FB1_ODE – AN INTERFACE FOR SYNTHESIS AND SOUND PROCESSING WITH ORDINARY DIFFERENTIAL EQUATIONS IN SUPERCOLLIDER

Daniel MAYER (mayer@iem.at)[1]

[1] *Institute of Electronic Music and Acoustics (IEM)*, **University of Music and Performing Arts,** Graz, Austria

ABSTRACT

The class Fb1_ODE, included in the miSCellaneous_lib quark extension library [1] of SuperCollider (SC, [2, 3]), enables the audible integration of ordinary (systems of) differential equations (ODEs) with initial values in realtime. The prefix 'Fb1' refers to the class Fb1 for single sample feedback and feedforward, on which it depends [4]. Consequently, the numerical integration of ODE systems with a step width of one sample is possible with arbitrary block sizes of SC's audio engine. Fb1_ODE opens the possibility for immediate audio experiments with models from physics, electrical engineering, population dynamics, chemistry, etc., preferably those with oscillatory respectively quasi-oscillatory solutions or chaotic features. Designing new ODEs from scratch or altering respectively disturbing systems can also be interesting regarding the sounding results. Wrappers of Fb1_ODE include well-known systems like Van der Pol, Duffing, Hopf, Mass-Spring-Damper, and Lorenz; users can interactively add other systems with the class Fb1_ODEdef. The modulation of ODE parameters, system time, and the feeding of additional audio signals into ODE systems are, amongst others, further options for unorthodox synthesis with differential equations.

1. INTRODUCTION

We regard systems of the form

$$Y'(t) = F(t,\ Y(t)) \tag{1}$$

in the domain of real numbers where Y and F can be vector-valued functions and the restriction of an initial value condition

$$Y(t_0) = y_0 \tag{2}$$

In a physical interpretation, t is the system time.

1.1 Why using ODEs for audio synthesis?

That is a legitimate question, not at least because of some counterarguments. There are numerical hurdles, calculations often become CPU-demanding, and the usage of arbitrary ODE models in many fields of science and technology is no direct argument for their application in sound and music – besides from the ongoing research in acoustic and physical models [5]. However, an outweighing argument for comprehensive ODEs also comes from the fact that they can work as a generic **description system** for waveforms: many ODE solutions cannot be expressed in an analytical form. This consideration leads to the assumption that ODEs can act as a key to a land of unknown and intriguing possibilities in sound synthesis. The growing significance – one might even say: popularity – of non-linear dynamical systems has certainly supported this view. How to choose from these possibilities in artistic regard is a crucial question that needs practical exploration – a general-purpose tool as the presented one aims to provide quick feedback.

The power of ODEs as a description system already shows up by this trivial example. The second-order differential equation

$$y''(t) = -y(t) \tag{3}$$

is – by the substitution

$$w(t) = y'(t) \tag{4}$$

– reduced to the first-order ODE system

$$\begin{aligned} y'(t) &= w(t) \\ w'(t) &= -y(t) \end{aligned} \tag{5}$$

With the initial values $y(0) = 0$ and $w(0) = 1$, the system has the solution

$$\begin{aligned} y(t) &= \sin(t) \\ w(t) &= \cos(t) \end{aligned} \tag{6}$$

The initial equation (3) is, obviously, much more compact than the representations of sine and cosine as infinite series derived from Taylor expansion. On the other hand, using numerical ODE integration for producing a sine

wave doesn't make much sense except a proof of concept. However, there exist many ODE systems with brief definitions that can generate rich and evolving spectra.

Researchers have made several suggestions to use ODEs for audio synthesis and processing. The approaches partially origin from the world of analog circuits (Slater [6]: Ueda, a variant of the Duffing oscillator, Choi [7] and Rodet [8]: Chua circuit). Jacobs [9] uses the FitzHugh-Naguamo model of which Van Der Pol is a particular case. See Falaize and Hélie for stable simulations of analog audio circuits from electronic schematics [10]. State space models aim to represent analog systems by input, output and state variables as parts of ODE systems (e.g., [11, 12]). Wave digital filters are an attempt to digitize analog circuits by ODEs and traveling-wave components [13].

SC's main class library already includes the famous and widely used Lorenz model. The implementation via Fb1_ODE additionally allows the feeding of external input into the system, an experimental feature of ODE synthesis also recommended by Stefanakis, Abel, and Bergner [14, pp. 53–55, 57].

1.2 Numerical integration of ODEs

There exist many integration techniques, which serve well in typical engineering applications. However, there is a specific demand with the audification of ODEs: oscillations should be kept stable over a relatively long period – or put in other words, we need many oscillations to get an audible signal for a significant amount of time. It is likely to encounter drifts in the long run with arbitrary integration methods. E.g., already in the case of a simple harmonic oscillator, a 3rd order Runge-Kutta scheme can fail (Figure 5). For several oscillators, there exist specific integration schemes (Duffing: Bilbao [5, p. 75–77], Van der Pol: [15]). More general, so-called symplectic procedures have gained attention [16]. Roughly spoken, they preserve volume in a geometric sense and are often well suited for ODE audification. See the chapter on integration in David Pirrò's dissertation [17, pp. 135–146]. David Pirrò has implemented the symmetric symplectic "rattle" integration in his optimized ODE compiler *Henri*, Fb1_ODE uses this scheme as default [18, 19, 20]. Implicit integration techniques are also used widely in audio applications [21]. For implementational reasons – especially the interactive adding of schemes – preference has been given to the explicit methods, from which several are built-in at choice (see 2.5).

2. IMPLEMENTATION IN SC

2.1 Definition of ODEs and usage as unit generators

The user interface for the most general purposes mainly consists of two classes in the SC language (the SC client): Fb1_ODEdef and Fb1_ODE. After defining an ODE with Fb1_ODEdef – by providing the function F of (1) as an SC Function – it is ready for synthesis usage with Fb1_ODE. The latter is a so-called pseudo-UGen (pseudo unit generator), a compound UGen structure comparable to macros in other languages. Under the hood, Fb1_ODE merges the selected numerical procedure, applied to F, into a UGen graph. That ensures integration on a per-sample base when employing the compiled synthdef (instrument) on the SC server (the audio engine). It also holds for a server block size greater than 1 by involving the single sample feedback pseudo-UGen Fb1 [4]. Figure 1, as a proof of concept, shows the code for producing a sine wave by using the harmonic oscillator system.

```
// The Function (curly brackets) must take time and
// an array as args and return an array of the same size (2).
// In contrast to math notation indices start from 0.

Fb1_ODEdef(\harmonic,
    { arg t, y; [y[1], y[0].neg] },
    // default init values t0, y0
    t0: 0, y0: [0, 1]
)

// The stereo output is a pair of 90-degree-shifted sines.
// As one wavelength would be 2pi seconds, multiply by
// a factor 500 * 2pi to get 500 Hz.
// blockSize is detected automatically.

x = { Fb1_ODE.ar(\harmonic, tMul: 500 * 2pi) * 0.1 }.play
```

Figure 1. Sine wave by ODE integration

More interesting, basic systems like the harmonic oscillator or exponential decay can be the starting point for experimental variations. It's a promising strategy to gradually drift away from a base case, e.g., define the system

$$\begin{aligned} y'(t) &= w(t) \\ w'(t) &= -y(t)\,(1 + k\,w(t)) \end{aligned} \tag{7}$$

With $k = 0$, it equals the harmonic oscillator, greater values produce a brass-like sound.

```
// define system function with k as additional arg
Fb1_ODEdef(\harmonic_ext_1, { arg t, y, k;
    [
       y[1],
       y[0].neg * (1 + (k * y[1]))
    ]
}, 0, [0, 1]);

x = {
    Fb1_ODE.ar(
        \harmonic_ext_1,
        [2], 1500, // k = 2, tMul = 1500
        0, [0, 1] // init values
    ) * 0.1;
}.play;
```

Figure 2. Blurred harmonic oscillator.

2.2 Wrapper classes

The library provides dedicated pseudo-UGen classes for the well-known systems Van der Pol, Duffing, Hopf, Lorenz, and Mass-Spring-Damper (MSD). The latter satisfies the second-order differential equation

$$m\,y''(t) = -k\,y(t) - c\,y'(t) + F(t) \tag{8}$$

whereby m denotes the mass, c the dampen factor, k the spring stiffness, and $F(t)$ the externally applied force. Like

the harmonic oscillator, one can transform it into a system of two first-order equations whose solutions describe position and velocity. The code example in Figure 3 uses the oscillation, which converges to an end position, for frequency modulation. The audible result is a timbral development to a sine with a fixed frequency. Note that, by default, Fb1_ODE and the wrapper classes apply a DC leaker, which, in this case, is disabled.

```
x = {
    var f = 0.2, mass = 0.001, spring = 50,
        dampen = 0.0005, sig, mod;
    sig = Fb1_MSD.ar(
        f, mass, spring, dampen,
        // deflection converges to 0.004
        // hence take no LeakDC
        leakDC: false
    );
    // map position to useful frequencies
    // LR-decorrelation by slightly different ranges
    mod = sig[0].linlin(0, 0.005, [100, 101], 700);
    SinOsc.ar(mod, 0, 0.1)
}.play
```

Figure 3. Mass-Spring-Damper (MSD) used for FM.

See the help files of the classes Fb1_VanDerPol, Fb1_Duffing, Fb1_Hopf, Fb1_Lorenz, and Fb1_MSD for further examples. The adaptive variants Fb1_HopfA and Fb1_HopfAFDC can preserve the frequency of an external force after its stopping. That provides an unusual synthesis option. The classes implement the techniques described in [22] and [23], which are generic ways to make ODE systems adaptive.

2.3 Models from various fields

ODE models occur in many fields like physics (eminently mechanics), electrical engineering, population dynamics, and even chemistry. They can be interesting audio engines themselves or act as a starting point for further explorations. For an overview, see the SIAM publication *Exploring ODEs* [24] with an experimental approach in the graphic domain and many links and examples. Also, it's worth being aware that ODEs of one type can occur in different forms. The decision for a particular parameter set can have a vast impact on audio usability. Ultimately, parameter spaces demand a practical investigation. Fb1_ODE's help file contains examples from mechanics (driven pendulum, reduced two-body problem, Ex. 8a/b) and population dynamics (Lotka-Volterra and Hastings-Powell, Ex. 9a/b).

2.4 Modulations

It's possible to modulate systems parameters and the time scaling factor at audio rate – the latter can also get negative values. The example of Figure 4 varies that of Figure 3 by modulations of external force, mass, and time scaling (`tMul` argument). The specific choice preserves the development of the previous example – in general, it is easy to make systems unstable by operations of such kind. While the changing mass might still have a physical plausibility, the use of changing and even negative time steps is finally destroying a correct integration, though still possibly useful as a synthesis option.

```
x = {
    var f = SinOsc.ar(0.1).range(0.18, 0.2),
        mass = SinOsc.ar(1.2).range(1, 10) * 0.0001,
        spring = 50, dampen = 0.0005, sig, mod;
    sig = Fb1_MSD.ar(f, mass, spring, dampen,
        tMul: SinOsc.ar(5).range(-0.2, 1.5),
        leakDC: false
    );
    mod = sig[0].linlin(0, 0.005, [100, 101], 700);
    SinOsc.ar(mod, 0, 0.1)
}.play
```

Figure 4. MSD with modulations, used for FM.

2.5 Integration methods

While the use of symplectic integration procedures has advantages for the cited reasons, Fb1_ODE supports other families of integration methods like Euler, Prediction-Evaluation-Correction (PEC), Runge-Kutta, Adams-Bashforth, and Adams-Bashforth-Moulton as well. As an advanced feature, more integration methods can be added interactively with the class Fb1_ODEintdef. In some cases, alternative integration methods can lead to timbral variations. Quite often, though, they lead to blowups or decays, where stable oscillations should occur. Figure 5 shows an example with a 3rd order Runge-Kutta integration of the harmonic oscillator, which leads to decay after a few seconds.

```
// Needs Fb1_ODEdef from Fig. 1.
// Decay after a few seconds with Runge-Kutta 3rd order.
// ATTENTION: blowups with other non-symplectic procedures!
// Euler variants are particularly bad.

x = {
    var sig = Fb1_ODE.ar(\harmonic,
        tMul: 1000 * 2pi,
        intType: \rk3
    );
    Limiter.ar(sig) * 0.1 * EnvGen.ar(Env.asr(0.1))
}.play
```

Figure 5. Failing integration with 3rd order Runge-Kutta.

The symplectic "rattle" procedure has another advantage: it offers the option to improve accuracy by the iterated division of step sizes. To employ these variants, pass the SC Symbol 'sym' with one of the suffixes 2, 4, 6, 8, 12, 16, 32, 64 on Fb1_ODE's `intType` argument (default 'sym4').

2.6 Handling unstable systems

It is possible to insert an additional function, which applies to every array of samples that is the intermediate result – and next input – of the numerical integration procedure. Consequently, the correct integration of the ODE is out of scope. The option still has its value: limiting functions can prevent systems from blowing up. Let us regard this paraphrase on the Mass-Spring-Damper model:

$$m\,y''(t) = -k\,y(t) - c\,y'(t) + F(t) + y(t)\,y'(t) \qquad (9)$$

The use of this ODE in similar ways as in the examples from Figures 3-4 leads to a derail by infinite numbers after few seconds. However, the system remains inside practical bounds with a limiting operator like `clip2` (Figure 6).

```
Fb1_ODEdef(\MSD_ext, { arg t, y, f = 0,
    m = 1, s = 1, d = 0;
    [
        y[1],
        f - (d * y[1]) - (s * y[0]) + (y[0] * y[1]) / m
    ]
}, 0, [0, 0]);

x = {
    var f = 0.5, m = 0.001, s = 150, d = 0.001, mod;
    // smooth random LFO for clip values
    var lfo = LFDNoise3.kr(2).exprange(0.2, 1500);
    var sig = Fb1_ODE.ar(\MSD_ext,
        [f, m, s, d],
        leakDC: false,
        compose: { arg y; y.clip2(lfo) }
    );
    mod = sig[0].linlin(0, 0.006, [50, 50.1], 700);
    SinOsc.ar(mod, 0, 0.1)
}.play
```

Figure 6. MSD with disturbance, used for FM.

2.7 Further options and settings

Initial values – the system state y_0 at time t_0: $y(t_0) = y_0$ – are essential for an ODE solution. The user can pass them on Fb1_ODE with the corresponding arguments `t0` and `y0`. In many cases – like Mass-Spring-Damper with constant external force – a change of the start time does not have a consequence: the resulting waveform is the same, whereas the initial system values (position and velocity) have a significant impact.

Fb1_ODE can also return additional information in separate channels. By default, it returns the solution function(s) of the ODE system. Optionally, it can also output the differential and the time – which is not necessarily linear, as there might be a time modulation. The corresponding arguments are `withDiffChannels` and `withTimeChannel`. See the Fb1_ODE help file examples 6a and 6b.

Fb1_ODEdef allows for amplitude scaling factors. Usually, they default to 1, but certain ODEs, like Lorenz, produce a very high amplitude level with standard parameters. Therefore, it makes sense to scale their output down by default. However, with Fb1_ODE's `withOutScale` argument, the default scaling can be disabled (Fb1_ODE help file examples 7).

As many system solutions produce an unwanted DC offset, a DC leaker applies by default. The user can disable the option with Fb1_ODE's `leakDC` argument.

One of Fb1_ODE's basic arguments is `tMul` for time scaling. As in Figure 1, it can determine the resulting frequency (alternatively, the user might define the multiplication in the system definition with Fb1_ODEdef). It is important to note that numerical integration in the audio rate case is always performed on a per-sample base – even if the block size is larger than 1 – only the unit of the system time is varied. However, scaling is restricted to numerical accuracy limits: with extreme `tMul` values or numerically sensitive equations, you might encounter blowups or situations where the resulting frequency does not linearly relate to the scaling. Besides, Fb1_ODE can alternatively run at control rate (Fb1_ODE.kr), which helps to save CPU-load if the current block size is larger than 1.

2.8 Workflow recommendations, troubleshooting

The direct definition of the ODE systems in the language is a convenience that comes with the price of a possibly large number of unit generators involved. That does not necessarily mean a high CPU-load of the audio engine but leads to a higher compile-time. The user might want to extend SC's server resources before booting, e.g., set a higher number of unit generators with the server option `numWireBufs`. For a smooth workflow, I would recommend taking a reduced `blockSize` (e.g., 1, 2, 4, 8, 16) while experimenting because compile-time is shorter. But after fixing the design of a `SynthDef`, it might pay going back to a `blockSize` value of 32 or 64 for runtime efficiency, even more if many control rate unit generators are involved.

Especially with custom-designed ODEs, the usage of Fb1_ODE is – inherently – highly experimental. I strongly recommend being careful with amplitudes! Sudden blowups might result from the mathematical characteristics of the ODE systems. They might also stem from parameter adjustments – on which ODEs can react with extreme sensitivity – or from numerical accumulation effects. As a precautionary measure, users can employ SC's limiting/distorting operators (`tanh`, `clip`, `softclip`, `distort`) with the compose option (2.6) or external limiting, e.g., with the quarks JITLibExtensions (MasterFX) or SafetyNet.

The numerical integration procedure supposes well-defined ODE systems. The Fb1_ODE framework doesn't perform any checks concerning the principal existence and uniqueness of an ODE solution.

3. CONCLUSIONS

The SC class extension Fb1_ODE enables the audification of ordinary systems of differential equations with initial values in realtime. ODEs serve as a generic description system for waveforms, which one often cannot define by (explicit) mathematical means. Synthesis experiments have proven to be promising with well-known ODE systems from many scientific fields, as well as with custom-designed ODEs. Options for the modulation of ODE parameters and system time – and the integration mechanism itself – blur the model concept, though, also widen the field of sonic exploration.

Acknowledgments

Great thanks to my colleague David Pirrò of IEM Graz, who gave me a nudge to dive into the world of ODE audification, and his recommendation of symplectic integration procedures. Fb1_ODE rests upon the Fb1 class, therefore also big thanks to Nathaniel Virgo, whose idea for single sample feedback – with block sizes greater than 1 – lives therein.

4. REFERENCES

[1] D. Mayer, miSCellaneous_lib, SuperCollider quark extension: https://github.com/dkmayer/miscellaneous_lib, Accessed February 20, 2021.

[2] S. Wilson, D. Cottle and N. Collins (Eds.), The SuperCollider Book. The MIT Press, Cambridge, 2008.

[3] SuperCollider page, https://supercollider.github.io, Accessed February 20, 2021.

[4] D. Mayer, "Fb1 – an interface for single sample feedback and feedforward in SuperCollider," in Proc. Int. Conf. Sound and Music Computing (SMC2020), Torino, 2020, pp. 200–203. https://smc2020torino.it/adminupload/file/SMCCIM_2020_paper_53.pdf, Accessed February 20, 2021.

[5] S. Bilbao, Numerical sound synthesis: Finite difference schemes and simulation in musical acoustics, J. Wiley, Chichester, UK, 2009.

[6] D. Slater, "Chaotic sound synthesis," in Computer Music Journal, vol. 22, no. 2, pp. 1219, 1998.

[7] I. Choi, "Interactive exploration of a chaotic oscillator for generating musical signals in realtime concert performance," in Journal of the Franklin Institute, vol. 331, no. 6, pp. 785–818, 1994.

[8] X. Rodet, "Sound and Music from Chua's Circuit," in Journal of Circuits, Systems and Computers, vol. 3, no. 1, pp 49–61, 1993.

[9] B. Jacobs, "A Differential Equation Based Approach to Sound Synthesis and Sequencing," in International Computer Music Conference Proceedings, pp. 557–561, 2016.

[10] A. Falaize, T. Hélie, "Passive guaranteed simulation of analog audio circuits: A port-Hamiltonian approach," in Applied Sciences, vol. 6, no. 10, p. 273, 2016.

[11] M. Holters and U. Zölzer, "A generalized method for the derivation of non-linear state-space models from circuit schematics," Proc. 23rd European Signal Process. Conf. (EUSIPCO 2015), (Nice, France), pp. 1073–1077, IEEE, August 2015.

[12] K. Dempwolf, M. Holters, and U. Zölzer, "Discretization of Parametric Analog Circuits for Real-Time Simulations," Proc. of the 13th Int. Conference on Digital Audio Effects (DAFx-10), Graz, Austria, September 6-10, 2010, pp. 42–49.

[13] A. Fettweis, "Wave Digital Filters: Theory and Practice," Proc. IEEE, Vol. 74, No. 2, pp. 270–327, Feb. 1986.

[14] N. Stefanakis, M. Abel, and A. Bergner, "Sound synthesis based on ordinary differential equations," in Computer Music Journal, vol. 39, no. 3, pp. 46–58, 2015.

[15] A. Bernardini, P. Maffezzoni, and A. Sarti, "Linear Multistep Discretization Methods With Variable Step-Size in Nonlinear Wave Digital Structures for Virtual Analog Modeling," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 27, no. 11, pp. 1763-1776, Nov. 2019.

[16] E. Hairer, C. Lubich, and G. Wanner, Geometric numerical integration: structure-preserving algorithms for ordinary differential equations, Springer, 2006.

[17] D. Pirrò, Composing Interactions. Dissertation, Institute of Electronic Music and Acoustics, University of Music and Performing Arts Graz, 2017. https://pirro.mur.at/media/pirro_david_composing_interactions_print.pdf, Accessed February 20, 2021.

[18] D. Pirrò, Henri – llvm-based sound synthesis and dynamical systems numerical integration environment. https://git.iem.at/davidpirro/henri, Accessed February 20, 2021.

[19] G. Strang, "On the Construction and Comparison of Difference Schemes," in SIAM Journal on Numerical Analysis, Vol. 5, No. 3 (Sep., 1968), pp. 506-517.

[20] R.I. McLachlan, G.R.W. Quispel, "Splitting methods," in Acta Numerica, Volume 11, January 2002, pp. 341–434.

[21] F.G. Germain, "Fixed-rate modeling of audio lumped systems: A comparison between trapezoidal and implicit midpoint methods," in Proc. 20th Int. Conf. Digital Audio Effects (DAFx-17), Edinburgh, UK, Sept. 2017, pp. 168–175.

[22] L. Righetti, J. Buchli, and A.J. Ijspeert, "Adaptive Frequency Oscillators and Applications," in The Open Cybernetics and Systemics Journal, 3, pp. 64–69, 2009. https://www.researchgate.net/publication/41666931, Accessed February 20, 2021.

[23] T. Nachstedt, C. Tetzlaff, and P. Manoonpong, "Fast Dynamical Coupling Enhances Frequency Adaptation of Oscillators for Robotic Locomotion Control," in Frontiers in Neurorobotics. Published online March 21, 2017: https://www.frontiersin.org/articles/10.3389/fnbot.2017.00014/full, Accessed February 20, 2021.

[24] L. Trefethen, A. Birkisson, and T.Driscoll, Exploring ODEs. SIAM – Society for Industrial and Applied Mathematics, 2018. http://people.maths.ox.ac.uk/trefethen/Exploring.pdf Accessed February 20, 2021.