MUMT 618 Final Project:
Dynamical Systems for Audio Synthesis

Adam Bognat, 260550169

December 17, 2016

1 Introduction

"Virtual analog " describes a collection of approaches to sound synthesis that attempt to simulate analog hardware in software. A number of techniques exist, from transform methods rooted in ideas from the DSP literature, to physically-based methods that attempt to model the system from a fundamental perspective [SS96a, LHMW97, VH06, DSS10, RPS+07].

For a number of systems of interest to digital audio applications, traditional methods often produce digitized systems with uncomputable delay-free loops. Naively, one may correct this problem by inserting an ad hoc unit delay in the signal flow diagram, but this leads to undesirable artifacts, such as coupling parameters that should be independent, or aliasing and inharmonic distortion [SS96b, Med16].

This paper explores the approach proposed in [Med16] and [Med15]. The author considers a dynamical systems approach to modeling analog components. Specifically, he considers networks in which each component may be described by an explicit first-order ordinary differential equation (ODE) of the form

$$\dot{x} = f(x, t) \tag{1}$$

where a dot denotes differentiation with respect to time and the vector x represents some physical state variable, such as voltage across a capacitor. In particular, no assumptions are made on the derivative function f; indeed, of special interest is the solution of *non-linear* ODEs with which DSP-based approaches struggle.

The related K-method [BDPR00] was introduced to handle delay-free loops in systems of the form

$$\dot{x} = Ax + Bi(v) \tag{2}$$

$$v = Cx + Di(v) \tag{3}$$

where v and i are implicitly related in terms of non-linear functions. However, this method requires iterative solvers to handle the non-linearity which can lead to poor performance in real-time simulation of complex systems with many controllable parameters.

This paper is organized as follows. In section 2, we give an introduction to numerical methods for solving ODEs of the form (1) and review some common integration schemes. In section 3, we give a description of our objected-oriented programming environment, within which we can easily define systems described by (1) and obtain their solutions for general purpose processing. Section 4 develops a number of examples amenable to this approach, while section 5 discusses some of the associated difficulties and limitations. Section 6 offers a conclusion and some directions for future work. The appendix gives a brief summary of the included MATLAB code.

2 Numerical Methods

Most ODEs of the form (1) do not admit a closed-form solution in terms of familiar functions, and numerical methods must be used to obtain a solution. In choosing an appropriate scheme, one must consider issues of stability, convergence, and accuracy. While there is a huge literature on the subject [HH73, Kel92, Pre89], and a proper analysis is incredibly technical, the essence can be demonstrated with a simple example.

Consider the simple harmonic oscillator ODE, given by

$$\ddot{x} + \omega_0^2 x = 0 \tag{4}$$

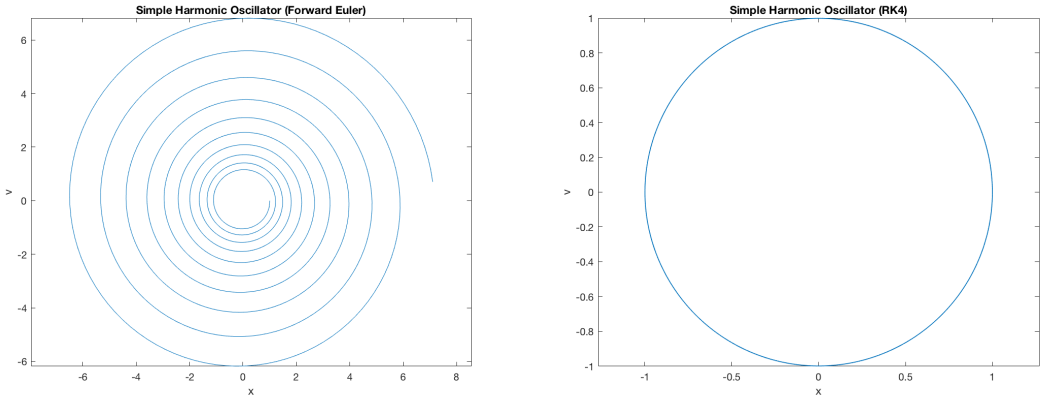
This is a second-order linear differential equation that describes a vast number mechanical, acoustic, and electrical phenomenon that involve simple vibrations; generically, one can think of x as describing a single quadratic degree of freedom, oscillating with natural frequency ω_0. By introducing a new variable $v = \dot{x}$, the above can be expressed as a first-order linear differential equation in two variables, *viz.*

$$\dot{x} = \omega_0 v \tag{5}$$

$$\dot{v} = -\omega_0 x \tag{6}$$

This equation admits the exact solution

$$x = A\cos(\omega_0 t + \phi) \tag{7}$$



(a) (b)

Figure 1: Phase portraits of the simple harmonic oscillator. (a) Forward Euler diverges very badly after only a few cycles. (b) Fourth-order Runge-Kutta remains bounded.

where the amplitude A and phase ϕ may be determined from initial conditions. Suppose, however, that it did not admit a nice closed-form solution. Given initial conditions $x(0) = x_0, v(0) = v_0$, an approximate solution could be obtained by partitioning time into N intervals t_n of small length h, and computing the value of $x_n = x(t_n)$ according to the rule

$$x_{n+1} = x_n + h\dot{x}_n \tag{8}$$
$$v_{n+1} = v_n + h\dot{v}_n \tag{9}$$

This "Forward Euler" scheme is based on the first-order Taylor approximation of a differentiable function. As such, the local error is $\mathcal{O}(h^2)$ while the global error is $\mathcal{O}(h)$, and so an impractically small stepsize needs to be chosen to obtain a reasonably accurate solution.

Even if accuracy is not an issue, the Forward Euler scheme is limited in applicability due to its poor stability properties. Consider a test equation

$$\dot{x} = \lambda x \tag{10}$$

with $\lambda \in \mathbb{C}$. For λ purely imaginary, this system is equivalent to the simple harmonic oscillator. More generally, it describes a damped harmonic oscillator, with damping given by the real part. Substituting into (8), we find

$$x_n = x_{n-1} + h\dot{x}_{n-1} \tag{11}$$
$$= (1 + h\lambda)x_{n-1} \tag{12}$$
$$= (1 + h\lambda)^N x_o \tag{13}$$

For absolute stability, the asymptotic behaviour should be bounded, so that

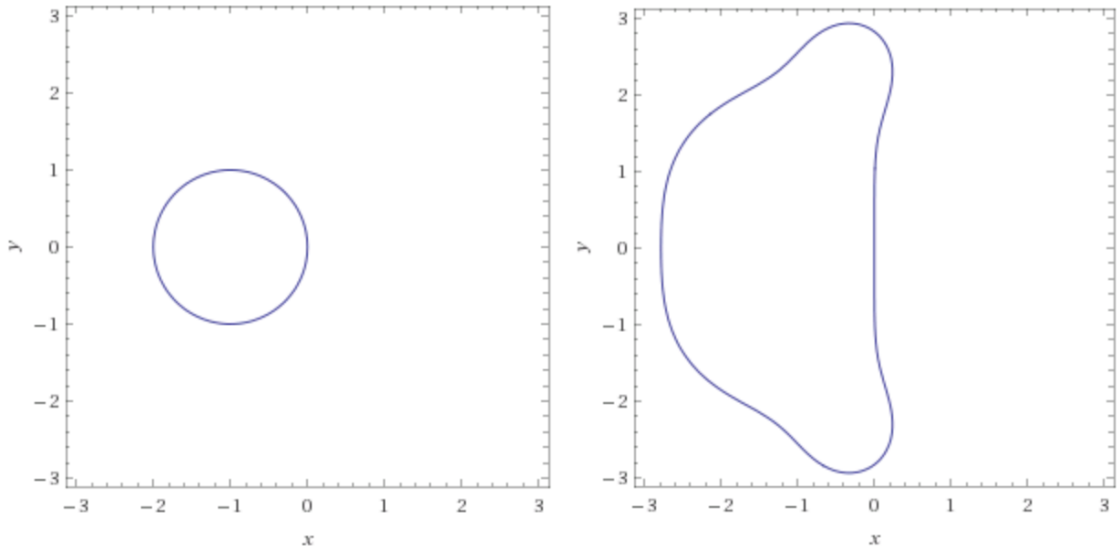
$$||1 + h\lambda|| < 1 \tag{14}$$

Writing $\lambda = a + \imath b$ with $a, b \in \mathbb{R}$, and setting $h = 1$, we require

$$(a + 1)^2 + b^2 < 1 \tag{15}$$

The region of stability is shown in Figure 2(a). Note that purely oscillatory systems fall outside this region, and so Forward Euler is inappropriate for the SHO. In practice, artificial damping is introduced to ensure stability for a limited range of frequencies. Figure 1(a) illustrates the rapid divergence of the Forward Euler scheme.

A more appropriate integration scheme for integrating (1) is the so-called fourth-order Runge-



(a) (b)

Figure 2: The regions of stability corresponding to (a) Forward Euler and (b) RK4. The test equation (11) with $\lambda = x + \imath y$ is absolutely stable in the regions defined by the interiors of the closed curves.

Kutta (RK4) method, defined by [Pre89]

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{16}$$

$$k_1 = f(x_n, t_n) \tag{17}$$

$$k_2 = f(x_n + h/2k_1, t_n + h/2) \tag{18}$$

$$k_3 = f(x_n + h/2k_2, t_n + h/2) \tag{19}$$

$$k_4 = f(x_n + hk_3, t_n + h) \tag{20}$$

The global error scales as $\mathcal{O}(h^4)$, and the region of stability is shown in Figure 2(b). In particular, it includes a portion of the imaginary axis, so that RK4 is appropriate for purely oscillatory systems. The region is still bounded, however, so that very high frequency ("stiff") systems will still suffer from convergence problems.

The results in this paper were generating using RK4, but our programming interface is designed to be used with any "explicit" integration scheme. Both RK4 and FE are explicit schemes, in that the new state of the system depends only on the derivative evaluated at the old state. One can also consider implicit schemes, such as

$$x_{n+1} = x_n + hf(x_{n+1}) \tag{21}$$

This "Backward Euler" scheme can be shown to be unconditionally stable, but generally requires iterative methods to solve for x_{n+1} and is thus computationally expensive.

3 Oscillator Systems

We would like to separate the description of a system in terms of ODEs from both the integrator and the processing of the output. For explicit schemes, all the integrator needs to know is the current state, the step size, and the derivative evaluated at the current time. Consequently, it is the job of the system to evaluate its derivative and pass it to the integrator. Finally, the output may be processed in different ways; real-time audio processing requires writing to a buffer, while offline processing can write to disk.

With these considerations in mind, we define the abstract classes `OscillatorSystem` and `Integrator` to implement the numerical integration of our system within an object-oriented framework.

An `OscillatorSystem` represents a dynamical system whose state is governed by an explicit ODE. The name is inspired by the computer graphics literature, wherein objects that can be described by a collection of masses connected by springs comprise a `ParticleSystem` [Wit01]. The `ParticleSystem` has knowledge of its state and the forces acting on it, and has methods for computing its derivative and updating its state. Likewise, a concrete system meant to represent a component of an analog network derives from `OscillatorSystem` and overwrites its `calcDeriv` and `updateState` methods appropriately.

Integration schemes are represented with the `Integrator` base class. The class is initialized with the step size and a specific integration scheme is implemented by overriding the `step` method. There are no specific classes to handle output processing; the included demos are not real-time and write the output to an array for visualization and auralization.

To update its state, an `OscillatorSystem` calls `Integrator`'s `step` method; `step` calls `OscillatorSystem`'s `calcDeriv` method, computes the new state, and passes it back by calling `OscillatorSystem`'s `updateState` method. This is demonstrated in the following pseudocode:

```
class ForwardEuler extends Integrator
    step(sys,t)
        old = sys.state;
        delta = old + h * sys.calcDeriv(t);
        sys.updateState(old,delta)

class QuadratureOscillator extends OscillatorSystem
    omega
    deriv = calcDeriv(t)
        deriv[1] =  omega * state[2]
        deriv[2] = -omega * state[1]
     updateState(old, delta)
        state = old + delta

FE = new ForwardEuler(h)
SHO = new QuadratureOscillator(omega, state0)

while simulation is running
    FE.step(SHO)
    //process output
```

4 Examples

4.1 Reciprocal Sync

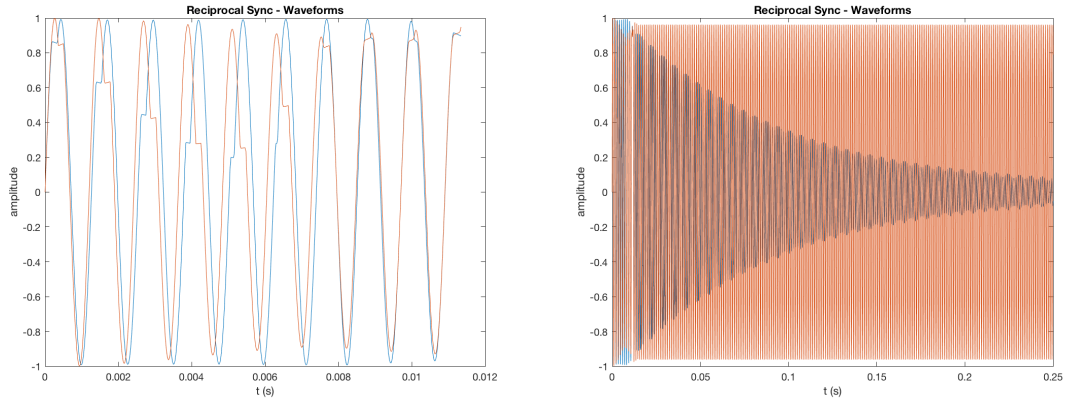
We have already discussed the basic oscillator circuit. A more interesting system is two oscillators in reciprocal sync. Traditionally, "hard sync" describes when the phase of a "slave" oscillator is reset whenever the amplitude of a "master" oscillator crosses a certain threshold. In reciprocal sync, the amplitude of the slave can also reset the phase of the master. Naive implementations of either result in unpleasant aliasing [Bra01, Med16].

Let x_1, v_1, x_2, v_2 denote the states of two oscillators with frequencies ω_1, ω_2, respectively, and let $\theta 1, \theta 2$ be the amplitude threshold values. Then reciprocal sync may be approximated by the equations

$$\dot{x}_1 = \begin{cases} \omega_1 v_1, & \text{if } x_2 < \theta_1. \\ g(x_{1,0} - x_1), & \text{otherwise.} \end{cases} \tag{22}$$

$$\dot{v}_1 = \begin{cases} -\omega_1 x_1, & \text{if } x_2 < \theta_1. \\ g(v_{1,0} - v_1), & \text{otherwise.} \end{cases} \tag{23}$$

with similar equations for x_2, v_2. Resetting the phase discontinuously would result in a large value of the derivative and lead to stability problems. Instead, we exponentially drive the system back to its initial conditions. The "gain" parameter g controls the speed of the decay, and should be on the order of the oscillator frequency for musical applications.



(a) (b)

Figure 3: Waveforms from reciprocal sync at different resolutions. The parameters used were $\omega_1 = 950$ Hz, $\omega_2 = 900$ Hz, $\theta_1 = \theta_2 = 0.9$, and $g = 100$.

Figure 3 shows an example of the waveforms resulting from reciprocal sync. The global exponential envelop is an artifact of the initial conditions not being exactly restored.

4.2 Feedback FM

Freqency Modulation (FM) synthesis is a synthesis technique whereby the frequency of one oscillator is modulated by another at audio rates [Cho77]. Incredibly complex sounds can be produced by a network of oscillators modulating each other in this way. If x_i, v_i denote the state of the i^{th} oscillator in the network, we have

$$\dot{x}_i = (\omega_i + \lambda_i)v_i \tag{24}$$
$$\dot{v}_i = -(\omega_i + \lambda_i)x_i \tag{25}$$

where

$$\lambda_i = \sum_j \alpha_{ij} x_j \tag{26}$$

and the sum runs over all oscillators. The α_{ij}'s give the "modulation index" of oscillator i acting on oscillator j. They can be thought of as components of a matrix α that describes the connectivity of the oscillator network. Of interest is the case of a system comprising a single oscillator feeding back on itself, with

$$\lambda_1 = \alpha x_1 \tag{27}$$

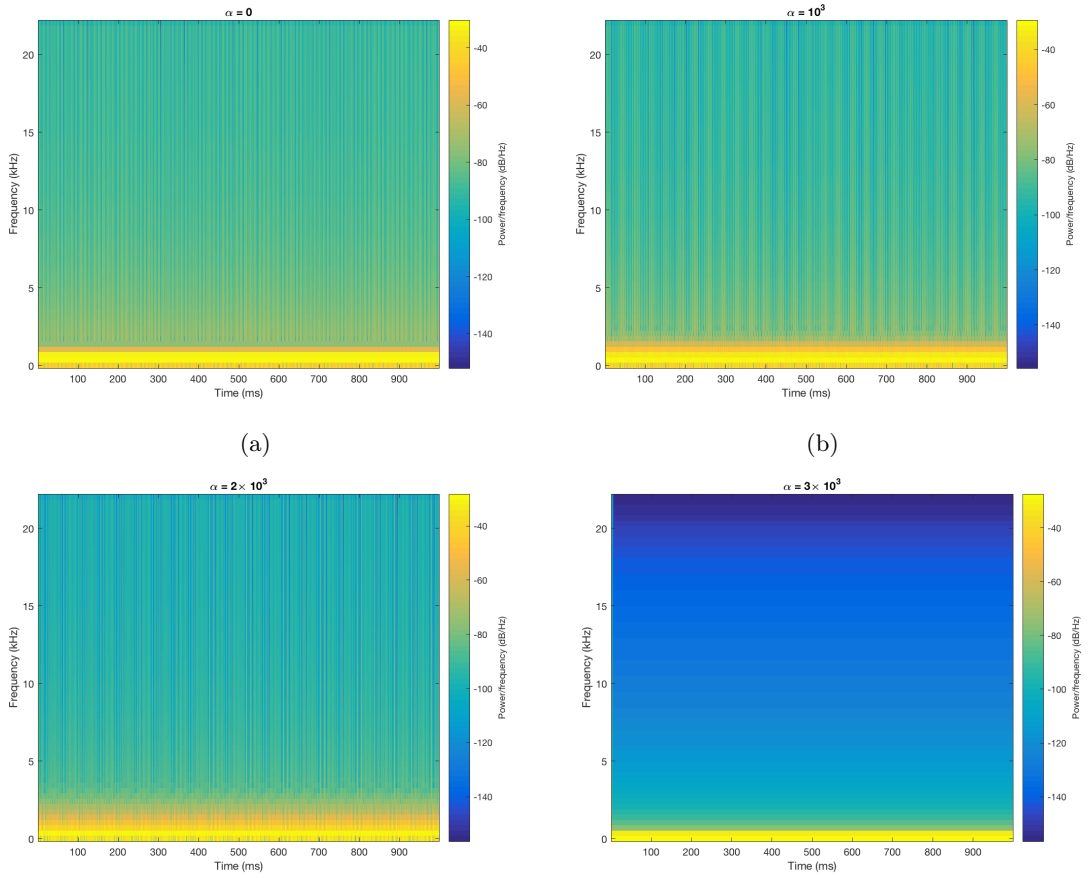
Figure 4 shows spectrograms of the resulting signal for different values of α. Note the narrow range of useful values of the modulation index; $\alpha \leq 10^3$ results in no audible modulation, while the signal frequency is too low to be perceived for $\alpha \geq 3 \times 10^3$.

4.3 Moog Ladder Filter

Finally, we consider a dynamical systems model of the Moog Ladder Filter. The circuit consists of a non-linear four-stage analog network with a feedback path to provide resonance. The feedback results in a delay-free loop when digitized using traditional transformation methods, and an ad hoc unit delay couples the resonance parameter to the cutoff frequency [SS96b]. If V_i represents the voltage across the i^{th} stage of the transistor ladder, we have

$$\dot{V}_i = \omega_c(\tanh(V_{i-1}) - \tanh(V_i)), \qquad i = 1, 2, 3 \tag{28}$$
$$\dot{V}_0 = \omega_c(\tanh(V_{in} - rV_3) - \tanh(V_0)) \tag{29}$$

where the gain ω_c is the cutoff frequency and r is the resonance parameter. V_{in} is the input voltage at the base of the ladder and can represent a control signal coming from another part of the circuit. This is the first example where the derivative is explicitly time dependent. This can be implemented



(a) (b) (c) (d)

Figure 4: Spectrograms of feedback FM waveform with fundamental frequency $\omega_0 = 440$ Hz. The appearance of sidebands with increasing modulation index is accompanied by a decrease in fundamental frequency.

in our system in a number of ways. One option is to hardcode the time dependence into the class as a static method; for simplicity, this is the approach taken in the demo. More generally, we may create a separate class for the time-dependent aspect (thinking of it as an external control signal) and include pointers to these classes within `OscillatorSystem`.

Figure shows the impulse response of the filter for different values of r. The filter is clearly self-resonating for $r \geq 4$.

5 Discussion

Our system is straightforward to implement and readily extensible. However, the results are disappointing from the perspective of musical applications. A major difficulty is mapping "physical" system parameters to useful creative parameters. For example, the feedback FM system yields audible modulation for α in a very narrow range centered on $\alpha = 10^3$; moreover, α couples to the base frequency of the oscillator, so that a richer spectrum comes at the cost of lower frequency. This behaviour is not typical of FM synthesizers implemented using DSP chips, and it is not immediately obvious how to fix it.

Implementing reciprocal sync by dynamically driving the phase to zero leads to problems of its own. If the g parameter is too small, the waveform will not reset rapidly enough, defeating the purpose of the sync. On the other hand, a very large g leads to stability problems, resulting in divergent solutions. Moreover, most combinations of parameter values give decaying solutions which may be useful for synthesis of percussive sounds but not the steady, harmonically rich signals that sync is meant to produce. A more accurate modeling of the analog circuitry with which sync is implemented may resolve these issues, but then it is not obvious that a dynamical systems

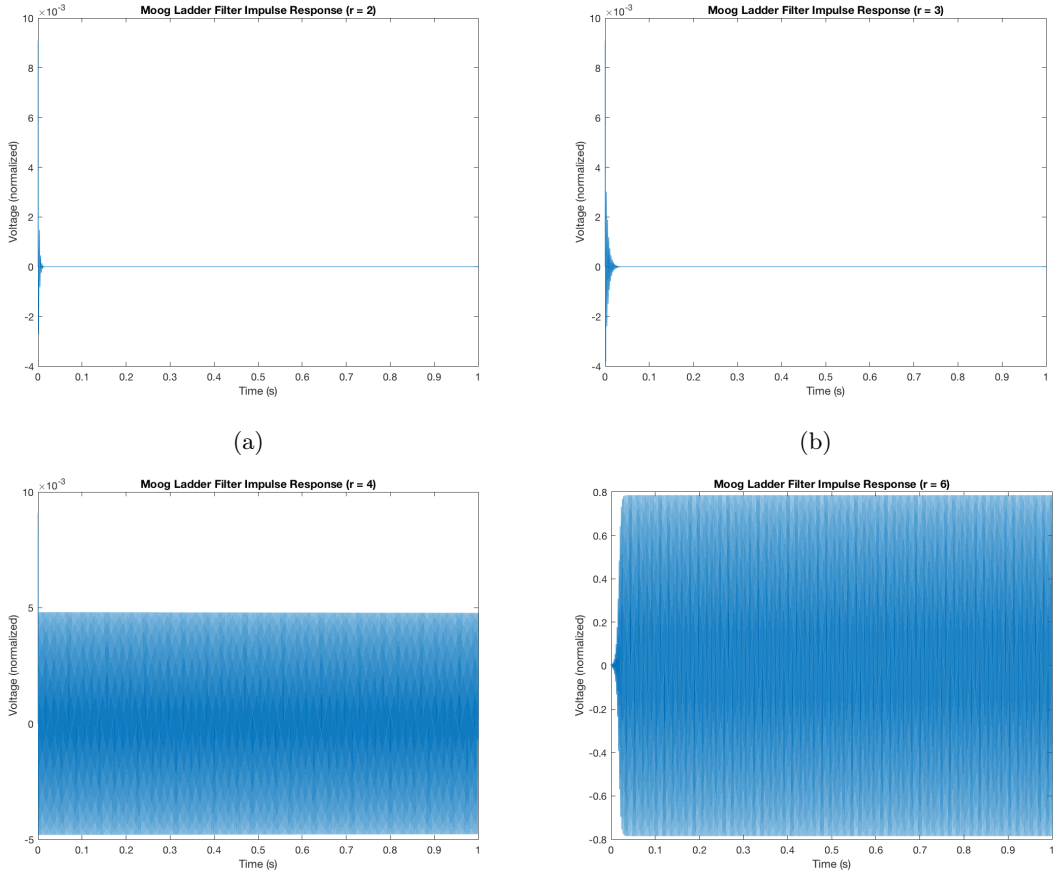


Figure 5: Impulse response of system governed by (25), (26). For low values of the resonance the filter response decays but self-resonates for values of r greater than 4.

approach is preferable to others.

6 Conclusion

A dynamical systems approach to analog circuit modeling allows emulation of systems with delay-free loops without introducing ad hoc delays and their digital artifacts. To this end, we have presented an object-oriented programming interface for simulating analog circuitry that can be described by a set of explicit ODEs. The interface separates the description of the system from the integration scheme and output processing, and is easily extended to allow for different explicit integration schemes, more complicated dynamical systems, and real-time output.

At present, the musically relevant possibilities of this approach are limited. Difficulties include accurate modeling of the desired system and translation of physical parameters to parameters suitable for creative purposes. A graphical interface for real-time exploration would help overcome these difficulties.

A MATLAB Code

The accompanying MATLAB code includes the abstract classes `OscillatorSystem`, `Integrator`, all the derived subclasses corresponding to examples presented in the paper, and demo scripts illustrating their use, as well as code for plotting the states, their spectra, the dynamical phase space, and listening to the output waveforms.

To implement a new dynamical system, extend `OscillatorSystem` and override the `calcDeriv` and `updateState` appropriately.

To implement a different integration scheme, extend `Integrator` and override the `step` method. The interface assumes an explicit integration scheme, and a new abstract class for implicit integration should be implemented if an implicit scheme is desired.

The step size should correspond to the sample rate appropriate for reconstructing your audible output signal without aliasing. The default is 44100 Hz.

References

[BDPR00] Gianpaolo Borin, Giovanni De Poli, and Davide Rocchesso. Elimination of delay-free loops in discrete-time models of nonlinear acoustic systems. *IEEE Transactions on Speech and Audio Processing*, 8(5):597–605, 2000.

[Bra01] Eli Brandt. Hard sync without aliasing. *power (dB)*, 60:40, 2001.

[Cho77] John M Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Computer Music Journal*, pages 46–54, 1977.

[DSS10] Giovanni De Sanctis and Augusto Sarti. Virtual analog modeling in the wave-digital domain. *IEEE transactions on audio, speech, and language processing*, 18(4):715–727, 2010.

[HH73] R.W. Hamming and R.W. Hamming. *Numerical Methods for Scientists and Engineers*. Dover books on engineering. Dover, 1973.

[Kel92] H.B. Keller. *Numerical methods for two-point boundary-value problems*. Dover Publications, 1992.

[LHMW97] John Lane, Dan Hoory, Ed Martinez, and Patty Wang. Modeling analog synthesis with dsps. *Computer Music Journal*, 21(4):23–41, 1997.

[Med15] David Medine. Unsampled digital synthesis: Computing the output of implicit and non-linear systems. In *Looking Back, Looking Forward: Proceedings of the 41st International Computer Music Conference, ICMC 2015, Denton, TX, USA, September 25 - October 1, 2015*, 2015.

[Med16] David Medine. Dynamical systems for audio synthesis: Embracing nonlinearities and delay-free loops. *Applied Sciences*, 6(5):134, 2016.

[Pre89] W.H. Press. *Numerical Recipes in Pascal (First Edition): The Art of Scientific Computing*. Number v. 1. Cambridge University Press, 1989.

[RPS+07] Rudolf Rabenstein, Stefan Petrausch, Augusto Sarti, Giovanni De Sanctis, Cumhur Erkut, and Matti Karjalainen. Blocked-based physical modeling for digital sound synthesis. *IEEE Signal Processing Magazine*, 24(2):42–54, 2007.

[SS96a] Tim Stilson and Julius Smith. Alias-free digital synthesis of classic analog waveforms. In *Proc. International Computer Music Conference*, 1996.

[SS96b] Tim Stilson and Julius Smith. Analyzing the moog vcf with considerations for digital implementation. In *Proceedings of the 1996 International Computer Music Conference, Hong Kong, Computer Music Association*, 1996.

[VH06] Vesa Välimäki and Antti Huovilainen. Oscillator and filter algorithms for virtual analog synthesis. *Computer Music Journal*, 30(2):19–31, 2006.

[Wit01] Andrew Witkin. Physically based modeling particle system dynamics. *ACM SIGGRAPH Course Notes*, 2001.