A Toolbox for Simulation of Hybrid Systems in Matlab/Simulink

[Hybrid Equations (HyEQ) Toolbox] *

Ricardo G. Sanfelice
University of Arizona
USA
sricardo@u.arizona.edu

David A. Copp
University of Arizona
USA
dacopp@email.arizona.edu

Pablo Ñañez
Universidad de Los Andes
Colombia
pa.nanez49@uniandes.edu.co

ABSTRACT

This paper describes the Hybrid Equations (HyEQ) Toolbox implemented in Matlab/Simulink for the simulation of hybrid dynamical systems. This toolbox is capable of computing approximations of trajectories to hybrid systems given in terms of differential and difference equations with constraints, called *hybrid equations*. The toolbox is suitable for the simulation of hybrid systems with different type of trajectories, including those that are Zeno and that have multiple jumps at the same instant. It is also capable of simulating hybrid systems without inputs, with inputs, as well as interconnections of hybrid systems. The structure, components, and usage of the simulation scripts within the toolbox are described. Examples are included to illustrate the main capabilities of the toolbox.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems - Environments; G.4 [**Mathematical Software**]: Reliability and robustness; I.2.8 [**Problem Solving, Control Methods, and Search**]: Control theory.

Keywords

Simulation; Hybrid Systems; Matlab/Simulink

1. INTRODUCTION

Simulation is a key tool in the validation of results in engineering and science. Complete theories of numerical simulation have been documented in several textbooks (see, e.g., [16, 3]) and have permitted the development of integration schemes for accurate computation of solutions to differential equations. Integration schemes for such systems are widely available in simulation packages and include one-step methods, such as forward/backward Euler and Runge-Kutta, multi-step methods, such as Adams method and backward differentiation, and their variable step versions. When, in addition to continuous behavior, the model incorporates variables that exhibit jumps, advanced integration methods are needed. Numerous software packages have been recently developed for such class of systems, including Modelica [6], Ptolemy [12], Charon [2], HYSDEL [17], and HyVisual [11], to just list a few. Significant progress has been also made in the development of theory for hybrid systems, with results including the definition of semantics for simulation [10, 14, 11], event detection [13, 7, 5], solvers and error control [7, 4, 1], and structural properties of simulators [15].

Exploiting the robustness properties of hybrid systems modeled as in [9, 8] and the structural properties of simulators in [15], we present a toolbox for the simulation of hybrid systems in Matlab/Simulink. A hybrid system $\mathcal{H}$ is given by the *hybrid equations*

$$\mathcal{H}: \quad x \in \mathbb{R}^n,\ u \in \mathbb{R}^m \quad \begin{cases} \dot{x} &= f(x,u) & (x,u) \in C \\ x^+ &= g(x,u) & (x,u) \in D \end{cases} \tag{1}$$

with the following objects defining its data: the set $C \subset \mathbb{R}^n \times \mathbb{R}^m$ called the *flow set*; the function $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ called the *flow map*; the set $D \subset \mathbb{R}^n \times \mathbb{R}^m$ called the *jump set*; the function $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ called the *jump map* [1]. These objects define the *data* of $\mathcal{H}$, which is explicitly denoted as $\mathcal{H} = (f, C, g, D)$. Examples illustrate the type of systems that can be modeled within the hybrid equations framework; see [9, 8] for more examples.

The proposed toolbox to simulate this class of systems is called the *Hybrid Equations (HyEQ) Toolbox* and consists of a set of Matlab/Simulink scripts to numerically compute and plot the trajectories of hybrid systems given in terms of hybrid equations. Among several things, the HyEQ Toolbox has the following components and features: 1) A Matlab script for simulation of hybrid equations without inputs and a Simulink library for simulation of hybrid equations with inputs; 3) Computation of trajectories that are Zeno and that

*The current version of the toolbox can be found at Matlab Central and at the author's website http://www.u.arizona.edu/∼sricardo/. This research has been partially supported by the National Science Foundation under CAREER Grant no. ECS-1150306 and by the Air Force Office of Scientific Research under Grant no. FA9550-12-1-0366.

HSCC'13, April 8–11, 2013, Philadelphia, Pennsylvania, USA.

[1]In other works, the terms *reset map* and *impact map* are used for g while the term *switching surface* is sometimes used for D.

have multiple jumps at the same instants; 4) Basic event detection (set inclusion) and capability to implement advanced crossing detection algorithms; 5) Simulation of interconnections of hybrid systems. In Section 2, the main components of the toolbox are described. In Section 3, examples illustrate the main features of the toolbox listed above as well as its broad applicability. The examples feature a system with Zeno behavior, of a system with zero-crossing event detection, and of interconnections of hybrid systems with multiple jumps at the same instant.

2. HYEQ: A TOOLBOX FOR SIMULATION OF HYBRID EQUATIONS

The HyEQ Toolbox provides a set of Matlab/Simulink scripts to numerically compute and plot the trajectories to hybrid systems given in terms of hybrid equations as in (1). Given an input u, a trajectory (or solution) to $\mathcal{H}$ is conveniently defined as a function[2]

$$x : \operatorname{dom} x \to \mathbb{R}^n \tag{2}$$

parameterized by ordinary time $t \in \mathbb{R}_{\geq 0}$ and discrete time $j \in \{0, 1, \ldots\} =: \mathbb{N}$, where $\operatorname{dom} x \subset \mathbb{R}_{\geq 0} \times \mathbb{N}$ defines the *hybrid time domain* of x and has the following structure: there exists a sequence of times $0 = t_0 \leq t_1 \leq \ \ldots \ \leq t_j \leq \ldots$ such that $\operatorname{dom} x$ is the union of (possibly infinitely) many intervals of flow time $[t_j, t_{j+1}]$ indexed by j when $t_{j+1} > t_j$ (i.e., of intervals of the form $[t_j, t_{j+1}] \times \{j\}$ with $[t_j, t_{j+1}]$ of nonzero length) and of jump instants $[t_j, t_{j+1}] \times \{j\}$ when $t_{j+1} = t_j$, with the last interval of flow possibly of the form $[t_j, t_{j+1}) \times \{j\}$. During flows, the trajectory x satisfies the continuous dynamics

$$\dot{x} = f(x, u) \quad (x, u) \in C \tag{3}$$

over the intervals of flow while, at jumps, it satisfies the discrete dynamics

$$x^+ = g(x, u) \quad (x, u) \in D. \tag{4}$$

Over a finite amount of flow and a finite number of jumps, the HyEQ Toolbox computes an approximation of the trajectory x in (2) by evaluating the flow condition $x \in C$ and the jump condition $x \in D$, and according to the result of this evaluation, by appropriately discretizing the differential equation defining the flows in (3) or computing the new value of the state after jumps using (4). In this way, the HyEQ Toolbox returns a discrete version of x and its hybrid time domain $\operatorname{dom} x$. More precisely, given an input u, the computed version of the trajectory x is denoted

$$x_s : \operatorname{dom} x_s \to \mathbb{R}^n,$$

which we call *a simulated trajectory* of $\mathcal{H}$, and satisfies

$$x_s^+ = f_s(x_s, u_s) \quad (x_s, u_s) \in C \tag{5}$$

over the intervals of flow and, at jumps, satisfies the discrete dynamics

$$x_s^+ = g(x_s, u_s) \quad (x_s, u_s) \in D. \tag{6}$$

The input u_s is the discretization of u. The function f_s is the resulting discretized flow map obtained when employing an integration scheme for the differential equation $\dot{x} = f(x, u)$. For instance, when the integration scheme is given by the forward Euler integration scheme, $f_s(x_s, u_s) = x_s + sf(x_s, u_s)$ with $s > 0$ denoting the step size for integration. Formal definitions of simulated trajectories (or solutions) and dynamical properties of the discretization (5)-(6) of $\mathcal{H}$ can be found in [15].

Within this framework for simulation, the HyEQ Toolbox includes two scripts to compute trajectories to $\mathcal{H}$: 1) a Matlab script for simulation of hybrid equations within Matlab's workspace, called *Lite HyEQ Simulator*, and 2) a Simulink library and associated Matlab scripts for simulation of hybrid equations within Simulink, called *HyEQ Simulator*. Next, we describe these scripts as well as common plotting functions.

2.1 Lite HyEQ Simulator

A way to compute the trajectories of hybrid equations is to use command-line ODE function calls with events that reset the state according to the discrete dynamics of the system. The Lite HyEQ Simulator computes x_s in this manner and uses four Matlab functions to implement the data of the hybrid system (1) without external input u. The Matlab functions employed by this simulator are defined as follows:

i) The flow map is defined in the Matlab function `f.m`. The input to this function is a vector with components defining the state of the system x_s. Its output is the value of the flow map f evaluated at x_s.
ii) The flow set is defined in the Matlab function `C.m`. The input to this function is a vector with components defining the state of the system x_s. Its output is equal to 1 if the state belongs to the set C or equal to 0 otherwise.
ii) The jump map is defined in the Matlab function `g.m`. Its input is a vector with components defining the state of the system x_s. Its output is the value of the jump map g evaluated at x_s.
iv) The jump set is defined in the Matlab function `D.m`. Its input is a vector with components defining the state of the system x_s. Its output is equal to 1 if the state belongs to D or equal to 0 otherwise.

The Matlab function `HyEQsolver.m` implements the simulation of a hybrid equation defined by functions `f.m, C.m, g.m,` and `D.m` implementing the data. `HyEQsolver.m` uses these functions to integrate the differential equation during continuous evolution and to execute the jump map when jumps shall occur. To do this, the algorithms in `HyEQsolver.m` check at each integration step if x_s is in the set C, D, or neither. Depending on which set x_s is in, the simulation is accordingly reset following the dynamics given in f or g, or the simulation is stopped, respectively. `HyEQsolver.m` is configured to call ODE45, but a different ODE solver can be configured similarly.

The syntax to perform a simulation using `HyEQsolver.m` is as follows:

```
[t j xs] = HyEQsolver(@f,@g,@C,@D,x0,TSPAN,
                      JSPAN,rule,options);
```

The function call has arguments given by the data implemented as Matlab functions as well as simulation parameters. The $n \times 1$ vector `x0` defines the initial condition. The 2×1 parameter `TSPAN = [TSTART TFINAL]` defines the initial and final values of the flow variable t, i.e., the *continuous horizon* and the 2×1 parameter `JSPAN = [JSTART`

[2]See [9] for formal definition of this function as a hybrid arc.

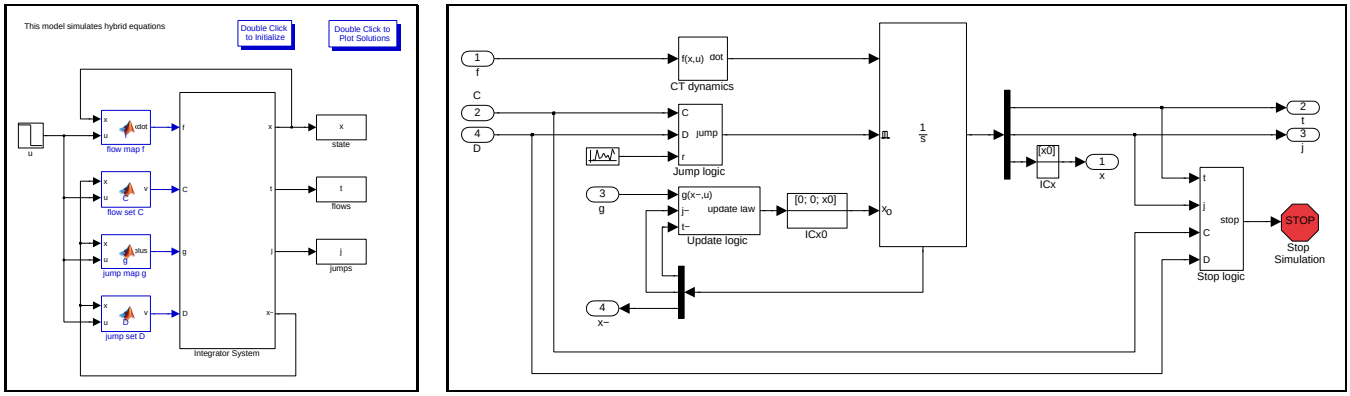


Figure 1: Matlab/Simulink implementation of a hybrid system $\mathcal{H} = (C, f, D, g)$ with inputs (left). Internals of integrator system (right).

`JFINAL]` defines the initial and final values of the jump index j, i.e., the *discrete horizon*. The simulation stops when either one of the horizons ends, `TFINAL` or `JFINAL`, is reached (or exceeded). The scalar parameter `rule` defines whether the simulator gives priority to jumps (`rule`$= 1$), priority to flows (`rule`$= 2$), or no priority (`rule`$= 3$) when both $x \in C$ and $x \in D$ hold. When no priority is selected, then the simulator selects flowing or jumping randomly. The parameter `options` configures the relative tolerance, maximum integration step allowed, and other knobs of the ODE solver. The Matlab function `odeset` can be used to define this parameter; e.g., `options = odeset('RelTol',1e-6,'MaxStep',.1)` sets the relative tolerance to 10^{-6} and the maximum integration step to 0.1.

The `HyEQsolver.m` returns the computed state x_s along with the (discretized) hybrid time domain $\mathrm{dom}\, x_s$. The Matlab script `run.m` is provided to initialize these parameters and run the Lite HyEQ solver. This script can also be used to plot the computed trajectories after the simulation is complete.

2.2 HyEQ Simulator

Trajectories to hybrid equations with inputs are computed by implementing the ideas in the Lite HyEQ simulator of Section 2.1 within Simulink. This permits the use of Simulink libraries for the generation of signals to be used as inputs. It also permits the computation of trajectories to hybrid equations with interconnections defined using Simulink's user interface. Figure 1 depicts a diagram of the HyEQ Simulator (left) and its internals (right). The HyEQ Simulator computes x_s using an integrator block with external reset and state port enabled, and uses four Embedded Matlab function blocks to implement the data of the hybrid system (1) with external input u. More precisely:

i) The flow map f is implemented in the Embedded Matlab function block "flow map f." Its input is a vector with components defining the state of the system x and the input u. Its output is the value of the flow map f which is connected to the input of an integrator.

ii) The flow set C is implemented in the Embedded Matlab function block "flow set C." Its input is a vector with components x^- and input u [3]. Its output is equal to "true" if the state belongs to the set C or equal to "false" otherwise.

iii) The jump map g is implemented in the Embedded Matlab function block "jump map g." Its input is a vector with components x^- and input u of the *Integrator system*. Its output is the value of the jump map g.

iv) The jump set is implemented in an *Embedded Matlab function block* executing the function `D.m`. Its input is a vector with components x^- and input u. Its output is equal to "true" if the state belongs to the set D or equal to "false" otherwise.

The integrator block integrates $\dot{x} = f(x, u)$ to get the state trajectory from a given initial condition `x0`. This block also computes the (discretized) hybrid time domain associated with the state trajectory. The internal components of this block are shown in Figure 1 (right) and are described next.

2.2.1 CT Dynamics

This block defines the continuous-time (CT) dynamics by assembling the time derivative of t, j, and x. The parameters t and j are considered as states of the integrator and, in this way, are updated throughout the simulation so as to keep track of the flow time and number of jumps. The "CT dynamics" block implements the differential equation

$$\dot{t} = 1, \qquad \dot{j} = 0, \qquad \dot{x} = f(x, u),$$

where u is provided by an external signal generator (see left of Figure 1).

2.2.2 Jump Logic

This block uses the outputs of the Embedded Matlab function blocks "flow set C" and "jump set D," which indicate whether the state and input are in those sets or not, and a random signal r with uniform distribution in $[0, 1]$. Figure 2 shows the Simulink blocks used to implement the logic for jumps. As defined in Section 2.1, the variable *rule* defines whether the simulator gives priority to jumps, priority to flows, or no priority. The output of the "Jump logic" block is equal to one when either

- the output of the "jump set D" block is equal to one and `rule`$= 1$;

[3] The minus notation denotes the previous value of the state (before integration). In Simulink, the value x^- is obtained from the state port of the integrator.

- the output of the "flow set C" block is equal to zero, the output of the "jump set D" block is equal to one, and `rule`= 2;
- the output of the "flow set C" block is equal to zero, the output of the "jump set D" block is equal to one, and `rule`= 3; or
- the output of the "flow set C" block is equal to one, the output of the "jump set D" block is equal to one, `rule`= 3, and the random signal r is larger or equal than 0.5.

Under any of these events, the output of the "Jump logic" block, which is connected to the integrator's external reset input, triggers a reset of the integrator, that is, a jump of the hybrid equation being simulated. The reset or jump is activated since the configuration of the reset input is set to "level hold," which executes resets when this external input is equal to one (if the next input remains set to one, multiple resets would be triggered). Otherwise, the output is equal to zero.

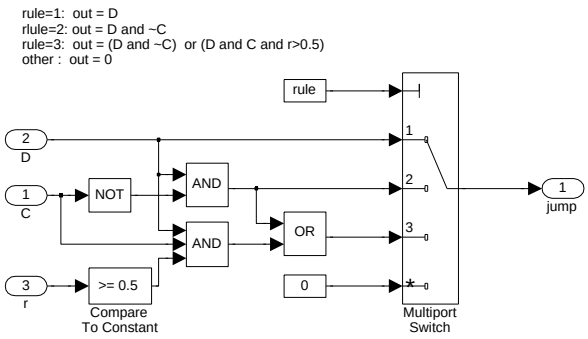


Figure 2: Implementation of the logic for jumps within "Jump logic" block.

2.2.3 Update Logic

The "Update logic" block makes use of the *state port* of the integrator to reset the state. This port reports the value of the state of the integrator at the exact instant that the reset condition becomes true. Notice that x^- differs from x since at a jump, x^- indicates the value of the state that triggers the jump, but it is never assigned as the output of the integrator. In other words, "$x \in D$" is checked using x^- and, if true, x is reset to $g(x^-, u)$. Notice, however, that u is the same because at a jump, u indicates the next evaluated value of the input, and it is assigned as the output of the integrator. The flow time t is kept constant at jumps and j is incremented by one. More precisely, the "Update logic" block executes the following reset law

$$t^+ = t^-, \qquad j^+ = j^- + 1, \qquad x^+ = g(x^-, u),$$

where (x^-, u) is the value that triggers the jump.

2.2.4 Stop Logic

This block stops the simulation under any of the following events:

- The flow time is larger than or equal to the maximum flow time specified by `TFINAL`.
- The jump time is larger than or equal to the maximum number of jumps specified by `JFINAL`.
- The state of the hybrid system x is neither in C nor in D.

Under any of these events, the output of the logic operator connected to the "Stop logic" block becomes one, stopping the simulation. Note that the inputs C and D are routed from the output of the blocks computing whether the state is in C or D and use the value of x^-.

2.3 Initialization and postprocessing functions

The following functions are used to generate plots:

- `plotflows(t,j,x)`: plots (in blue) the projection of the trajectory x onto the flow time axis t. The value of the trajectory for intervals $[t_j, t_{j+1}]$ with empty interior is marked with $*$ (in blue). Dashed lines (in red) connect the value of the trajectory before and after the jump.
- `plotjumps(t,j,x)`: plots (in red) the projection of the trajectory x onto the jump time j. The initial and final value of the trajectory on each interval $[t_j, t_{j+1}]$ is denoted by $*$ (in red) and the continuous evolution of the trajectory on each interval is depicted with a dashed line (in blue).
- `plotHybridArc(t,j,x)`: plots (in black) the trajectory x on hybrid time domains. The intervals $[t_j, t_{j+1}]$ indexed by the corresponding j are depicted in the $t - j$ plane (in red).

An initialization file is used to define variables needed for a simulation. Also, a post-processing file is used to plot the simulated solutions after a simulation is complete. These two `.m` files are called by double-clicking the *Double Click to...* blocks at the top of the HyEQ Simulator (left in Figure 1).

3. EXAMPLES

Next, we illustrate the HyEQ Toolbox in several hybrid systems modeled as hybrid equations. The files associated with the examples below are available at the software entry at http://www.u.arizona.edu/∼sricardo/.

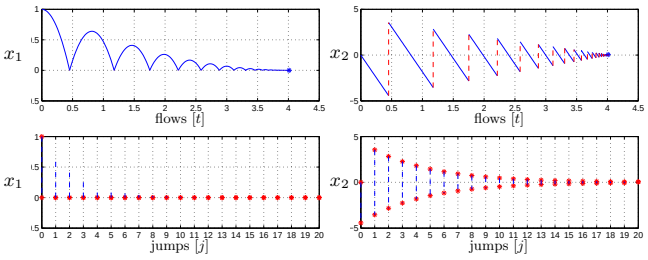
3.1 A system with Zeno behavior

Consider the model of a bouncing ball given by a hybrid equation with data

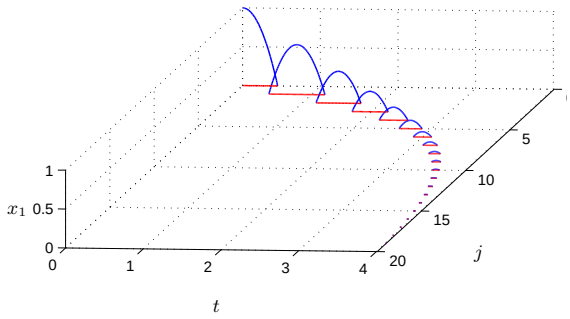
$$f(x) := \begin{bmatrix} x_2 \\ -\gamma \end{bmatrix}, \; C := \left\{ x \in \mathbb{R}^2 \mid x_1 \geq 0 \right\}$$

$$g(x) := \begin{bmatrix} 0 \\ -\lambda x_2 \end{bmatrix}, \; D := \left\{ x \in \mathbb{R}^2 \mid x_1 \leq 0 \, , \; x_2 \leq 0 \right\}$$

where $\gamma > 0$ is the gravity constant and $\lambda \in [0, 1)$ is the restitution coefficient. This system models a ball bouncing on a floor at zero height. The following procedure is used to simulate this system in the Lite HyEQ Simulator: 1) Inside the Matlab script `run.m`, initial conditions, simulation horizons, a rule for jumps, ode solver options, and a step size coefficient are defined. The function `HyEQsolver` is called in order to run the simulation, and a script for plotting simulated solutions is included; 2) Then, the Matlab functions `f.m, C.m, g.m, D.m` are edited according to the data given above; 3) Finally, the simulation is run by calling `run.m` in the Matlab command window. A simulated



(a) Height (b) Velocity



(c) Simulated solution vs. (t, j)

Figure 3: Simulated solution to system in Section 3.1.

solution to the bouncing ball system from `x0`$= [1, 0]^\top$ and with `TSPAN`$= [0\ \ 10]$, `JSPAN`$= [0\ \ 20]$, `rule`$= 1$, $\gamma = 9.81$, and $\lambda = 0.8$ is depicted in Figure 3(a) (height) and Figure 3(b) (velocity). Both the projection onto t using `plotflows` and onto j using `plotjumps` are shown. Figure 3(c) depicts the corresponding hybrid arc for the position state using `plotHybridArc`.

3.2 A system with special event detection

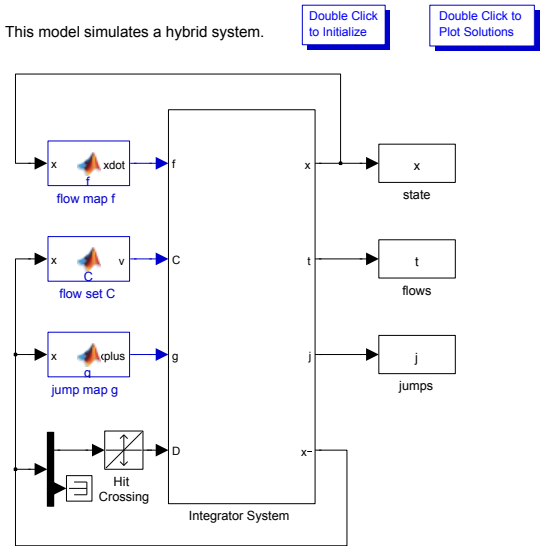
Consider the hybrid equation with data

$$f(x) := \begin{bmatrix} -x_2 \\ 0 \end{bmatrix}, \quad C := \{x \in \mathbb{R} \times \{-1, 1\} \mid x_1 > 0\} \cup \{x \in \mathbb{R} \times \{-1, 1\} \mid x_1 < 0\},$$

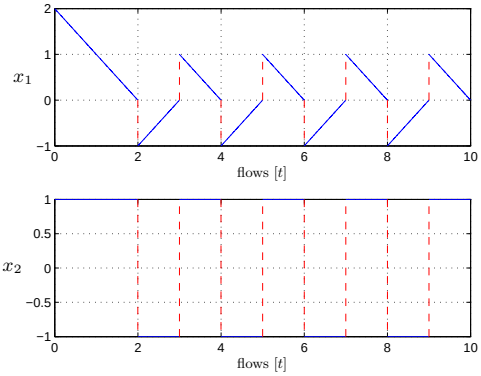
$$g(x) := \begin{bmatrix} -x_2 \\ -x_2 \end{bmatrix}, \quad D := \{x \in \mathbb{R} \times \{-1, 1\} \mid x_1 = 0\}.$$

The trajectories to this system are such that x_1 flows according to $\dot{x}_1 = 1$ when $x_2 = -1$ and according to $\dot{x}_1 = -1$ when $x_2 = 1$, and, when x_1 hits zero, then x_1 is reset to 1 or to -1, respectively. Unfortunately, the jump condition is fragile and numerical errors would prevent from being satisfied.

So that the simulated solution jumps when x_1 crosses zero, a zero-crossing detection algorithm could be used. Figure 4 shows a construction of the jump set that exploits the capabilities of Simulink's block *Hit Crossing*, which will not miss the crossing of x_1 by zero and, hence, trigger a jump. A trajectory for this system from `x0`$= [2, 1]$ is depicted in Figure 4(b) (projected onto t and j) with jumps at the correct instants.



(a) HyEQ Simulator with *Hit Crossing* block.



(b) Simulated solution with *Hit Crossing* block.

Figure 4: HyEQ Simulator using zero-crossing detection to detect jumps and a trajectory.

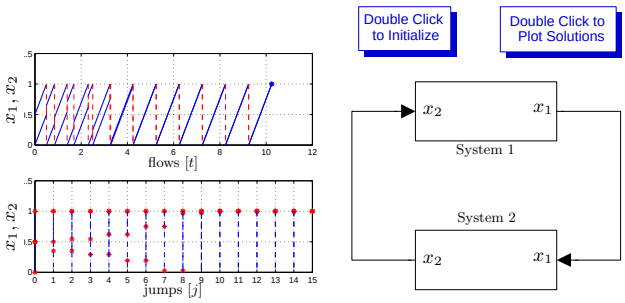
3.3 An interconnected system w/multiple jumps

Consider the synchronization of two impulse-coupled oscillators. The timers within each oscillator are modeled as periodic oscillators with timer state x_i, $i = 1, 2$, evolving on the unitary interval $[0, 1]$. When the i-th state reaches $x_i = 1$, the said state is reset to zero but the state of the other timer is reset via $x_j^+ = \max\{1, (1+\varepsilon)x_j\}$, where $\varepsilon > 0$ is a constant coefficient. It can be shown that the timers converge to each other asymptotically, for almost every initial condition [9]. This model has been used in the literature to model impulse-coupled oscillators in nature (fireflies, crickets, Parkinson's disease, etc.). Each timer can be modeled

by the hybrid equations given by $f_i(x_i, u_i) := 1$,

$$
\begin{aligned}
C_i &:= \left\{(x_i, u_i) \in \mathbb{R}^2 \mid 0 \le x_i \le 1\right\} \cap \\
&\qquad \left\{(x_i, u_i) \in \mathbb{R}^2 \mid 0 \le u_i \le 1\right\}, \\
g_i(x_i, u_i) &:= \begin{cases} (1+\varepsilon)x_i & \text{if } (1+\varepsilon)x_i < 1 \\ 0 & \text{if } (1+\varepsilon)x_i \ge 1, \end{cases} \\
D_i &:= \left\{(x_i, u_i) \in \mathbb{R}^2 \mid x_i = 1\right\} \cup \\
&\qquad \left\{(x_i, u_i) \in \mathbb{R}^2 \mid u_i = 1\right\}.
\end{aligned}
$$

The interaction between the timers can be modeled as the interconnection between a copy of this system for $i = 1$ and another copy for $i = 2$, with the interconnection relationship $u_1 = x_2$ and $u_2 = x_1$. In this way, the reset of one timer affects the dynamics of the other timer. A trajectory to this interconnection is obtained by coding the system above in the HyEQ Simulator, converting it into a subsystem, and then duplicating the system and interconnecting them within a common Simulink file; see Figure 5(b)[4]. A simulated solution for $\varepsilon = 0.3$ is depicted in Figure 5(a). It confirms the expected behavior of the interconnected hybrid equations: the timers initially evolve out of phase and progressively synchronize their resets. Once the timers are synchronized, two jumps at the same instant occur.



(a) Simulated solution (b) Interconnection

Figure 5: Simulated solution to system and the interconnected system in Section 3.3.

4. CONCLUSIONS

The Hybrid Equations (HyEQ) Toolbox for simulation of hybrid systems modeled as hybrid equations was described and illustrated in examples. A webinar presenting this toolbox and illustrating it in examples has been prepared and is currently under review by Mathworks before it goes live at their website. The toolbox has been tested for the past five years, including graduate courses taught at the University of Arizona and at the University of California Santa Barbara, and at short courses and workshops at the International EECI Graduate School on Control, CDC, and ACC.

5. REFERENCES

[1] A. Abate, A. D. Ames, and S. S. Sastry. Error bounds based stochastic approximations and simulations of hybrid dynamical systems. In *Proc. 25th American Control Conference*, pages 4742–4747, 2006.

[4]Furthermore, a complex system with up to twenty five impulse-coupled oscillators is included in the examples that come with the toolbox.

[2] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proc. of the IEEE*, 91:11–28, 2003.

[3] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.

[4] M. K. Camlibel, W.P.M.H. Heemels, and J.M.H. Schumacher. Consistency of a time-stepping method for a class of piecewise-linear networks. *IEEE Trans. Circ. Syst. I*, 49(3):349–357, 2002.

[5] D. A. Copp and R. G. Sanfelice. On the effect and robustness of zero-crossing detection algorithms in simulation of hybrid systems jumping on surfaces. In *Proceedings of the American Control Conference*, pages 2449–2454, 2012.

[6] H. Elmqvist, S.E. Mattsson, and M. Otter. Modelica: The new object-oriented modeling language. In *Proc. of 12th European Simulation Multiconference*, 1998.

[7] J.M. Esposito, V. Kumar, and G.J. Pappas. Accurate event detection for simulating hybrid systems. In *Hybrid Systems: Computation and Control: 4th International Workshop*, pages 204–217, 2001.

[8] R. Goebel, R. G. Sanfelice, and A. R. Teel. *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, New Jersey, 2012.

[9] R. Goebel, R.G. Sanfelice, and A.R. Teel. Hybrid dynamical systems. *IEEE Control Systems Magazine*, pages 28–93, 2009.

[10] K.H. Johansson, J. Lygeros, S. Sastry, and M. Egerstedt. Simulation of Zeno hybrid automata. In *Proc. 38th IEEE Conference on Decision and Control*, volume 4, pages 3538–3543, 1999.

[11] E. A. Lee and H. Zheng. Operational semantics for hybrid systems. In *Hybrid Systems: Computation and Control: 8th International Workshop*, pages 25–53, 2005.

[12] J. Liu and E. A. Lee. A component-based approach to modeling and simulating mixed-signal and hybrid systems. *ACM Transactions on Modeling and Computer Simulation*, 12(4):343–368, 2002.

[13] J. Liu, X. Liu, T. J. Koo, B. Sinopoli, S.Sastry, and E. A. Lee. A hierarchical hybrid system model and its simulation. In *Proc. 38th IEEE Conf. Dec. Contr.*, volume 4, pages 3508 – 3513, 1999.

[14] P.J. Mosterman and G. Biswas. A hybrid modeling and simulation methodology for dynamic physical systems. *Simulation*, 78:5–17, 2002.

[15] R. G. Sanfelice and A. R. Teel. Dynamical properties of hybrid systems simulators. *Automatica*, 46(2):239–248, 2010.

[16] A. M. Stuart and A.R. Humphries. *Dynamical Systems and Numerical Analysis*. Cambridge University Press, 1996.

[17] F.D. Torrisi and A. Bemporad. HYSDEL - A tool for generating computational hybrid models for analysis and synthesis problems. *IEEE Trans. Control Systems Technology*, 12:235–249, 2004.