# 𝔞-chaos, v 0.1
# a set of chaos macros for Loomer's Architect

Alberto Zin

January 30, 2019

## 1 Introduction

𝔞-chaos[1] is a set of chaos macros for Loomer's Architect implementing chaos- and fractal-related algorithms. "𝔞-"stands for "Architect" , a midi manipulation tool from Loomer (https://www.loomer.co.uk/products.htm).

The macros can be used in the generation of chaotic series, for sonification purposes, for generating LFO's, to create powerful modulators or for any other purpose. The macros were purposely implemented using the graph part of Architect (and not by simpler LUA scripting) in order to allow me to better learn the Architect syntax. The use of a chaos algorithm is particularly useful when a certain pattern has to be proposed and varied in an unusual/unexpected way, differently from a standard random (gaussian white noise) representation, but more like an "evolution" which is typical of chaotic systems. In case the output of the algorithm is "bounded" (on 1, 2 or 3D) the capability of mapping in a certain way (linear, etc) the data to sound parameters (like note height, LFO's, velocity, patches, cutoff, rythm ...) is what makes this type of algorithms a deep source of inspiration.

## 2 Install

𝔞-chaos is a set of .frag files for Architect. Place the folder a-chaos in `~/.local/share/Loomer/Architect/Fragments` folder (in Linux) or in the Architect Fragments folder of your system. Then you can access the macros by right clicking in the graph and looking in the "External" menu.

## 3 Use

Instantiate one 𝔞-chaos macro in the Architect graph as described above. All the macros have a common external interface: a RESET inlet, an INIT inlet, few constants inlets (the number depending on the algorithm) and a STEP inlet. The algorithm is self-initialized:

[1]This collection is unrelated to "A-chaos" lib for Max/MSP http://s373.net/code/A-Chaos-Lib/A-Chaos.html

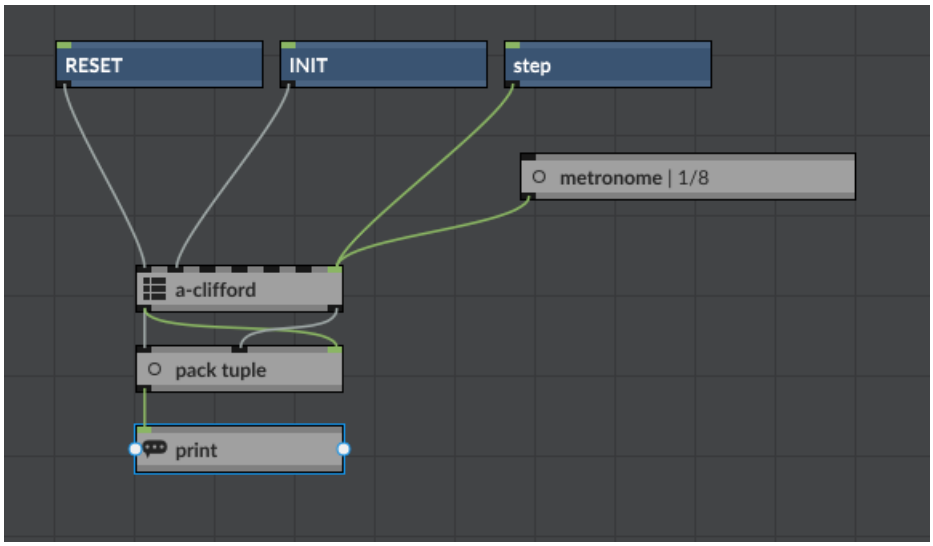


Figure 1: Common algorithm interface

when the macro is instatiated, the constants are already initialized and the RESET and INIT inputs do not need to be present or pressed. The constants inputs are necessary in case the default values need to be changed, for experimentation with the algorithm. The INIT input could be useful to restore the default algorithm values. The STEP inlet is typically connected to a metronome, otherwise it can be activated with single click to a signal object to output a single outcome of the algorithm. The RESET inlet instead restarts the iteration sequence from zero (where it makes sense). In order to facilitate the usage of the algorithms, a mapping function ("`a-linmap`") is part of the macro set. This macro can be used to map the output range of the algorithm data to the range needed by the final use (for example midi note or midi CC to [0, 127]). Note that the "pack tuple" object, used for printing the data on the console, is banged by the first (left) outlet of each algorithm. The library has been built and verified against Architect beta version 0.9.9

# 4 Implemented algorithms

A nice and visual reference for the algorithms[2] implemented in `a`-chaos can be found on the page `http://paulbourke.net/fractals/`. Currently the following algorithms are implemented in the library:

- `a-lorenz` (3D))
- `a-henon` (2D)
- `a-hopalong` (2D)
- `a-dejong` (2D)

[2]The correctness of implemetation in Architect is verified against an implementation in Octave (Matlab).

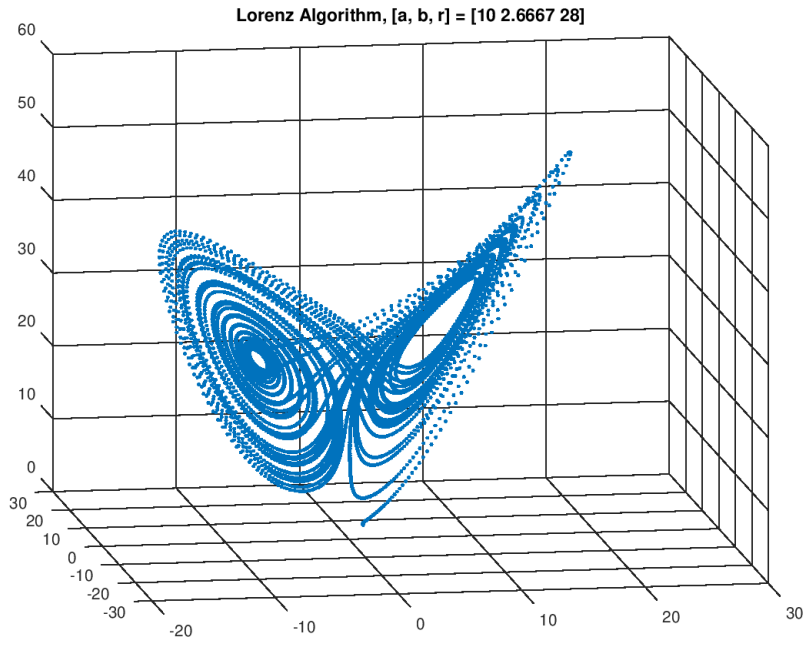


Figure 2: 3D representation of Lorenz algorithm

- `a-clifford` (2D)
- `a-duffing` (2D)
- `a-logistic` (1D) - courtesy of Charles Turner

More algorithms to come.

### 4.1 Lorenz Attractor

The Lorenz attractor is implemented by the following system of differential equations:

$$\dot{x} = a(y - x) \tag{1}$$
$$\dot{y} = -xz + rx - y \tag{2}$$
$$\dot{z} = xy - bz \tag{3}$$

with the constants being $a = 10.0$, $r = 28$, $b = 8/3 = 2.666667$. The algorithms provides 3 outputs (the coordinates x(t), y(t) and z(t)). The time series and the 3D plot is shown in Figure 2 and Figure 3. These data can be mapped to some audio characteristics of a signal as exlained in the introduction. The graph implementation in Architect is shown in Figure 4.

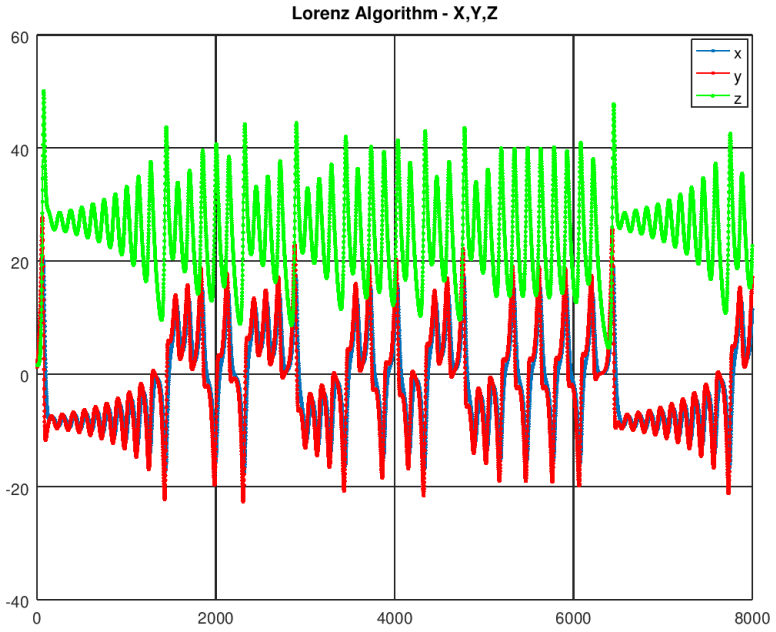


Figure 3: 3D representation of Lorenz algorithm

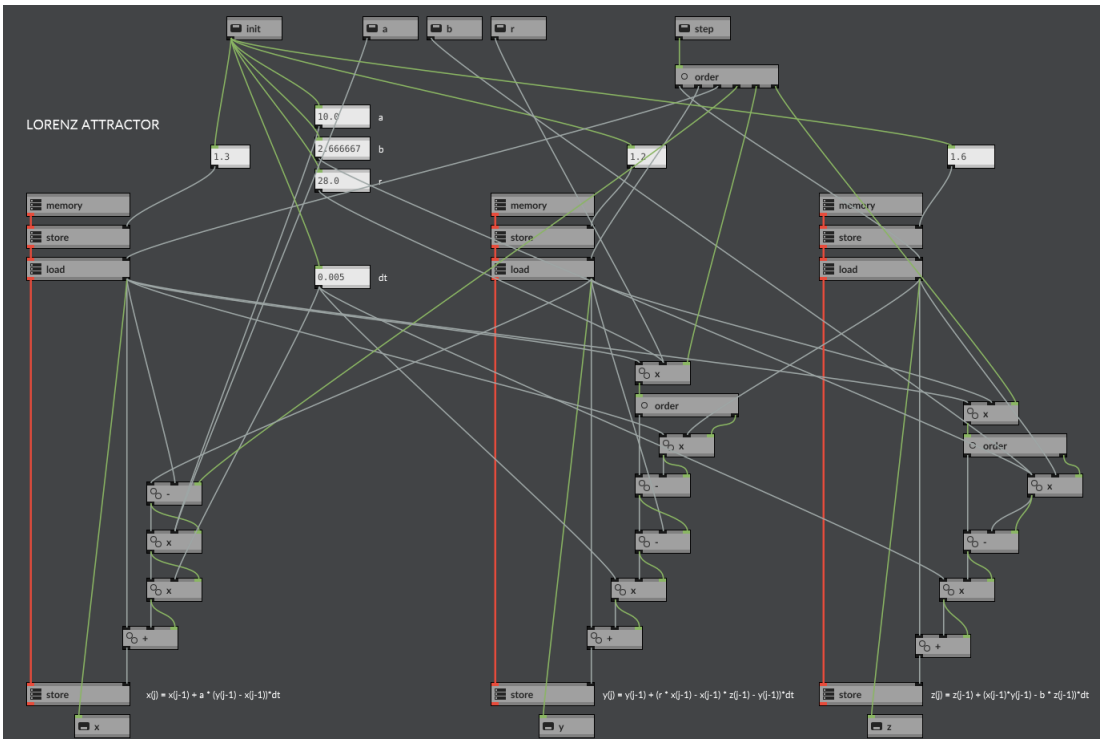


Figure 4: Lorenz patch implementation

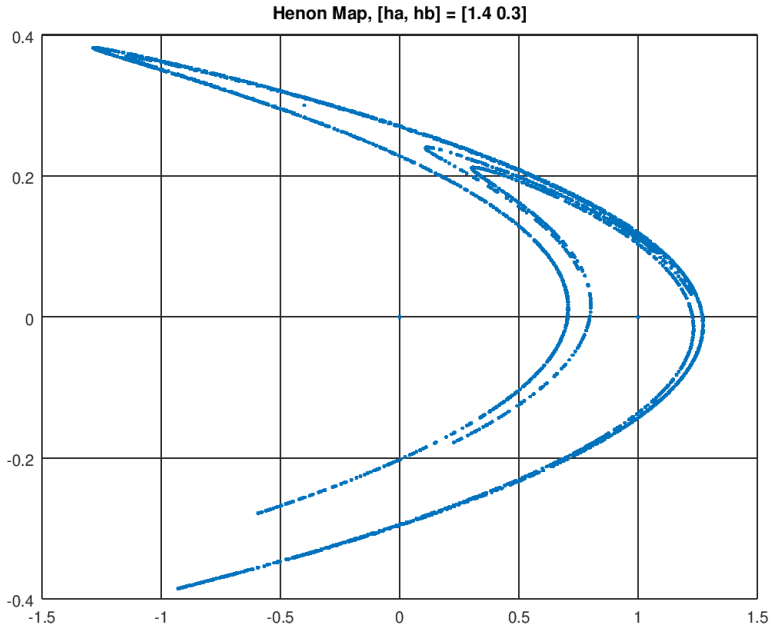


Figure 5: 4000 iteration of the Henon algorithm

### 4.2 Henon Map

The standard Henon map is implemented by the following set of difference equations:

$$x_{n+1} = 1 + y_n - ax_n^2 \tag{4}$$

$$y_{n+1} = bx_n \tag{5}$$

with the default constants $a = 1.4$, $b = 0.3$. Using those constants the algorithm is bounded in X between -1.5 and 1.5 and in Y between -0.4 and 0.4. The graph implementation in Architect is provided in Figure 5. The first 4000 outcomes of the default algorithm are plotted in Figure 6.

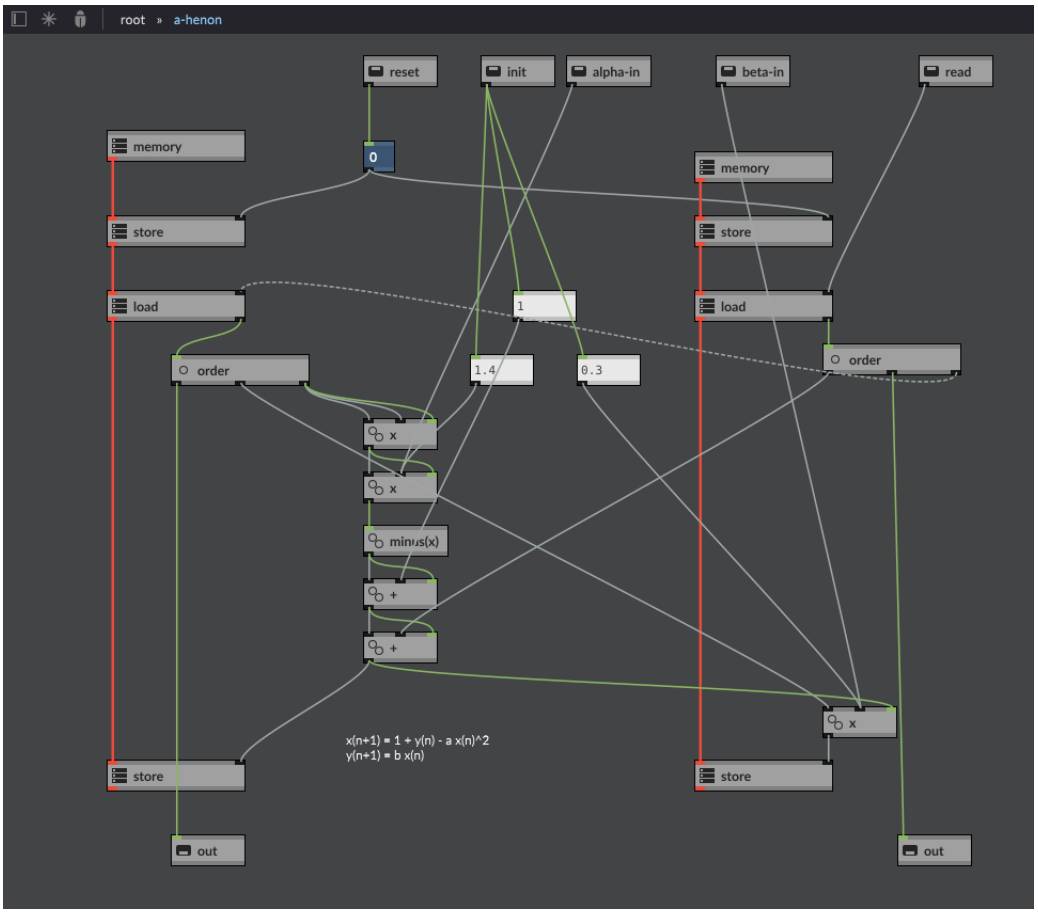


Figure 6: Henon algorithm implementation

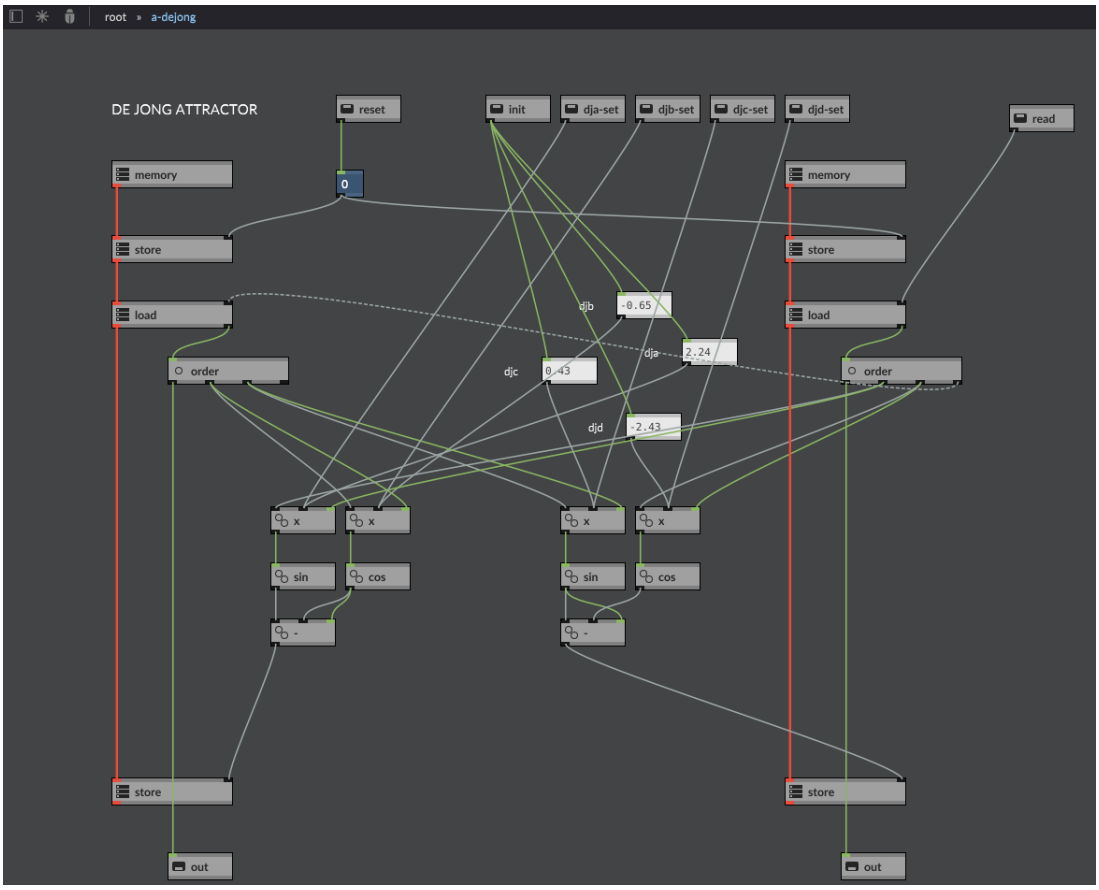


Figure 7: DeJong Algorithm Implementation

### 4.3 De Jong Attractors

The De Jong attractor family is defined by the following iterative equations:

$$x_{n+1} = \sin(ay_n) - \cos(bx_n) \quad (6)$$

$$y_{n+1} = \sin(cx_n) - \cos(dy_n) \quad (7)$$

The default constants are $dja = 2.24$, $djb = -0.65$, $djc = 0.43$, $djd = -2.43$. The graph implementation in Architect is provided in Figure 7. The first 4000 outcomes of the default agorithm are plotted in Figure 8. In general the output is bounded in the range [-2, 2] for both X and Y.

### 4.4 Clifford Attractors

The clifford algorithm is implemented as the following equations (See "The Pattern Book, Fractals, Art, and Nature" by Clifford Pickover, chapter titled "Swirl"):

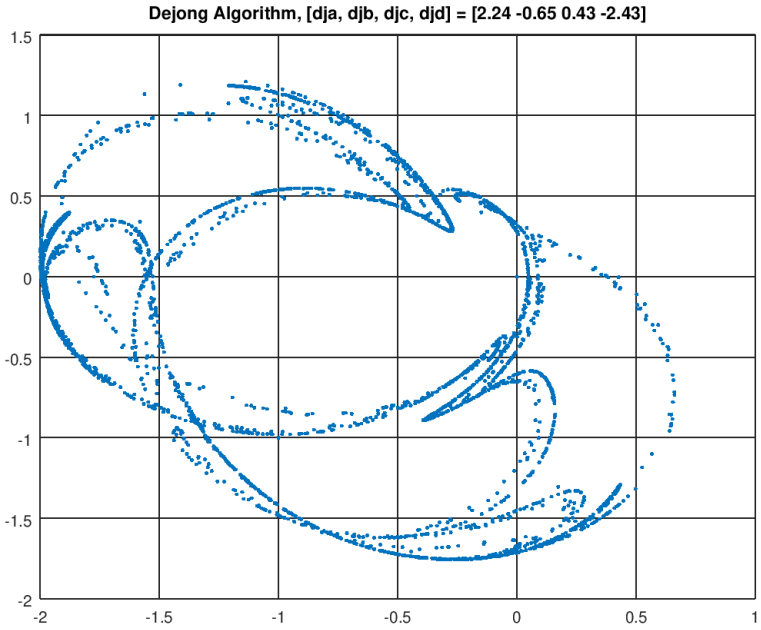


Figure 8: 4000 iteration of the De Jong algorithm

$$x_{n+1} = \sin(ay_n) + c\cos(ax_n) \tag{8}$$
$$y_{n+1} = \sin(bx_n) + d\cos(by_n) \tag{9}$$

The default constants are $a = 1.4$, $b = 1.6$, $c = 1.0$, $d = 0.7$. The graph implementation in Architect is provided in Figure 10.

### 4.5 Duffing Attractors

The duffing attractor is represented by the following equation:

$$\ddot{x} + \delta\dot{x} + \alpha x + \beta x^3 = \gamma\cos(\omega t) \tag{10}$$

The equation describes the motion of a damped oscillator with a more complex potential than in simple harmonic motion (which corresponds to the case $\beta = \delta = 0$)

The graph implementation in Architect is provided in Figure 12.

### 4.6 Logistic Equation

$$f(x) = Rx(1 - x) \tag{11}$$

The typical use is a series of the type:

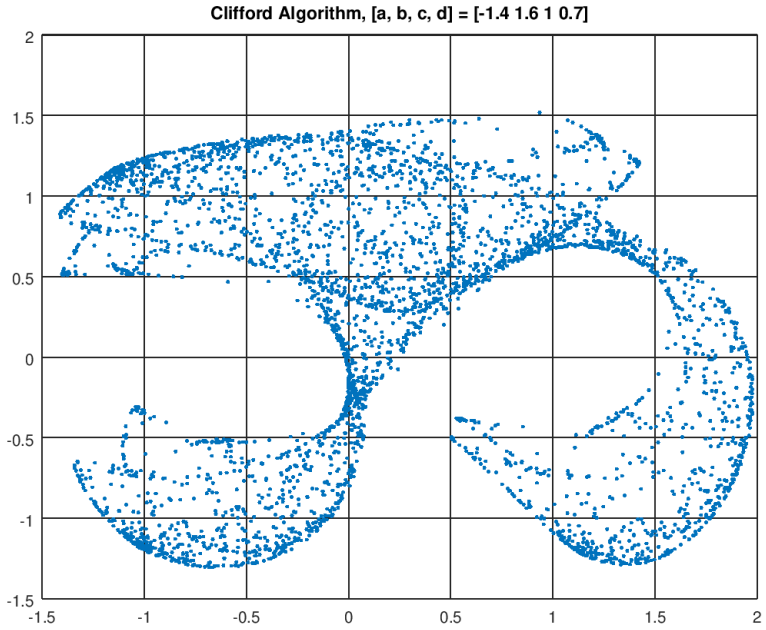


Figure 9: 4000 iteration of the Clifford algorithm

$$A_{n+1} = \lambda A_n(1 - A_n) \tag{12}$$

This series behaves[3] in one of the following ways depending on the value of R, the initial conditions don't matter (within reason).

- Extinction (Uninteresting fixed point): If the growth rate R is less than 1 the system "dies", $A_n -> 0$.

- Fixed Point: The series tends to a single value. How it reaches this value is not important but generally it oscillates about the fixed point but unlike a mass spring system, the series generally tends to rapidly approach fixed points. In the bifurcation diagram below the system can be seen to tend to fixed points for $1 < R < 3$.

- Periodic: The series jumps between two or more discrete states. In the bifurcation diagram below it can be seen that the system alternates between 2 states after R = 3. After about 3.44948 (1 + sqrt(6)) the system alternates between 4 states. Notice the system jumps between these states, it does not pass through intermediary values. The number of states steadily increases in a process called period doubling as R increases. For example at 3.5441 until 3.5644 there are 8 states. Between 3.5644 and 3.5688 there are 16 states.

[3]From P. Bourke website.

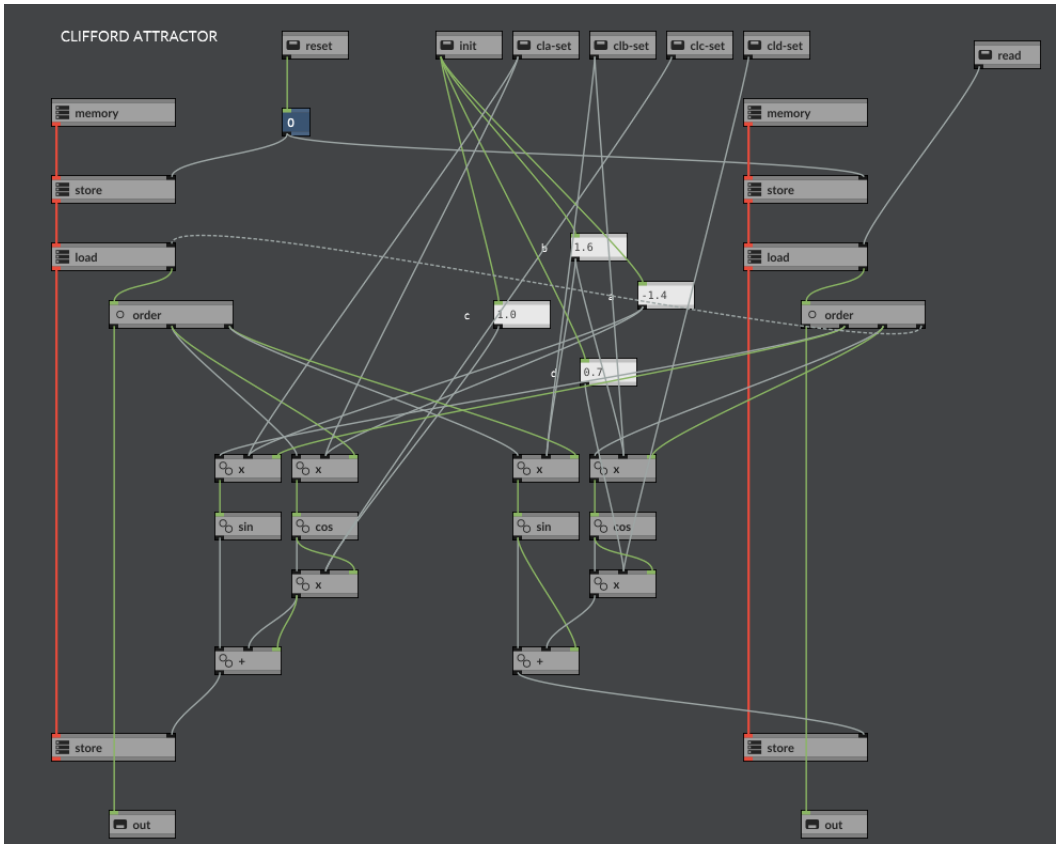


Figure 10: Clifford Patch Implementation

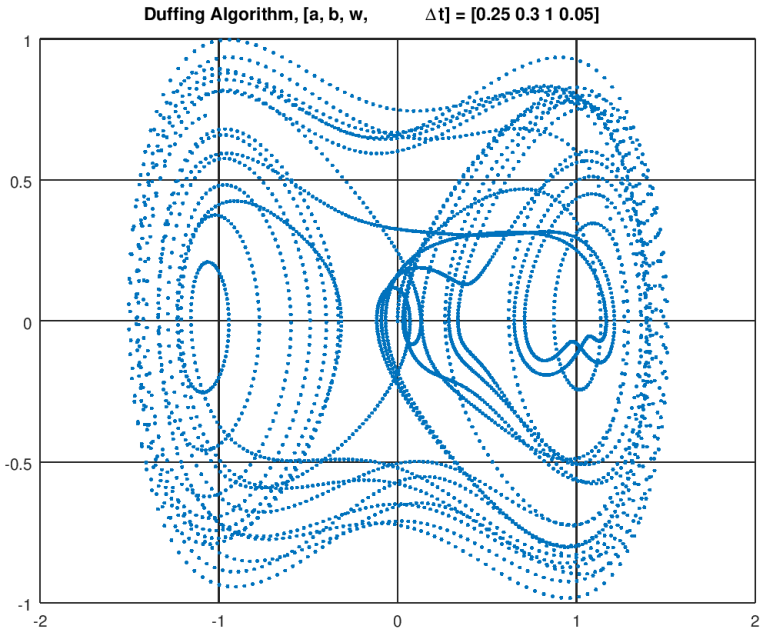


Figure 11: 4000 iteration of the Duffing algorithm

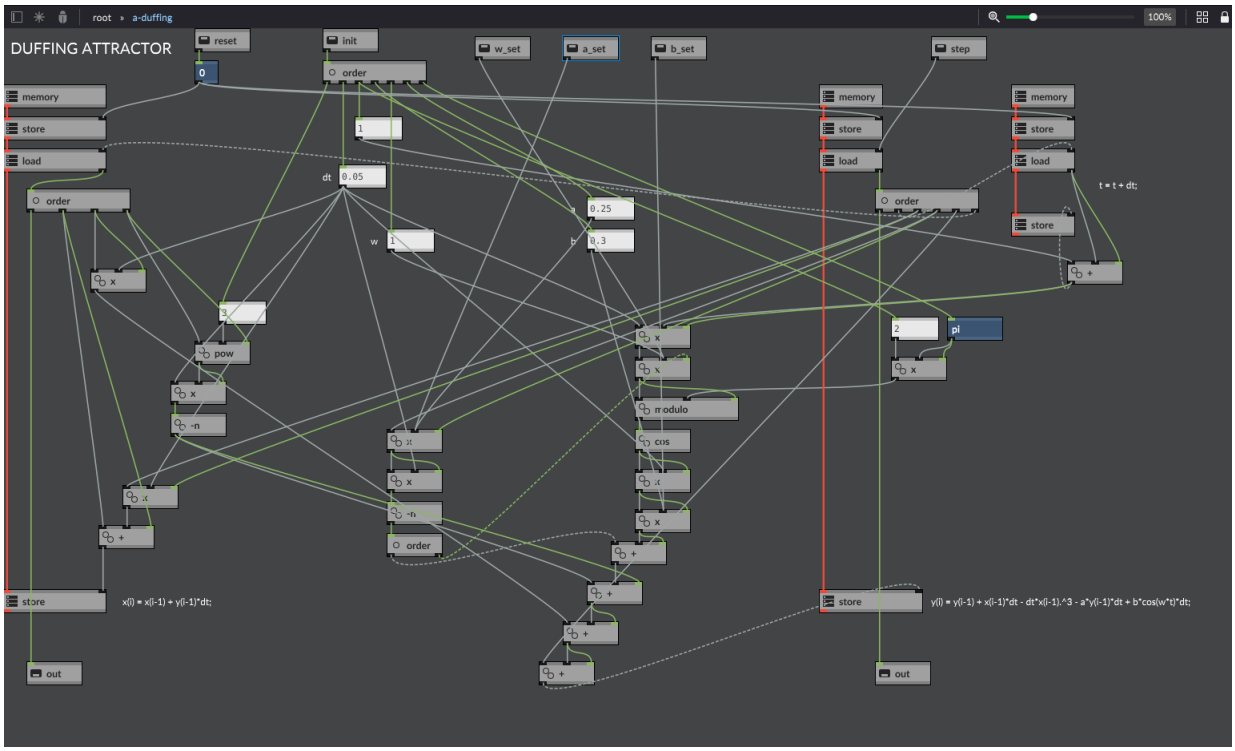


Figure 12: Duffing Algorithm Implementation

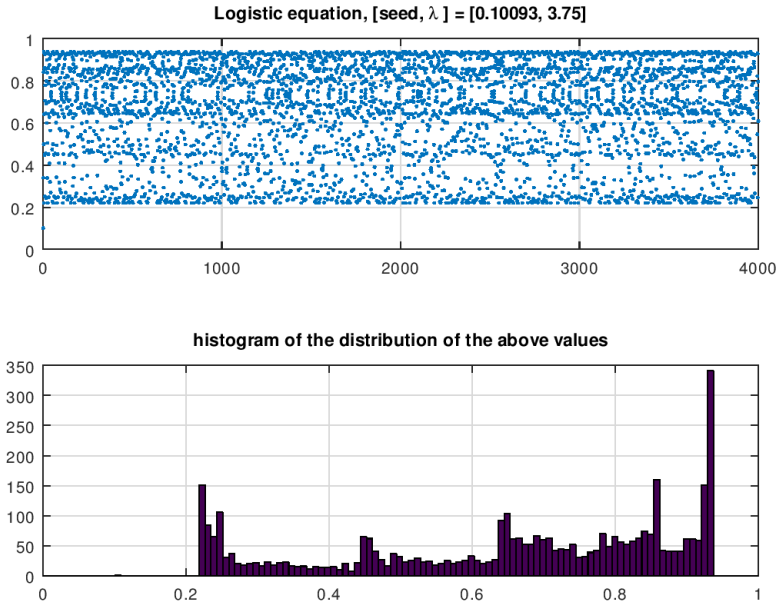


Figure 13: First 4000 iteration of the logistic equation (top), distribution of he data (bottom)

- Chaotic: In this state the system can evaluate to any position at all with no apparent order. In the bifurcation diagram below, the system undergoes increasingly frequent period doubling until it enters the chaotic regime at about 3.56994. Below R = 4 the states are bound between (0,1), above 4 the system can evaluate to (0,infinity). Amongst this chaos a 3-period surprisingly appears between about 3.8284

$$(1 + \sqrt{8}) < R < 3.8415$$

.

The graph implementation in Architect is provided in Figure 14.

## 5 Acknowledgements

Thanks to Colin Barry for creating Architect, the long awaited and powerful MIDI manipulation tool. Thanks to Charles Turner for his logistic map implementation.

## 6 Warranty & License

No warranty at all. The patches are provided "as-is" without any express or implied warranty. In no event shall the author be held liable for any damages arising for use of this patch. ɑ-Chaos is distributed under MIT Licence.

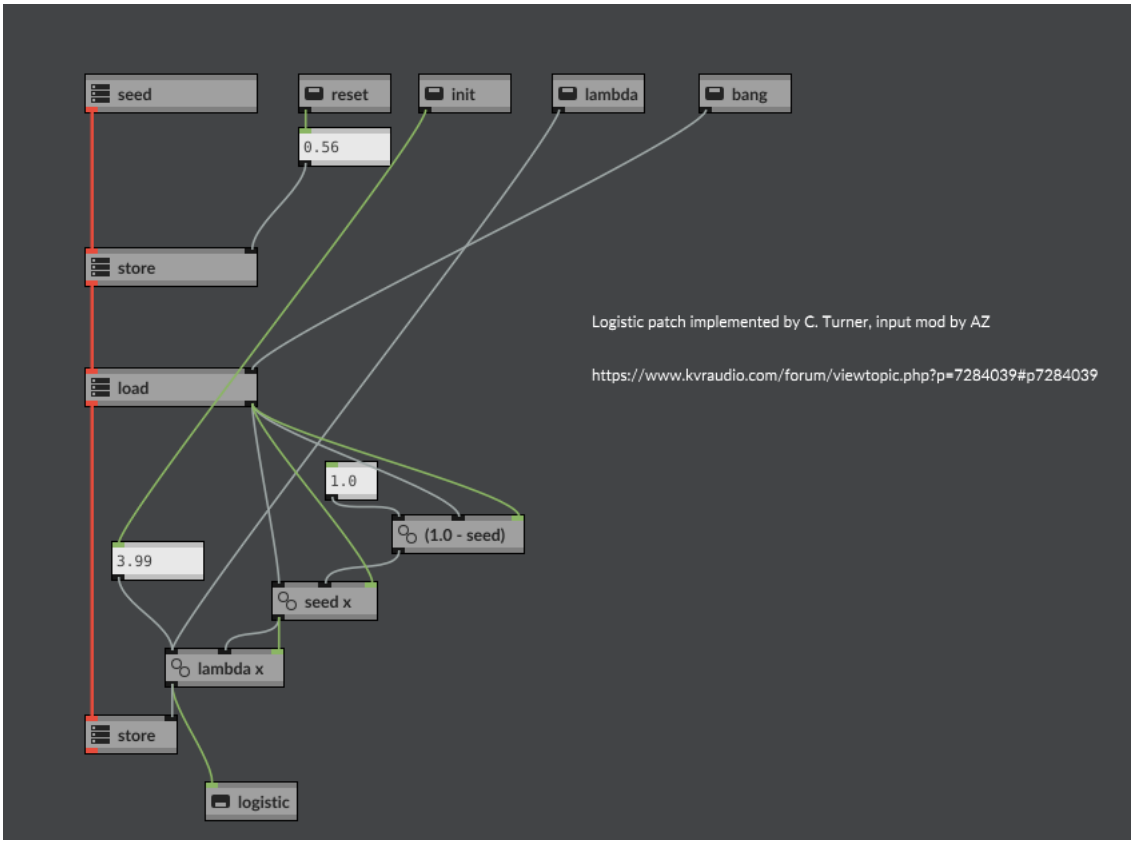


Figure 14: Logistic Algorithm Implementation (courtesy of C. Turner)