Lyapunov Space of Coupled FM Oscillators

Claude Heiland-Allen
claude@mathr.co.uk

Abstract

Consider two coupled oscillators, each modulating the other's frequency. This system is governed by four parameters: the base frequency and modulation index for each oscillator. For some parameter values the system becomes unstable. The Lyapunov exponent is used to measure the instability. Images of the parameter space are generated, with the number crunching implemented on graphics hardware using OpenGL. The mouse position over the displayed image is linked to realtime audio output, creating an audio-visual browser for the 4D parameter space.

Keywords

chaos, DSP, GPU

1 Introduction

Soft Rock EP [ClaudiusMaximus, 2005] and Soft Rock DVD [ClaudiusMaximus, 2006] explored the transitions between order and chaos in coupled FM oscillators. A more recent continuation of this project is to make a map of the parameter space of coupled FM oscillators on a perceptually relevant level and use it in live performance, choosing parameters on the basis of desired sound character.

A bifurcation diagram produced by an analogue Moog synthesizer [Slater, 1998] and images of Lyapunov fractals [Dewdney, 1991] were inspiration to apply the latter technique to the parameter space of coupled FM oscillators in the digital realm.

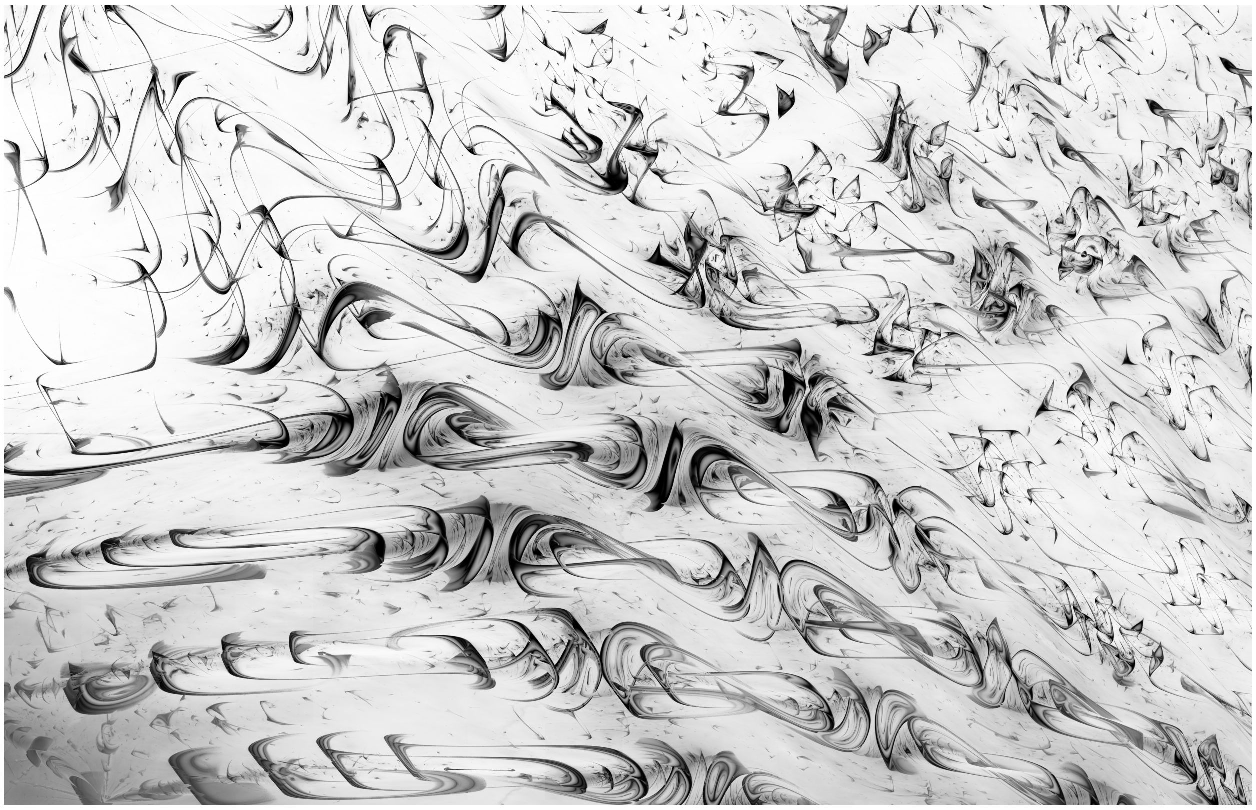


Figure 1: Example output.

2 Formulation

2.1 Coupled FM Oscillators

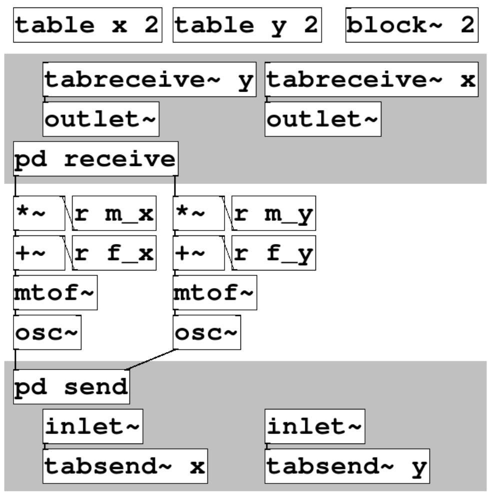


Figure 2: Coupled FM oscillators in Pure-data.

Consider the two coupled oscillators in Figure 2. Pure-data's model of interconnected components each with their own internal state maps poorly to GPU architecture. Considering the whole system as one and flattening the internal state into a single phase space vector leads to the following formulation as a mutual recurrence relation:

$$\begin{aligned} x_{n+1} &= \%(x_n + \mathrm{I}(f_x + m_x \cos(2\pi y_{n-d}))) \\ y_{n+1} &= \%(y_n + \mathrm{I}(f_y + m_y \cos(2\pi x_{n-d}))) \end{aligned} \tag{1}$$

where

$$\%(t) = t - \lfloor t \rfloor, \quad \mathrm{I}(t) = \frac{440}{\mathrm{SR}} 2^{\frac{t-69}{12}}$$

Here x_n, y_n is the phase of each oscillator at time step n, d is a delay measured in samples, f_x, f_y is the base frequency of each oscillator as a MIDI note number, and m_x, m_y is the modulation index of each oscillator as a MIDI note number. $\%(t)$ performs wrapping into $[0, 1)$, with $\lfloor t \rfloor$ being the flooring operation (the largest integer not greater than t).

The four-dimensional parameter space vector will be written

$$a = (f_x, f_y, m_x, m_y)$$

and the $(2d+2)$-dimensional phase space vector

$$z = (x_n, y_n, x_{n-1}, y_{n-1}, \ldots, x_{n-d}, y_{n-d})$$

with sample rate $\mathrm{SR} = 48000$. For reasons explained in Section 5.2, $d = 1$ will be fixed.

2.2 Lyapunov Exponents

Lyapunov exponents can be used to measure the stability (or otherwise) of a dynamical system. A good introduction is found in Chapter 4.3 *Lyapunov Exponent* [Elert, 2007]. The definition is covered in Chapter 13.7 *Liapounov exponents and entropies* [Falconer, 2003] which also relates it to measures of fractal dimension.

The Lyapunov exponent λ measures divergence in phase space:

$$|z_1(t) - z_0(t)| \approx e^{\lambda t} |z_1(0) - z_0(0)|$$

$$\lambda = \lim_{\substack{t \to \infty \\ z_1(0) \to z_0(0)}} \frac{1}{t} \log \frac{|z_1(t) - z_0(t)|}{|z_1(0) - z_0(0)|} \tag{2}$$

An attracting orbit has $\lambda < 0$ and a divergent (chaotic) orbit has $\lambda > 0$.

A modified norm is required to take into account the wrapping of phase into $[0, 1)$:

$$|z|_{\%} = \sqrt{\sum_i (\min(\%(z_i), 1 - \%(z_i)))^2}$$

For example the distance between 0.1 and 0.9 is properly 0.2 (not 0.8) because 0.1 can be phase-unwrapped to 1.1.

2.3 Viewing Planes

An image is 2D, which requires choosing a subset of the 4D parameter space to visualize. Two particular planes were chosen:

$$\begin{aligned} A_+(a_0, r_0) &= a_0 + r_0 \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \\ A_-(a_0, r_0) &= a_0 + r_0 \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \end{aligned} \tag{3}$$

where (u, v) is the coordinates of the pixel, a_0 is the centre of the view, and r_0 is the radius of the view.

These planes were chosen because they are simple, while still being flexible enough to explore the whole 4D space. The A_+ plane varies both oscillators in the same direction, while the A_- plane varies each oscillator in opposite directions. To center on a particular target point (f_x, f_y, m_x, m_y) one might use the A_+ plane to center on the midpoint

$$\left(\frac{f_x + f_y}{2}, \frac{f_x + f_y}{2}, \frac{m_x + m_y}{2}, \frac{m_x + m_y}{2}\right)$$

and then switch to the A_- plane to break the (x, y) symmetry.

3 Implementation

The implementation uses OpenGL [Segal, 2013] and OpenGL Shading Language [Kessenich, 2013] for computation and graphical rendering, GLUT [Kilgard, 1996] for windowing and input event handling, and JACK [Davis, 2013] for audio output.

3.1 Introduction to Modern OpenGL

Modern OpenGL has a programmable shader pipeline. Vertex attributes are read from vertex buffers and processed by vertex shaders. The outputs of the vertex shader (called varyings) are further manipulated by an optional geometry shader stage. Geometry shaders can output a different vertex count to their input count, whereas vertex shaders are one-in one-out. The result of the geometry shader can be captured into another vertex buffer using transform feedback. Following the geometry shader the primitives (points or triangles) are rasterized, and varyings interpolated across each primitive. Finally a fragment shader takes these values and computes the colour at that pixel. The output of a fragment shader can be captured by attaching a texture to a framebuffer.

3.2 Computation Overview

To render an image a texture is first filled with (u, v) coordinates using a framebuffer object and a fragment shader. This texture is copied to a vertex buffer object, interleaved with an initial phase space vector $z = (0, 0, 0, 0)$ and Lyapunov exponent statistics vector $l = (0, 0, 0, 0)$ for each point.

Using a vertex shader, a is calculated from (u, v) using Equation 3, and then a rough estimate of the Lyapunov exponent is computed using Equation 2 by perturbing $z_1(0) = z_0(0) + \delta$ with δ small and performing $t = 256$ iterations of Equation 1. The first few repetitions are discarded, along with those resulting in $-\infty$, and the rough λ estimates are accumulated in l.

Between each repetition the working set is compacted using a geometry shader. Points whose mean Lyapunov exponent estimate changed very little during the previous step are plotted and removed from the working set. The other points are kept to be refined further, directing the computational effort on the points that need it most: those slow to converge.

To ensure user interface responsiveness, the computation is amortized over several frames. The target frame period is divided by the measured time for one repetition to compute how many repetitions to perform that frame. The repetitions-per-frame increases as the working set becomes smaller.

3.3 Noise Increases Stability

At the end of each repetition z_1 is kept instead of z_0. This effectively adds a small amount of noise, counter-intuitively increasing stability. Noise allows more of the phase space to be explored, and makes it more likely for the perturbed orbit to reach an attracting part of the phase space.

3.4 Dither Increases Quality

To reduce grid sampling artifacts, (u, v) is perturbed within the bounds of its corresponding pixel before calculating the a parameter vector for each repetition.

4 Results

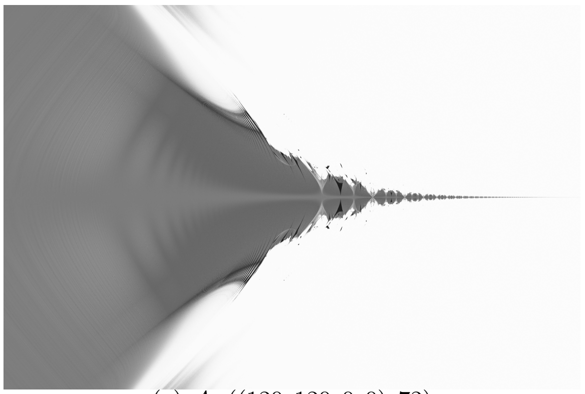
4.1 Examples

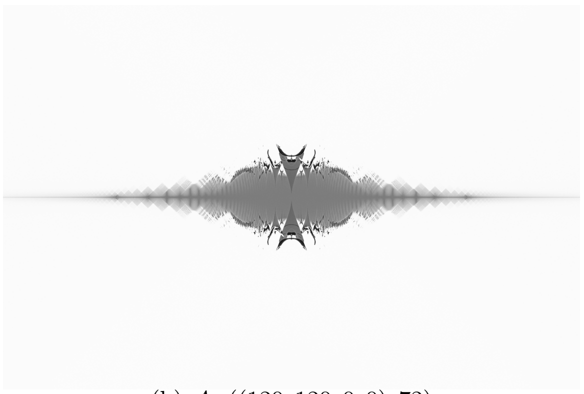
Figure 3(a) shows the initial view on starting the interactive browser. Low frequencies to the left are stable even at high modulation index away from the central axis. High frequencies to the right become chaotic at progressively lower modulation index. (b) shows the A_- plane at the same location. (c) shows bands alternating between stability and chaos. The bands become distorted and collapse as the modulation index and frequency increase. (d) shows its A_- plane, bands become rings. When the frequency is greatly increased, the shapes become more intricate. (e) exhibits spirals of stability, with similar spirals in the A_- plane in (f).

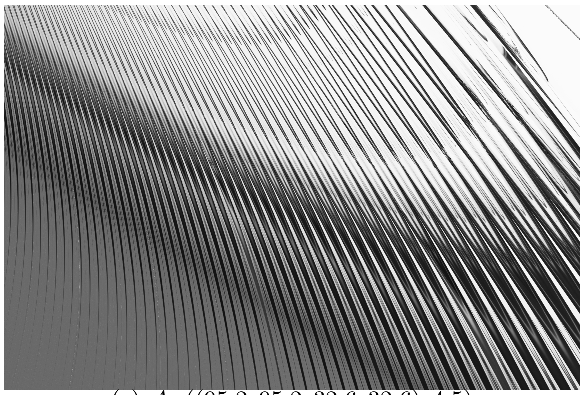
When $f_x = f_y$ and $m_x = m_y$ the A_+ plane has mirror symmetry about its horizontal axis, and the A_- plane has two-fold rotational symmetry about its centre. Breaking the symmetry and setting $f_x \neq f_y$ or $m_x \neq m_y$ leads to diverse forms. In particular Figure 3(h) has shapes that resemble those of Lyapunov space images of the logistic map.

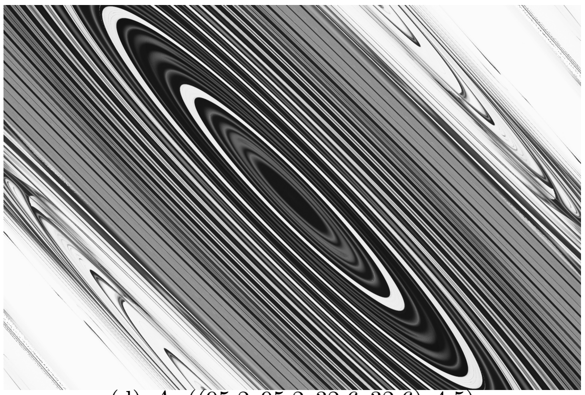
4.2 Interactive Explorer

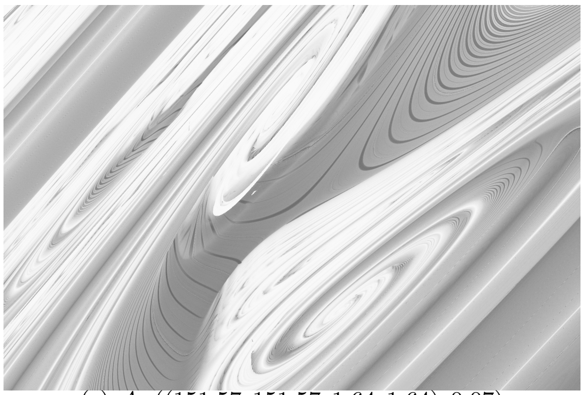
The implementation is an interactive audio-visual explorer for the parameter space of cou-

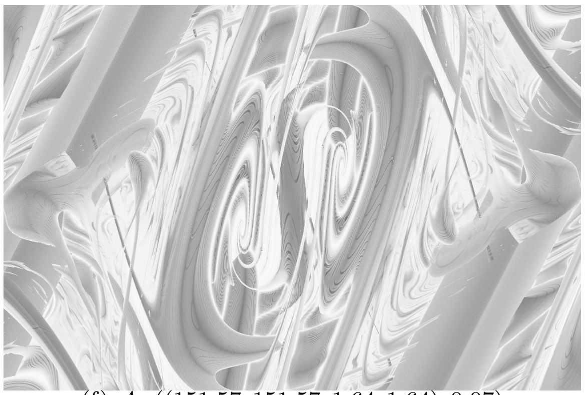

(a) $A_+((120, 120, 0, 0), 72)$

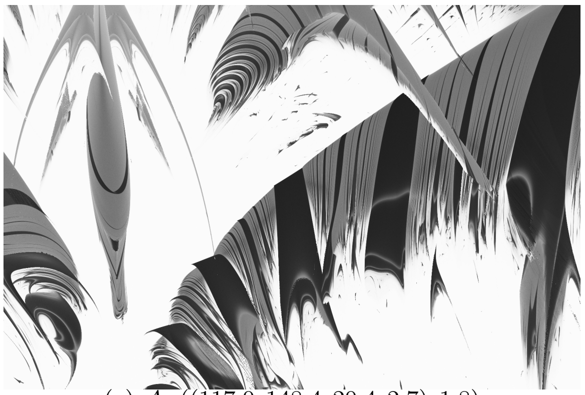

(b) $A_-((120, 120, 0, 0), 72)$


(c) $A_+((95.2, 95.2, 32.6, 32.6), 4.5)$


(d) $A_-((95.2, 95.2, 32.6, 32.6), 4.5)$


(e) $A_+((151.57, 151.57, 1.64, 1.64), 0.07)$


(f) $A_-((151.57, 151.57, 1.64, 1.64), 0.07)$


(g) $A_-((117.0, 148.4, 20.4, 2.7), 1.8)$

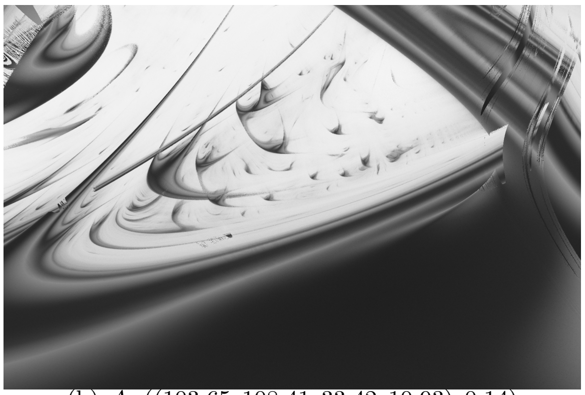

(h) $A_+((103.65, 108.41, 33.42, 10.93), 0.14)$

Figure 3: Example images. Darker shades are stable, lighter shades chaotic.

pled FM oscillators. Clicking with the mouse zooms the view about the clicked point. The left button (or scroll up) zooms in, the right button (or scroll down) zooms out, the middle button centers the view on the target point. Pressing the TAB key toggles between the A_+ and A_- planes in Equation 3, and F11 toggles full screen operation.

While the GPU simulates and analyses one oscillator pair per pixel, the CPU simulates one oscillator pair with a determined from the pixel under the mouse pointer. The image acts as a map, a reference frame for chosing parameters to audition by moving the mouse.

5 Conclusions

5.1 Original Intent

Earlier experiments used one Pure-data batch mode instance per CPU core each sending analysis data to a realtime Pure-data instance. The analysis used various methods (including FFT for spectral statistics and the sigmund external for pitch tracking) to classify points into pitched (ordered, stable) or unpitched (chaotic, unstable) with measures of distortion or noisiness. Sadly this approach proved impractical as it achieved only tens of pixels per second, even with a fast multi-core CPU, and porting these signal analysis algorithms to massively-parallel programmable graphics hardware seemed to be too difficult.

5.2 OpenGL Issues

The current implementation is hardcoded with delay $d = 1$ and would be very awkward to generalize. OpenGL architecture limits each vertex attribute to four components with the maximum number of attributes typically limited to sixteen. This totals 64 floats per vertex, 6 of which are needed for the pixel coordinates and Lyapunov exponent statistics accumulation. Therefore using OpenGL imposes a limit $d < 28$. For comparison the original experiments in *Soft Rock EP* used Pure-data's default block size of 64, with $d = 32$. Moreover, increasing d increases video memory consumption. With the maximum $d = 27$, browsing at 1920×1080 resolution would require over 1GB.

Future work on this project will look into using OpenCL, which provides a heterogenous CPU and GPU computation framework, in the hope that it will avoid the inherent awkwardness of abusing OpenGL shaders to perform calculations.

5.3 Audio Issues

While the implementation works as intended, with $d = 1$ the sound is nowhere near as rich and varied as with $d = 32$. With small d there is much more very high frequency content in interesting-looking regions. There seem to be few if any regions of the parameter space with both interesting appearance and palatable audio frequencies at $d = 1$, while with high d there are parameters that generate sounds that fluctuate intermittently between smooth tones and noise. Visualization with high d has not been possible so far, so whether their neighbourhoods look as interesting as they sound remains an openquestion .

Unfortunately, heavy use of the GPU in the interactive browser can block the operating system for too long and cause audible glitches (JACK xruns). This situation may change as free drivers continue to improve, allowing use of the browser in a live situation.

5.4 Pretty Pictures

Despite these shortcomings, I think the images look good. I plan to render a selection at high resolution and print postcards and posters. For huge images it is possible to divide the image plane into tiles and compute each tile in succession, finally combining the pieces into one large picture.

There is also scope for video work, moving and rotating the viewing plane through the 4D parameter space, with different shapes forming and collapsing over time. Rough benchmarks take 5-10 seconds per frame at 1920×1080, so it seems sensible to wait until faster cheaper graphics cards become available.

6 Obtaining the Implementation

The implementation was written on GNU/Linux Debian Wheezy running on a quad-core AMD64 processor with NVIDIA GTX 550Ti graphics card using proprietary drivers. The source code is available at: `https://gitorious.org/maximus/lyapunov-fm`

7 Acknowledgements

Thanks to the anonymous reviewers for their constructive criticism on a number of issues, and to Rob Canning, Adnan Hadzi, and Joanne Seale for their helpful feedback on earlier versions of this paper.

References

ClaudiusMaximus. 2005. *Soft Rock EP*. http://archive.org/details/ClaudiusMaximus_-_Soft_Rock_EP.

ClaudiusMaximus. 2006. *Soft Rock DVD*. http://archive.org/details/ClaudiusMaximus_-_Soft_Rock_DVD.

Paul Davis. 2013. *The JACK Audio Connection Kit*. http://jackaudio.org.

A. K. Dewdney. 1991. Mathematical Recreations: Leaping into Lyapunov Space. *Scientific American*, 265:178–180.

Glenn Elert. 2007. *The Chaos Hypertextbook*. http://hypertextbook.com/chaos/.

Kenneth Falconer. 2003. *Fractal Geometry: Mathematical Foundations and Applications, Second Edition*. Wiley.

John Kessenich. 2013. *The OpenGL Shading Language*. http://www.opengl.org/registry/doc/GLSLangSpec.4.30.8.pdf.

Mark J. Kilgard. 1996. *The OpenGL Utility Toolkit (GLUT) Programming Interface*. http://www.opengl.org/documentation/specs/glut/glut-3.spec.pdf.

Mark Segal. 2013. *The OpenGL Graphics System: A Specification*. http://www.opengl.org/registry/doc/glspec43.core.20130214.pdf.

Dan Slater. 1998. Chaotic Sound Synthesis. *Computer Music Journal*, 22(2):12–19.