In other words, the output stream of samples is the series of the *n*th iterations of *f* upon changing values in *x(0)* and *f*. If we call *i* the sample order index (discrete time index), the synthesis technique can be represented as a simple recursive formula:

x(n,i) = f(i) (x(n-1,i))

This model framework represents a class of synthesis techniques rather than a single technique. The aspect that is peculiar to all particular cases in the class is the fact that the stream of samples is calculated as the sequence of the *n*th iterations of some function. To implement a particular technique, we must select a particular nonlinear function and run a given number of iterations.

As an example, think of the technique known as *waveshaping* (often called *nonlinear distortion* in Europe). It involves the transformation of an input signal by a waveshaping function (usually a Chebychev polynomial, but can be a sine wave or something else). If the operation is repeated, feeding the output sample back into the waveshaper, we get a special case of iterated function synthesis.

Obviously, every nonlinear function determines a peculiar process of its own. However, in the literature on theory of deterministic chaos, many have stressed that the numerical sequences obtained with iterated functions are more heavily dependent on the iteration itself, rather than on the function. It is the iteration that allows coherent or chaotic patterns to emerge, not the nonlinear function being iterated.

3. IMPLEMENTATION

The Csound examples illustrated below illustrate the following iteration:

x(n) = sin (r*x(n-1))

i.e. the mapping of the sine function onto itself in the interval [-1,1]. The parameters in play include: *r* (or *control* parameter) and *x(0)* (initial datum). As a general rule, the control parameter ranges from 0 to 4, but as a matter of fact only values between 3.14

and 4 are of practical relevance for us (smaller values cause the process to move towards a "fixed-point attractor", yielding a straight line as a result). The initial datum can be assigned any value from the interval [-1,1] (but in theory it can be any real number).

What is the effect of the parameters? The control parameter, r, determines the overall timbral quality in the output sound. On the other hand, *x(0)* determines the particular series of output sample values, and is somewhat akin to the seed of a random numbers generator. Indeed, you can think of this synthesis process as a generator of "structured noise" that we can control in its internal process. The achievable results, anyway, are far more varied and rich than the results obtained with white noise generators (such as Csound's *rand* and *randh* opcodes).

To create some sound, either *x(0)* or *r* (or both) have to move across their value range. This can be done by driving them with an envelope generator or with an oscillator. By varying *x(0)* during the note and keeping *r* fixed, we obtain several sounds (several different output signals) of very similar timbral properties. The reverse is also true. By varying *r* and keeping *x(0)* constant, we obtain different sounds starting from identical initial data. By varying both *r* and *x(0)*, we obtain sounds with highly dynamic properties.

Another crucial parameter is the particular number of iterations utilized as the digital sample. In general, the higher is the iterate order, *n*, and the more chaotic and turbulent the output sound. Too high a number of iterations would generate something very close to white noise, but internally articulated and not as static. Too low a number of iterations would probably generate some silence.

Now, what would be a good way of changing *r* and/or *x(0)* in time? If we use ramps, i.e. series of linearly increasing or decreasing values, we obtain acoustical turbulences, sometime having a wind-like, or even water-like quality to the ear. If you look at the waveform of such sounds, you will notice phase- or frequency-modulation effects, which are sometimes heard as distinct gestures separated by silent pauses (the pauses are chunks of direct current signal, with either a positive or negative offset). That happens especially when a small number of iterations are used, and when *r* is close to 3.14. These strange rumbles are like the "natural state" of the mathematical model. By simply ramping either *r* or *x(0)*, we are "visiting" the whole field of possibilities implicit to the iterated function. Some of the Csound examples below, *Ex1.sco* through *Ex6.sco* (all to play with the *IFS1.orc* orchestra), illustrate this basic behavior. They show the dependency of the results on the two parameters. They also show that the timbral characteristics in the output sound are dependent on the note duration as well. Different durations imply differently quantized iterated processes, and therefore a different resultant bandwidth.

The remaining examples, *Ex7.sco* through *Ex10.sco*, illustrate the possibility of controlling *x(0)* by means of oscillating signals. This approach forces the overall process

towards more regular, periodic patterns, either in the infra-audible (rhythm) or the audible (frequency) range. These examples may eventually reveal that *r* can be considered a bit like the modulation index of FM synthesis. Indeed, in theory it would be possible to explain "feedback FM" (the output of the modulation is fed back into the system to modulate itself) as a kind of iterated nonlinear function synthesis.

The *IFS1.orc* orchestra includes three instruments, each representing a different implementation (see comments in the orchestra code). It should be used in conjunction with scores *Ex1.sco* through *Ex8.sco*. The *IFS2* orchestra introduces some slight modifications in instr 2 (mathematical details), and should be used in conjunction with the last two score examples, *Ex9.sco* and *Ex10.sco*.

```
; IFS1.orc
; sound synthesis by functional iterations
; sin(r*x) [sine map]
;
; the basic technique is instr 3
; instr 1 and 2 offer more refined controls:
; by exploiting the deterministic aspect of the process
; they also introduce some periodicity in the signal
;
;-------------------------------------------------
      sr     =   22050
      kr     =   22050
      ksmps=   1
      nchnls    =   2

;  why sr = kr?
;  Everything happens at audio rate (something even at "superaudio" rate, i.e. sr*p7).
;  However, some opcodes can only work at control rate (e.g. the program control
;  statements and value converters).

       instr 1

;  p4 = initial value for the control parameter (r)
;  p5 = final value for the control parameter (r)
;  p6 = initial value for x(0)
;  p7 = iterate number
;  p8 = stereo location (0,1)
;  p9 = increment step for x(0)
```

```
;  p10 = global amplitude

ar          init    p4                      ; initialize control parameter
irstp       init    (p5-p4)/(sr*p3)         ; initialize increment step for control parameter
ixstp       init    p9                      ; initialize increment step for x(0)
asam        init    0                       ; initialize output sample variable
kcnt        init    0                       ; initialize counter
ax          init    frac(p6)                ; initialize the initial datum, x(0)
kaxnew      init    frac(p6)                ; temporary storage variable

label:
kcnt    =    kcnt+1                         ; update counter
ax      =   sin(ar*ax)                      ; iteration x=sin(rx)

      if kcnt < p7 kgoto label ; iterate loop

asam    =           ax                      ; sample is the p7-th iterate
kcnt    =           0                       ; reset counter
ar      =           ar + irstp              ; increment control parameter
kaxnew  =           frac(kaxnew+ixstp)      ; update x(0), must be a k-variable, due to frac
ax      =           kaxnew                  ; reset the initial datum, x(0)

asig    =           p10*asam                ; amplitude scaling
asig    linen   asig, .05, p3, .05          ; fade in, fade out, to avoid clicks
        outs    asig*p8, asig*(1-p8)        ; stereo panning

        endin

        instr 2

; same pfields as above

ar          init    p4                      ; initialize control parameter
irstp       init    (p5-p4)/(sr*p3)         ; initialize increment step for control parameter
asam        init    0                       ; initialize output sample variable
kcnt        init    0                       ; initialize counter

aph         oscili  1,p9,1                  ; oscili determines the initial datum for next iteration
ax          =       (p6*2)*((aph+1)/2)      ; update initial datum, x(0)
```

```
label:
kcnt      =     kcnt+1                    ; increment counter
ax        =     sin(ar*ax)                ; iteration x=sin(rx)

if kcnt < p7 kgoto label                  ; iterate loop

asam      =     ax                        ; sample is the p7-th iterate
kcnt      =     0                         ; reset counter
ar        =     ar + irstp                ; increment control parameter

asig      =     p10*asam                  ; amplitude scaling
asig      linen asig, .05, p3, .05        ; fade in, fade out, to avoid clicks
          outs  asig*p8, asig*(1-p8)      ; stereo panning

          endin

          instr 3

; same pfields as above, but p9 void

ar        init  p4                        ; initialize control parameter
irstp     init  (p5-p4)/(sr*p3)           ; initialize increment step for control parameter
ixstp     init  (1-p6)/(sr*p3)            ; initialize increment step for x(0)
asam      init  0                         ; initialize output sample variable
kcnt      init  0                         ; initialize counter
ax        init  p6                        ; initialize initi value
axnew     init  p6                        ; temporary storage variable

label:
kcnt      =     kcnt+1                    ; increment counter
ax        =     sin(ar*ax)                ; iteration x=sin(rx)

      if kcnt < p7 kgoto label ; iterate loop

asam      =     ax                        ; sample is the p7-th iterate
kcnt      =     0                         ; reset counter
ar        =     ar + irstp                ; increment control parameter
axnew     =     axnew+ixstp
ax        =     axnew
```

```
asig        =       p10*asam                        ; amplitude scaling
asig        linen   asig, .05, p3, .05              ; fade in, fade out, to avoid clicks
            outs    asig*p8, asig*(1-p8)            ; stereo panning

            endin

; SCORES FOR IFS1.ORC
;
;  p4 = initial value for the control parameter (r)
;  p5 = final value for the control parameter (r)
;  p6 = initial value for x(0)
;  p7 = iterate number
;  p8 = stereo location (0,1)
;  p9 = increment step for x(0)
;  p10 = global amplitude
;
; EX1.SCO
; dependency of the output signal on the initial datum x(0)
; other parameters being constant (dur, r, iter)
;
;   start  dur     rstart   rend     x(0)     iter     stereo   xstep    ampl
i1  0      1       4        3.9      .1       10       .5       0        30000
i1  2      1       4        3.9      .11      10       .5       0        30000
i1 4       1       4        3.9      .2       10       .5       0        30000
i1 6       1       4        3.9      .3       10       .5       0        30000
i1 8       1       4        3.9      .4       10       .5       0        30000
i1 10      1       4        3.9      .6       10       .5       0        30000
i1 12      1       4        3.9      .8       10       .5       0        30000
i1 14      1       4        3.9      .9       10       .5       0        30000
s
; EX2.SCO
; two simultaneous processes start from nearly identical initial data
; and diverge in time
;
;   start  dur     rstart   rend     x(0)     iter     stereo   xstep    ampl
i1  0      10      4        3        .3       10       0        0        30000
i1  0      10      4        3        .305     10       1        0        30000
s
; EX3.SCO
```

```
; dependency of the bandwidth on r
; all other parameters being constant (dur, x, iter)
;
;   start  dur    rstart    rend    x(0)    iter    stereo   xstep   ampl
i3  0      1      2.5       2.5     .3      10      .5       0       30000
i3  2      1      2.6       2.6     .3      10      .5       0       30000
i3  4      1      2.7       2.7     .3      10      .5       0       30000
i3  6      1      2.8       2.8     .3      10      .5       0       30000
i3  8      1      2.9       2.9     .3      10      .5       0       30000
i3  10     1      3.0       3.0     .3      10      .5       0       30000
i3  12     1      3.1       3.1     .3      10      .5       0       30000
i3  14     1      3.2       3,2     .3      10      .5       0       30000
i3  16     1      3.3       3,3     .3      10      .5       0       30000
i3  18     1      3.4       3.4     .3      10      .5       0       30000
i3  20     1      3.5       3.5     .3      10      .5       0       30000
i3  22     1      3.6       3.6     .3      10      .5       0       30000
i3  24     1      3.7       3.7     .3      10      .5       0       30000
i3  26     1      3.8       3.8     .3      10      .5       0       30000
i3  28     1      3.9       3.9     .3      10      .5       0       30000
i3  30     1      4.0       4.0     .3      10      .5       0       30000
s
; EX4.SCO
; same as EX3.SCO, but with different x(0)
;
;   start  dur    rstart    rend    x(0)    iter    stereo   xstep   ampl
i3  0      1      2.5       2.5     .1      10      .5       0       30000
i3  2      1      2.6       .       .       .       .        .       30000
i3  4      1      2.7       .       .       .       .        .       30000
i3  6      1      2.8       .       .       .       .        .       30000
i3  8      1      2.9       .       .       .       .        .       30000
i3  10     1      3.0       .       .       .       .        .       30000
i3  12     1      3.1       .       .       .       .        .       30000
i3  14     1      3.2       .       .       .       .        .       30000
i3  16     1      3.3       .       .       .       .        .       30000
i3  18     1      3.4       .       .       .       .        .       30000
i3  20     1      3.5       .       .       .       .        .       30000
i3  22     1      3.6       .       .       .       .        .       30000
i3  24     1      3.7       .       .       .       .        .       30000
i3  26     1      3.7       .       .       .       .        .       30000
```

```
i3  28   1      3.9     .       .       .       .       30000
i3  30   1      4.0     .       .       .       .       30000
s
; EX5.SCO
; dependency of the bandwidth on duration
; (i.e. on the number of samples off the control parameter interval)
;
;   start  dur    rstart   rend    x(0)    iter    stereo  xstep   ampl
i3  0      1      3.5      3       .1      10      .5      0       30000
i3  2      2      3.5      3       .1      10      .5      0       30000
i3  5      3      3.5      3       .1      10      .5      0       30000
i3  9      4      3.5      3       .1      10      .5      0       30000
i3  14     5      3.5      3       .1      10      .5      0       30000
i3  20     6      3.5      3       .1      10      .5      0       30000
i3  27     7      3.5      3       .1      10      .5      0       30000
i3  35     8      3.5      3       .1      10      .5      0       30000
s
; EX6.SCO
; dependency of the output signal on the number of iterates
; all other parameters being constant
;
;   start  dur    rstart   rend    x(0)    iter    stereo  xstep   ampl
i3  0      1      3.5      3       .1      12      .5      0       30000
i3  2      1      3.5      3       .1      11      .5      0       30000
i3  4      1      3.5      3       .1      10      .5      0       30000
i3  6      1      3.5      3       .1      9       .5      0       30000
i3  8      1      3.5      3       .1      8       .5      0       30000
i3  10     1      3.5      3       .1      7       .5      0       30000
i3  12     1      3.5      3       .1      6       .5      0       30000
i3  14     1      3.5      3       .1      5       .5      0       30000
i3  16     1      3.5      3       .1      4       .5      0       30000
s
; EX7.SCO
; periodicity introduced by cycling x(0) with a sine wave
;
f 1 0 512 10 1
;
; with constant r
```

```
;   start  dur     rstart   rend     x(0)     iter     stereo   xfreq    ampl
i2  0      4       3.2      3.2      .1       6        .5       2        30000
i2  +      .       3.2      3.2      .1       6        .5       4        30000
i2  +      .       3.2      3.2      .1       6        .5       8        30000
i.  +      .       .        .        .        .        .        16       30000
i0 16 2
s
; with decreasing r (spectrum goes from complex to simple)
i2  0      1       3.2      1        .1       6        .5       32       30000
i.  +      .       .        .        .        .        .        64       .
i.  +      .       .        .        .        .        .        128      .
i.  +      .       .        .        .        .        .        256      .
i.  +      .       .        .        .        .        .        512      .
s
i2  0      .25     3.2      1        .1       6        .5       755      30000
i.  +      .       .        .        .        .        .        713      .
i.  +      .       .        .        .        .        .        663      .
i.  +      .       .        .        .        .        .        626      .
i.  +      .       .        .        .        .        .        591      .
i.  +      .       .        .        .        .        .        558      .
i.  +      .       .        .        .        .        .        527      .
i.  +      .       .        .        .        .        .        498      .
i.  +      .       .        .        .        .        .        470      .
i.  +      .       .        .        .        .        .        444      .
i.  +      .       .        .        .        .        .        419      .
i.  +      .       .        .        .        .        .        400      .
s
; EX8.SCO
; compare the periodicity achieved by instr 1
; with that achieved by instr 2
;
f 1 0 512 10 1
;
i1  0      10      3.2      2        .1       8        .2       .0008    30000
i1  0      10      3.2      2        .21      8        .8       .000803  30000
i2  10     10      3.2      2        .1       8        .2       20       30000
i2  10     10      3.2      2        .21      8        .8       20.01    30000
e
*****************************************************
```

```
; IFS2.orc
; synthesis of sound by functional iterations
; sine map [x=sin(rx)]
;
        sr      =   22050
        kr      =   22050
        ksmps=  1
        nchnls  =   2

        instr 2

; modification of IFS1's instr2
; to study optimal variations in x(0) [-3.14,+3.14], not [-1,1]

asam        init    0
kcnt        init    0
ifrq        init    1/p3

kar         oscili  p4-p5, ifrq, p12, 0         ; envelope for the control parameter
kar         =       kar + p5
ampx        line    p11, p3, p6                 ; envelope for x(0)
aph         oscili  ampx, p9, 1, 0              ; cycling of x(0)
ax          =       (aph+1)/2                   ; update x(0)

label:
kcnt        =       kcnt+1                      ; increment counter
ax          =       sin(kar*ax)                 ; iteration

        if kcnt < p7 kgoto label    ; iterate loop

asam        =       ax                          ; sample
kcnt        =       0                           ; reset counter

asig        =       p10*asam                    ; amp scaling
asig        linen   asig, .05, p3, .05          ; fade in fade out

            outs asig*p8, asig*(1-p8)

            endin
```

```
; SCORES FOR IFS2.ORC
;
;  p4 = initial value for the control parameter (r)
;  p5 = final value for the control parameter (r)
;  p6 = max value for x(0)
;  p7 = iterate number
;  p8 = stereo location (0,1)
;  p9 = increment step for x(0)
;  p10 = global amplitude
;  p11 = min value for x(0)
;  p12 = table function envelope for the control parameter(r)
;
; EX9.SCO
; dynamical control over x(0) (values between p11 and p6)
; envelope on r values
;
f    1   0   2048   10   1
f    2   0   2048   5    .01   250   1   1900   .001
;
; varying x(0), fixed r
;
;                          xstart          xend
i2   0   3   3    3    0    5    .5   50   30000   .2   2
i2   +   .   3    3    0    5    .5   50   30000   .4   2
i2   +   .   3    3    0    5    .5   50   30000   .6   2
i2   +   .   3    3    0    5    .5   50   30000   .8   2
i2   +   .   3    3    0    5    .5   50   30000   1    2
s
;
; fixed x(0), varying r
;
i2   0   3   3    2    .3   5    .5   50   30000   .3   2
i2   +   .   3.2  2    .3   5    .5   50   30000   .3   2
i2   +   .   3.4  2    .3   5    .5   50   30000   .3   2
i2   +   .   3.6  2    .3   5    .5   50   30000   .3   2
i2   +   .   3.8  2    .3   5    .5   50   30000   .3   2
s
; EX10.SCO
; similar with EX9.SCO
```

```
;
i2   0    2   3      3   0   5    .19  50    10000    .2    2
i2   1    .   3      3   0   5    .95  51    10000    .21   2
i2   1.1  .   3      3   0   5    .25  50.1  10000    .23   2
i2   1.9  .   3      3   0   5    .75  50    10000    .38   2
i2   2.1  .   3      3   0   5    .15  49.3  10000    .4    2
i2   3    .   3      3   0   5    .85  50.1  10000    .46   2
i2   3.2  .   3      3   0   5    .15  50    10000    .48   2
i2   4    .   3      3   0   5    .95  49.3  10000    .56   2
s
i2   0    2   3      2   .2  5    .45  50    10000    .2    2
i2   1    .   3.2    2   .2  5    .55  50    10000    .2    2
i2   1.1  .   3.24   2   .2  5    .15  50    10000    .2    2
i2   1.8  .   3.3    2   .2  5    .85  50    10000    .2    2
i2   3.1  .   3.4    2   .2  5    .35  50    10000    .2    2
i2   3.4  .   3.6    2   .2  5    .65  50    10000    .2    2
i2   4    .   3.68   2   .2  5    .55  50    10000    .2    2
i2   4.9  .   3.75   2   .2  5    .45  50    10000    .2    2
e
```

BIBLIOGRAPHICAL REFERENCES

A. Di Scipio “Caos deterministico, composizione e sintesi del suono”, *Proc. of the 9th Colloquium on Musical Informatics*, AIMI/DIST, Genova, 1991

A.Di Scipio & I.Prignano “Functional iteration synthesis. A revitalization of non-standard synthesis”, *Journal of New Music Research*, vol.25, 1996

I.Prignano “Sintesi di eventi sonori complessi per mezzo di Iterazioni Funzionali”, *Proc. of the 11th Colloquium on Musical Informatics*, AIMI/DAMS, Bologna, 1995

(Translated from the Italian by the author)