CHAOTIC SIGNAL SYNTHESIS WITH REAL-TIME CONTROL: SOLVING DIFFERENTIAL EQUATIONS IN PD, MAX/MSP, AND JMAX

Shahrokh Yadegari

Center for Research in Computing and the Arts
University of California, San Diego
http://www.crca.ucsd.edu/∼yadegari

ABSTRACT

Chaotic signals are useful in two different levels in audio synthesis: as sound material or control structure. Patching languages such as Pd, Max/MSP, and jMAX provide easier mechanisms for generating chaotic structures at control level. We can generate deterministic chaotic signals either by finding numerical solutions to differential equations or by using first return maps. While generating the next sample, both of these methods require calculations with the knowledge of the previous sample. Most signal processing environments for computer music, such as Pd, Max/MSP, and jMAX, transfer audio data among their objects by vectors (blocks). In such environments, finding numerical solutions to differential equations or generating signals based on first return maps, will require writing external objects or setting the block-size to 1. Writing external objects can be time consuming and the real-time control of the calculations have to be embedded in the external object, which will require a recompilation for every change to the mechanism. Setting the block-size to 1 can make writing the patch cumbersome and sometimes very confusing. In this paper we shall present the *fexpr∼* object, implemented for Pd, Max/MSP, and jMAX, which can be used for finding numerical solutions to differential equations by simply entering the difference equations as part of the object arguments. The object parameters can then be controlled in real-time using the host patching language. As examples, solutions to Lorenz Equations, Chua's Oscillators, Duffing's equation, and the use of first return maps will be presented using the *fexpr∼* object.

1. INTRODUCTION

Synthesis mechanisms are often separated by synthesis and control parameters. Chaotic signals, which can be used at both of these levels, could be generated by finding numerical solutions to certain differential equations or by iterative use of first return maps. Patching languages such as Pd[1], Max/MSP[2], and jMAX[3] environments offer objects with which one is able to generate chaotic sequences in control level; however, chaotic signal synthesis at sample level can prove to be difficult in such environments which transfer audio among their objects in blocks.

2. SOLUTION TO DIFFERENCE EQUATIONS

A differential equation is a formula which relates an unknown value to its derivative. Differential equations can also be written as a set of equations relating the rate of change of a number of unknowns to their derivatives. Differential equations often do not define an initial condition and to find numerical solutions to them, we need to set initial conditions and integrate over the variable in which the unknowns are changing. For example, the following is a first order differential equation for the variable Y:

$$\dot{Y} = \sin(Y) \tag{1}$$

If Y is changing in time (represented by t), a numerical solution to this equation would take the form of the following difference equation:

$$Y_{n+1} = Y_n + \sin(Y_n)\Delta t \tag{2}$$

Therefore, to find a numerical solution to Equation 1, we pick an initial condition for Y_0 and pick a value for Δt and find the solution function iteratively. In an environment which transfers audio among its objects in blocks, we either have to write an external object which performs the calculation for every sample or set the block-size 1. Both of these solutions have serious drawbacks. Writing external objects requires knowledge of a programming language, such as C, and often involves a learning curve for the uninitiated to the internal workings of the environment. Providing real-time control of the parameters in an external object, (in case of equation 2, Δt), can prove to be time consuming, and recompilation of the object is required for every change to the equation definition or control mechanism of its parameters. Setting the environment's block-size to 1 will make the patch creation difficult and inefficient. Using the *fexpr∼* object we could solve for equation 2 as follows:

```
fexpr~ $y + sin($y) * $x
```

Where `$x` is an input signal which controls the value of Δt in real-time. The 'set' method (explained later) can be used to set the initial condition of Y_0.

3. CHAOTIC SIGNAL SYNTHESIS

Chaotic signals could be synthesized by finding numerical solutions to differential equations of at least third order. One of the most widely used differential equations capable of generating chaotic signals is the Lorenz Equations set:[4]

$$\begin{aligned} \dot{X} &= Pr(Y - X) \\ \dot{Y} &= -XZ + rX - Y \\ \dot{Z} &= XY - bZ \end{aligned} \tag{3}$$

The variables Pr, r, and b are control parameters, and X, Y, and Z are the unknowns for which we find signals as solutions.

Therefore, The difference equations for numerical solutions to the Lorenz equations would take the following form:

$$\begin{aligned} X_{n+1} &= X_n + (Pr(Y_n - X_n))\Delta t \\ Y_{n+1} &= Y_n + (-X_n Z_n + rX_n - Y_n)\Delta t \\ Z_{n+1} &= Z_n + (X_n Y_n - bZ_n)\Delta t \end{aligned} \quad (4)$$

A solution for each of the unknowns in the above equation requires knowledge of the previous values of the other variables at every sample. *Fexpr*$\sim$ also allows for definition of multiple equations in the same instance of the object to accommodate this requirement. The next section discusses the implementation and the syntax used in *fexpr*$\sim$, followed by the examples.

4. EXPR AND FEXPR˜

The *expr* object developed in the original Macintosh version of MAX ("The Patcher") is used for expression evaluation of control streams. The expression syntax for *expr* is very similar to expression syntax of the C programming language.[5, p53] (None of the store, typecasting, nor any of the following operators "`-> . -- ++ ?:`" are supported at this time.) The rules for precedence of operators are also the same as those defined in the C language.

Fexpr$\sim$ is an extension of *expr*. It is best to think of *fexpr*$\sim$ as an *expr* which is evaluated for every sample. *Fexpr*$\sim$ provides a syntax for accessing previous samples of the input streams as well as previous samples of the output streams in the expressions to be evaluated. One block of every input and output streams are buffered. All the *expr* family objects allow for definition of multiple expressions, separated by semicolon, which results in multiple outputs of the same type. This is specially needed when using *fexpr*$\sim$ for finding numerical solutions to differential equations representing second (or higher) order dynamical systems with 2 (or more) variables.

4.1. Equation Definition Syntax

In addition to access to global variables in the host environment, special variables are used for accessing input and output streams. Inlet values are denoted by the following syntax: `$T#` where, `T` specifies the type of inlet and #, the inlet number. Integer, float, and symbol inlets are available to all *expr* objects. For example, `$i1`, specifies the value of the first inlet as integer, `$f3`, the value of the third inlet as float, and `$s2[5]`, the value of the fifth element of an array specified by the value of the second inlet. Signal inputs and outputs in *fexpr*$\sim$ are specified by the `$x#[n]` and `$y#[m]` syntax respectively, where # specifies the signal inlet or outlet number, and `n` and `m` are the indexes for accessing the previous values of the signals. *Fexpr*$\sim$ buffers one block of each of its inputs and outputs; therefore:

$$\begin{aligned} &\text{for } \texttt{\$x\#[n]}, \quad 0 <= n <= -blocksize \\ &\text{for } \texttt{\$y\#[m]}, \quad 0 < m <= -blocksize \end{aligned}$$

A number of shorthand notations are available to make the equation definitions easier to code and read as follows:

$$\texttt{\$x[n]} \rightarrow \texttt{\$x1[n]} \qquad \texttt{\$y[n]} \rightarrow \texttt{\$y1[n]} \quad (5)$$

$$\texttt{\$x\#} \rightarrow \texttt{\$x\#[0]} \qquad \texttt{\$y\#} \rightarrow \texttt{\$y\#[-1]} \quad (6)$$

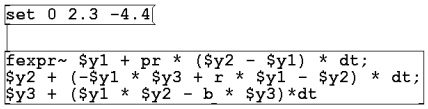
$$\texttt{\$x} \rightarrow \texttt{\$x1[0]} \qquad \texttt{\$y} \rightarrow \texttt{\$y1[-1]} \quad (7)$$

5. EXAMPLES

In this section we shall present a number of examples to show how *fexpr*$\sim$ can be used for chaotic signal synthesis. Along with the examples we shall also include a number of simple real-time control methods for their use in musical contexts.

5.1. Lorenz Equations

The following patch implements difference equations (4). The control parameters pr, r, b, and dt are defined as variables in the environment with the *value* object.



The 'set' method sets the previous values of the 3 output streams and figure (1) is a graph of the proceeding 2048 output values generated by above patch in Pd.

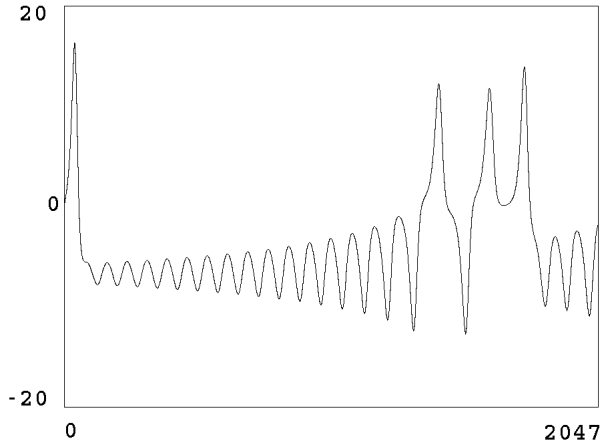


Figure 1: The first 2048 output values of the X signal for Lorenz equations (3) generated with *fexpr*$\sim$ object in Pd with $pr = 10$, $b = 2.66667$, $r = 18$, $dt = 0.01$, and initial values for $X[-1] = 0$, $Y[-1] = 2.3$, and $Z[-1] = -4.4$.

5.2. Chua's Equations

The Chua's Oscillator equations set is one of the popular tools for studying chaotic signals and due to its varied behavior from quasi-periodic to noisy oscillations, we have found it a good musical tool as well. A general dimensionless state equation for a Chua's Oscillator could be written as follows:[6, p xvi]

$$\dot{X} = k\alpha(Y - X - f(X)) \quad (8)$$

$$\dot{Y} = k(X - Y + Z) \quad (9)$$

$$\dot{Z} = k(-\beta Y - \gamma Z) \quad (10)$$

$$f(X) = bX + \frac{1}{2}(a - b)\{|X + 1| - |X - 1|\} \quad (11)$$

The function $f(x)$ implements the odd symmetric characteristic of one of the nonlinear components in the Chua's circuit (namely the nonlinear resistor called Chua's diode). Equation (11) could also be written as follows:

$$f(X) = \begin{cases} bX + (a-b), & X >= 1 \\ aX, & |X| <= 1 \\ bX - (a-b), & X <= -1 \end{cases} \quad (12)$$

The function $f(X)$ represented in (11) can be implemented in *fexpr*$\sim$ using the $abs()$ function; however, to demonstrate the use of the $if()$ function as a general way of handling singularities we have implemented the Chua's oscillator with $f(X)$ characterized in (12) as follows:

```
fexpr~ $y1 + k * (alpha * ($y2 - $y1 - $y4)) * dt;
$y2 + k * ($y1 - $y2 + $y3) * dt;
$y3 + k * (-beta * $y2 - gamma * $y3) * dt;
if ($y1>=1, b*$y1+(a-b), if ($y1>=-1, a*$y1, b*$y1-(a-b)))
```

The outputs of Chua's oscillator are musically useful in two different levels. Solutions obtained by fast integration of the equations (larger values for dt) are quasi-periodic or colored noise. Figure 5.2 shows the first 2048 outputs of the X signal with following variables: $alpha = 15.6$, $beta = 28.58$, $gamma = 0$, $a = -1.14286$, $b = -0.714286$, $k = 1$, $dt = 0.01$, and initial points $X_{-1} = 1.16346$, $Y_{-1} = -0.0972335$, and $Z_{-1} = -0.905656$ (see [6] for the source of these initial values).

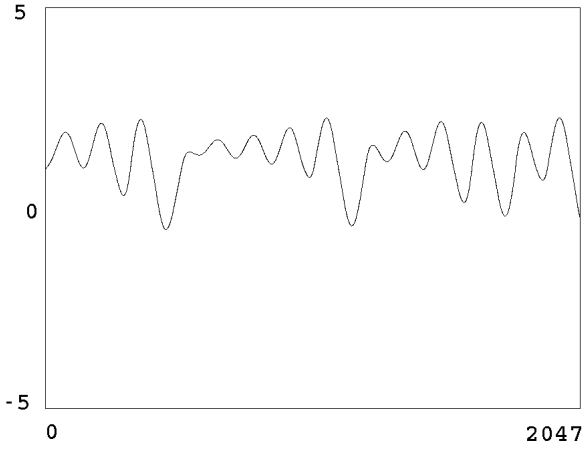


Figure 2: The first 2048 output values of the X signal for Chua's Equations (11) generated with *fexpr*$\sim$ object in Pd with $alpha = 15.6$, $beta = 28.58$, $gamma = 0$, $a = -1.14286$, $b = -0.714286$, $k = 1$, $dt = 0.01$, and initial points $X_{-1} = 1.16346$, $Y_{-1} = -0.0972335$, and $Z_{-1} = -0.905656$.

It is also possible to use the numerical solutions of Chua's equations as control parameters. For example, when scaled correctly, one can use the solutions as frequency values for oscillators. By using smaller values of dt (such as 0.00001) the numerical solutions most often oscillate at a much slower rate and when assigned as frequencies to oscillators the result exhibits certain breathing like quality which mimics musical phrasing. In the above example when the solution of the X signal is multiplied by 100 and used as an oscillator frequency with $dt = 0.01$, the resulting output of the oscillator is colored noise; when $dt = 0.0001$ the result is a quick rhythmical figure while the melodic and rhythmical characteristics are very slowly changing, and when $dt = 0.00001$ the frequency of the oscillator is sweeped slowly but with rather clear phrasing boundaries. When listened over a long period of time, the ever changing glissando phrases can be perceived to have a speaking quality. Thus, as a very simple real-time parameter one can use dt to continuously control the behavior of the synthesis over a spectrum of sounds from colored noise to long slow changing phrases.

5.3. Duffing's Equation

Consider the following Duffing's equation:[7, p 130]

$$\ddot{Y} + k\dot{Y} + \alpha Y + \beta Y^3 = \Gamma \cos(\omega t) \quad (13)$$

by setting:

$$\dot{Y}_n = \frac{Y_n - Y_{n-1}}{\Delta t} \quad (14)$$

$$\ddot{Y}_n = \frac{\dot{Y}_n - \dot{Y}_{n-1}}{\Delta t} \quad (15)$$

we can solve for Y_n as follows:

$$Y_n = \frac{\Delta t^2(\Gamma \cos(\omega t) - \beta * {Y_{n-1}}^3)}{(1 + k * \Delta t)} + \frac{Y_{n-1}(\Delta t\, k - \alpha * \Delta t^2 + 2) - Y_{n-2}}{(1 + k * \Delta t)} \quad (16)$$

The following patch implements the difference equation (16):

```
set y1 0.2 0.1                                          osc~ 92
fexpr~ (dt*dt*(gamma * $x2 - beta * $y*$y*$y) +
$y*(dt*k-alpha*dt*dt+2)-$y[-2])/(1+k*dt)
```

Note the use of the set method "set y1 0.1 0.2" which sets Y_{-1} to 0.1, and Y_{-2} to 0.2. To get deterministic results, it is important to make sure that one sets the phase of the oscillator connected to the second inlet to zero when setting the initial conditions of the *fexpr*$\sim$ instance. Figure 5.3 is a graph of 8192 points generated with the above patch with $dt = 0.01$, $k = 0.3$, $alpha = -1$, $beta = 1$, and $gamma = 0.5$. With the mentioned values, the output goes through an initial transient period which is not included the graph. The chaotic region of the above patch is approximately when $0.5 < gamma < 0.6$.

5.4. First Return Maps

In this section we shall demonstrate how *fexpr*$\sim$ could be used to implement first return maps. One dimensional first return maps can be used for nonlinear signal generation. Let us consider a map represented by $f(x)$ where every sample of the signal is calculated according to the following equation:

$$x_{n+1} = f(x_n) \quad (17)$$

When the map is a defined function, one can do the calculation while generating very sample; however, using a table lookup is often a far more efficient computational process. Even though very

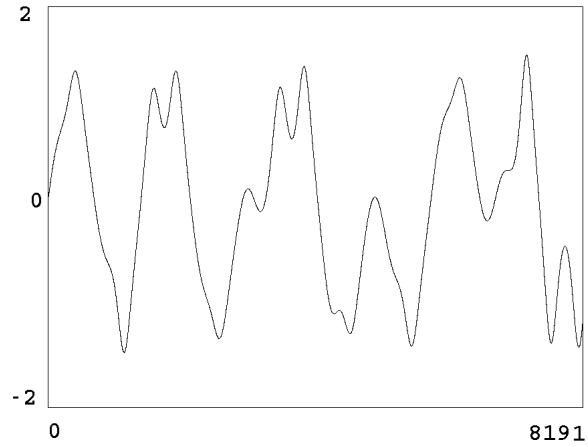


Figure 3: 8192 output values for Duffing's equation (16) generated with *fexpr*$\sim$ object in Pd with $dt = 0.01$, $k = 0.3$, $alpha = -1$, $beta = 1$, $gamma = 0.5$, and ω = 557.76.

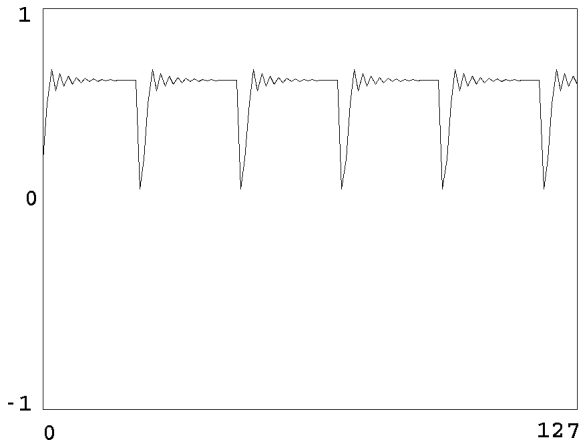


Figure 4: The periodic signal generated by using a 2048 point array as a first return map using equation (18) with setting the elements 1316 and 1317 in the first return map array to 0.1.

small changes to the map could result in considerably different solutions, with careful choices one can use the map as a very simple real-time control of the signal generation process.

Let us consider the iterations using the following simple map: (see [8, p 71])

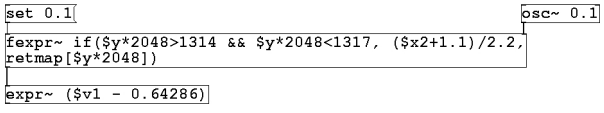
$$x_{n+1} = 4\mu x_n(1 - x_n), \quad x \in [0, 1]. \tag{18}$$

If we build the map of the equation (18) in a 2048 point array called "retmap", the signal synthesis patch would be simply:

```
fexpr~ retmap[$y * 2048]
```

With $\mu = 0.7$, equation (18) has an unstable fixed point at zero and an stable fixed point at 0.64286. In a 2048 point table the fixed point falls between the 1316th and 1317th elements of the table. If we set these two elements to any other number than what function (18) would specify, the iterative process will synthesize a periodic signal. Figure 5.4 shows the result of this iteration if we set "retmap[1316]=0.1", "retmap[1317]=0.1".

We could use an oscillator to reset the value of the generated output as the iteration gets close to the fixed point. The following patch produces a signal with varying pitch, rhythm, and dynamics. Note the scaling of the oscillator input as `($x2+1.1)/2.2` which is to make sure that the process does not generate an out of range sample, as well as not arrive at the zero fixed point. Of course one can make the limit values defining the vicinity of the fixed point a real-time controlled parameter as well.



6. SUMMARY

Using differential equations and first return maps are among popular methods for synthesizing chaotic signals. Implementing such methods in environments that transfer audio among their objects in blocks can be time consuming and difficult. A new object, called *fexpr*$\sim$, was introduced for Pd, Max/MSP, and jMAX, which facilitates solving difference equations and using first return maps while providing a simple mechanism for real-time parameter control.

7. ACKNOWLEDGMENTS

It is a pleasure to acknowledge the input and helpful suggestions of Miller Puckette throughout all development stages of the *expr* objects.

8. REFERENCES

[1] M. Puckette, "Pure data: another integrated computer music environment," in *Proceedings, International Computer Music Conference*. 1996, pp. 269–272, San Francisco: ICMA.

[2] D. Zicarelli, "An extensible real-time signal processing environment for max," in *Proceedings, International Computer Music Conference*. 1998, San Francisco: ICMA.

[3] F. Dechelle, R. Borghesi, E. De Cecco, M. Maggi, B. Rovan, and N. Schnell, "jmax: a new java-based editing and control system for real-time musical applications," in *Proceedings, International Computer Music Conference*. 1998, San Francisco: ICMA.

[4] Edward N. Lorenz, "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20, March 1963.

[5] Brian W. Kerninghan and Dennis M. Ritchie, *The C programming language*, Prentice Hall Press, 1988.

[6] *Chua's Circuit: A paradigm for Chaos*, World Scientific, New Jersy, 1993.

[7] D. W. Jordan and P. Smith, *Nonlinear Ordinary Differential Equations*, Clarendon Press, New York, second edition, 1987.

[8] P. Bergé, Y. Pomeau, and C. Vidal, *Order Within Chaos: Towards a Deterministic Approach to Turbulence*, Wiley-Interscience, New York, 1984.