Hybrid Equations (HyEQ) Toolbox v2.02

A Toolbox for Simulating Hybrid Systems in MATLAB/Simulink®

Ricardo G. Sanfelice
University of California
Santa Cruz, CA 95064
USA

David A. Copp
University of California
Santa Barbara, CA 93109
USA

Pablo Nanez
Universidad de Los Andes
Colombia

October 30, 2014

Abstract

This note describes the Hybrid Equations (HyEQ) Toolbox implemented in MATLAB/Simulink for the simulation of hybrid dynamical systems. This toolbox is capable of simulating individual and interconnected hybrid systems where multiple hybrid systems are connected and interact such as a bouncing ball on a moving platform, fireflies synchronizing their flashing, and more. The Simulink implementation includes four basic blocks that define the dynamics of a hybrid system. These include a flow map, flow set, jump map, and jump set. The flows and jumps of the system are computed by the integrator system which is comprised of blocks that compute the continuous dynamics of the hybrid system, trigger jumps, update the state of the system and simulation time at jumps, and stop the simulation. We also describe a "lite simulator" which allows for faster simulation.

Contents

1 Introduction

To get started, a webinar introducing the HyEQ Toolbox is available at
`http://www.mathworks.com/videos/hyeq-a-toolbox-for-simulation-of-hybrid-dynamical-systems-81992.html`
A free two-step registration is required by Mathworks.

A hybrid system is a dynamical system with continuous and discrete dynamics. Several mathematical models for hybrid systems have appeared in literature. In this paper, we consider the framework for hybrid systems used in [3,4], where a hybrid system $\mathcal{H}$ on a state space $\mathbb{R}^n$ with input space $\mathbb{R}^m$ is defined by the following objects:

- A set $C \subset \mathbb{R}^n \times \mathbb{R}^m$ called the *flow set*.
- A function $f\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ called the *flow map*.
- A set $D \subset \mathbb{R}^n \times \mathbb{R}^m$ called the *jump set*.
- A function $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ called the *jump map*.

We consider the simulation in MATLAB/Simulink of hybrid systems $\mathcal{H} = (C, f, D, g)$ written as

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

The flow map f defines the continuous dynamics on the flow set C, while the jump map g defines the discrete dynamics on the jump set D. These objects are referred to as the *data* of the hybrid system $\mathcal{H}$, which at times is explicitly denoted as $\mathcal{H} = (C, f, D, g)$. We illustrate this framework in a simple, yet rich in behavior, hybrid system.

Example 1.1 (bouncing ball system) Consider a model for a bouncing ball written as

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

$$g(x) := \begin{bmatrix} 0 \\ -\lambda x_2 \end{bmatrix}, D := \left\{x \in \mathbb{R}^2 \mid x_1 \leq 0\ ,\ x_2 \leq 0\right\} \tag{3}$$

where $\gamma > 0$ is the gravity constant and $\lambda \in [0, 1)$ is the restitution coefficient. In this model, we consider the ball to be bouncing on a floor at a height of 0. This model is re-visited as an example in Section 3 and Section 5. □

The remainder of this note is organized as follows. In Section 2, we describe how to install the HyEQ Toolbox in MATLAB. In Section 3, we introduce the Lite HyEQ Simulator for solving hybrid systems without inputs. In Section 4, we introduce the HyEQ Simulator implemented in Simulink for solving single and interconnected hybrid systems with inputs. In Section 5, we work through several examples for the simulation of single and interconnected hybrid systems. In Section 6, we give directions to where the simulator files can be downloaded.

2 Installation

The following procedure describes how to install the Hybrid Equations (HyEQ) Toolbox in MATLAB. This installation adds useful `.m` files to the MATLAB library and several blocks to the Simulink block library.

Steps for installation:

1. Download the HyEQ Toolbox from MATLAB Central or the author's website at `https://hybrid.soe.ucsc.edu/software`.
2. Extract all files and save in any place (except the root folder).
3. Open MATLAB and change the current folder to the folder where the `install.m` is located.
4. Type `install` in the command window and hit enter to run the file `install.m`.
5. Follow the on-screen prompts. Must answer yes to the question:

   ```
   Add toolbox permanently into your startup path (highly recommended)? Y/E/N [Y]: y
   ```

6. Once installation has finished, close and then reopen MATLAB.

Now the HyEQ Toolbox is ready for use.

If you wish to uninstall the HyEQ Toolbox from MATLAB, simply run the `tbclean.m` file inside the `HyEQ_Toolbox_V2_02` folder, and follow the on-screen prompts.

3 Lite HyEQ Simulator: A stand-alone MATLAB code for simulation of hybrid systems without inputs

One way to simulate hybrid systems is to use ODE function calls with events in MATLAB (see, e.g., [2]). Such an implementation gives fast simulation of a hybrid system.

In the lite HyEQ solver, four basic functions are used to define the *data* of the hybrid system $\mathcal{H}$ as in (1) (without inputs):

- 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. Its output is the value of the flow map f.
- 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. Its output is equal to 1 if the state belongs to the set C or equal to 0 otherwise.
- 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. Its output is the value of the jump map g.
- 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. Its output is equal to 1 if the state belongs to D or equal to 0 otherwise.

Our Lite HyEQ Simulator uses a main function `run.m` to initialize, run, and plot solutions for the simulation, functions `f.m, C.m, g.m,` and `D.m` to implement the data of the hybrid system, and `HyEQsolver.m` which will solve the differential equations by integrating the continuous dynamics, $\dot{x} = f(x)$, and jumping by the update law $x^+ = g(x)$. The ODE solver called in `HyEQsolver.m` initially uses the initial or most recent step size, and after each integration, the algorithms in `HyEQsolver.m` check to see if the solution is in the set C, D, or neither. Depending on which set the solution is in, the simulation is accordingly reset following the dynamics given in f or g, or the simulation is stopped. This implementation is fast because it also does not store variables to the workspace and only uses built-in ODE function calls.

Time and jump horizons are set for the simulation using `TSPAN = [TSTART TFINAL]` as the time interval of the simulation and `JSPAN = [JSTART   JSTOP]` as the interval for the number of discrete jumps allowed. The simulation stops when either the time or jump horizon, i.e. the final value of either interval, is reached.

The example below shows how to use the HyEQ solver to simulate a bouncing ball.

Example 1.2 (bouncing ball with Lite HyEQ Solver) Consider the hybrid system model for the bouncing ball with data given in Example 1.1.

For this example, we consider the ball to be bouncing on a floor at zero height. The constants for the bouncing ball system are $\gamma = 9.81$ and $\lambda = 0.8$. The following procedure is used to simulate this example in the Lite HyEQ Solver:

- 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.m` is called in order to run the simulation, and a script for plotting solutions is included.
- Then the MATLAB functions `f.m, C.m, g.m, D.m` are edited according to the data given above.
- Finally, the simulation is run by clicking the run button in `run.m` or by calling `run.m` in the MATLAB command window.

Example code for each of the MATLAB files `run.m, f.m, C.m, g.m,` and `D.m` is given below.

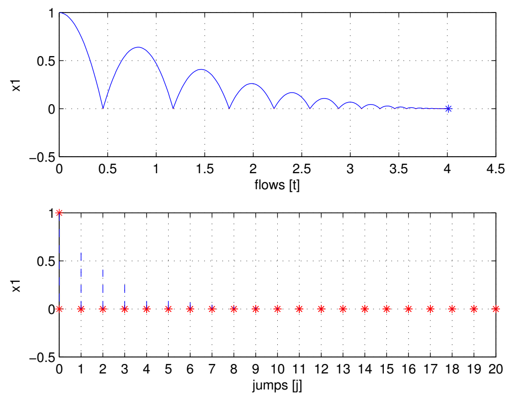
```
function run
% initial conditions
x1_0 = 1;
x2_0 = 0;
x0 = [x1_0;x2_0];
% simulation horizon
TSPAN=[0 10];
JSPAN = [0 20];
% rule for jumps
% rule = 1 -> priority for jumps
% rule = 2 -> priority for flows
rule = 1;
options = odeset('RelTol',1e-6,'MaxStep',.1);
% simulate
[t,j,x] = HyEQsolver(@f,@g,@C,@D,x0,TSPAN,JSPAN,rule,options);
% plot solution
figure(1) % position
clf
subplot(2,1,1),plotflows(t,j,x(:,1))
grid on
ylabel('x1')
subplot(2,1,2),plotjumps(t,j,x(:,1))
grid on
ylabel('x1')
figure(2) % velocity
clf
subplot(2,1,1),plotflows(t,j,x(:,2))
grid on
ylabel('x2')
subplot(2,1,2),plotjumps(t,j,x(:,2))
grid on
ylabel('x2')
% plot hybrid arc
plotHybridArc(t,j,x)
xlabel('j')
ylabel('t')
zlabel('x1')
```

```
function xdot = f(x)
% state
x1 = x(1);
x2 = x(2);
% differential equations
xdot = [x2 ; -9.81];
end
```

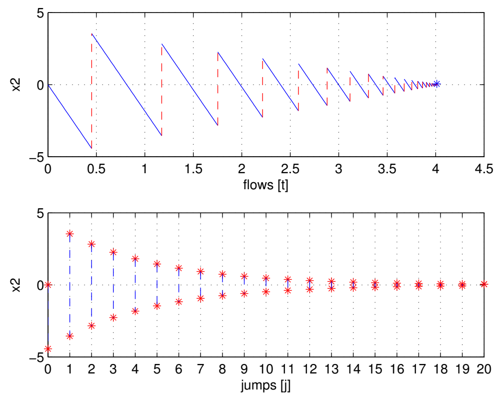
```
function [value discrete] = C(x)
x1 = x(1);
if x1 >= 0
    value = 1;
else
    value = 0;
end
end
```

```
function xplus = g(x)
% state
x1 = x(1);
x2 = x(2);
xplus = [-x1 ; -0.8*x2];
end
```

```
function inside = D(x)
x1 = x(1);
x2 = x(2);
if (x1 <= 0 && x2 <= 0)
    inside = 1;
else
    inside = 0;
end
end
```



(a) Height



(b) Velocity

Figure 1: Solution of Example 1.2

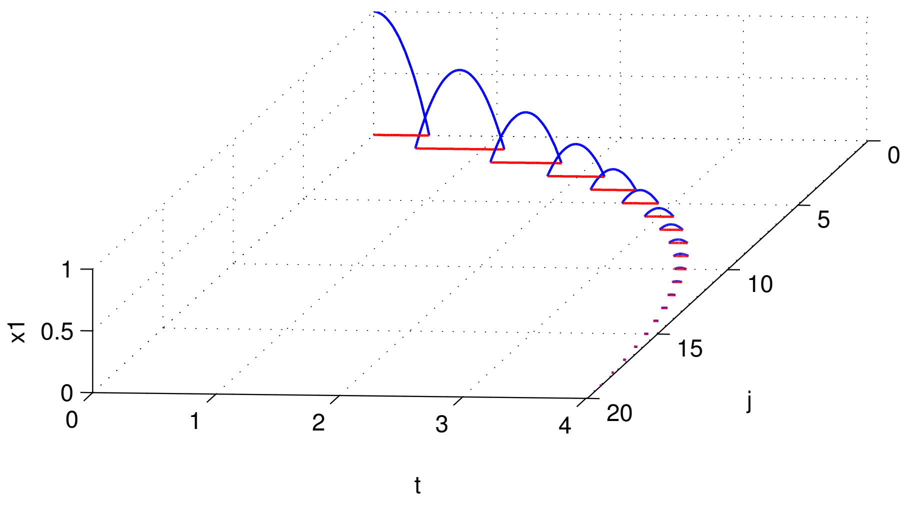


Figure 2: Hybrid arc corresponding to a solution of Example 1.2: height

A solution to the bouncing ball system from $x(0,0) = [1,0]^\top$ and with $TSPAN = [0\ \ 10], JSPAN = [0\ \ 20]$, $rule = 1$, is depicted in Figure 1(a) (height) and Figure 1(b) (velocity). Both the projection onto t and j are shown. Figure 2 depicts the corresponding hybrid arc for the position state.

For MATLAB files of this example, see Examples/Example_1.2.

□

3.1 Solver Function

The solver function `HyEQsolver` solves the hybrid system using three different functions as shown below. First, the flows are calculated using the built-in ODE solver function ODE45 in MATLAB. If the solution leaves the flow set `C`, the discrete event is detected using the function `zeroevents` as shown in Section 3.1.1. When the state jumps, the next value of the state is calculated via the jump map `g` using the function `jump` as shown in Section 3.1.2.

```
function [t j x] = HyEQsolver(f,g,C,D,x0,TSPAN,JSPAN,rule,options)
%HYEQSOLVER solves hybrid equations.
%   Syntax: [t j x] = HyEQsolver(f,g,C,D,x0,TSPAN,JSPAN,rule,options)
%   computes solutions to the hybrid equations
%
%   \dot{x} = f(x)   x \in C x^+ = g(x)   x \in D
%
%   where x is the state, f is the flow map, g is the jump map, C is the
%   flow set, and D is the jump set. It outputs the state trajectory (t,j)
%   -> x(t,j), where t is the flow time parameter and j is the jump
%   parameter.
```

```
%
%   x0 defines the initial condition for the state.
%
%   TSPAN = [TSTART TFINAL] is the time interval. JSPAN = [JSTART JSTOP] is
%       the interval for discrete jumps. The algorithm stop when the first
%       stop condition is reached.
%
%   rule for jumps
%       rule = 1 (default) -> priority for jumps rule = 2 -> priority for
%       flows
%
%   options - options for the solver see odeset f.ex.
%       options = odeset('RelTol',1e-6);
%
%         Example: Bouncing ball with Lite HyEQ Solver
%
%         % Consider the hybrid system model for the bouncing ball with data given in
%         % Example 1.2. For this example, we consider the ball to be bouncing on a
%         % floor at zero height. The constants for the bouncing ball system are
%         % \gamma=9.81 and \lambda=0.8. The following procedure is used to
%         % simulate this example in the Lite HyEQ Solver:
%
%         % * Inside the MATLAB script run_ex1_2.m, initial conditions, simulation
%         % horizons, a rule for jumps, ode solver options, and a step size
%         % coefficient are defined. The function HyEQsolver.m is called in order to
%         % run the simulation, and a script for plotting solutions is included.
%         % * Then the MATLAB functions f_ex1_2.m, C_ex1_2.m, g_ex1_2.m, D_ex1_2.m
%         % are edited according to the data given below.
%         % * Finally, the simulation is run by clicking the run button in
%         % run_ex1_2.m or by calling run_ex1_2.m in the MATLAB command window.
%
%         % For further information, type in the command window:
%         helpview(['Example_1_2.html']);
%
%         % Define initial conditions
%         x1_0 = 1;
%         x2_0 = 0;
%         x0   = [x1_0; x2_0];
%
%         % Set simulation horizon
%         TSPAN = [0 10];
%         JSPAN = [0 20];
%
%         % Set rule for jumps and ODE solver options
%         %
%         % rule = 1 -> priority for jumps
%         %
%         % rule = 2 -> priority for flows
%         %
%         % set the maximum step length. At each run of the
%         % integrator the option 'MaxStep' is set to
%         % (time length of last integration)*maxStepCoefficient.
%         %  Default value = 0.1
%
```

```
%          rule                = 1;
%
%          options             = odeset('RelTol',1e-6,'MaxStep',.1);
%
%          % Simulate using the HyEQSolver script
%          % Given the matlab functions that models the flow map, jump map,
%          % flow set and jump set (f_ex1_2, g_ex1_2, C_ex1_2, and D_ex1_2
%          % respectively)
%
%          [t j x] = HyEQsolver( @f_ex1_2,@g_ex1_2,@C_ex1_2,@D_ex1_2,...
%              x0,TSPAN,JSPAN,rule,options);
%
%          % plot solution
%
%          figure(1) % position
%          clf
%          subplot(2,1,1),plotflows(t,j,x(:,1))
%          grid on
%          ylabel('x1')
%
%          subplot(2,1,2),plotjumps(t,j,x(:,1))
%          grid on
%          ylabel('x1')
%
%          figure(2) % velocity
%          clf
%          subplot(2,1,1),plotflows(t,j,x(:,2))
%          grid on
%          ylabel('x2')
%
%          subplot(2,1,2),plotjumps(t,j,x(:,2))
%          grid on
%          ylabel('x2')
%
%          % plot hybrid arc
%
%          plotHybridArc(t,j,x)
%          xlabel('j')
%          ylabel('t')
%          zlabel('x1')
%
%          % plot solution using plotHarc and plotHarcColor
%
%          figure(4) % position
%          clf
%          subplot(2,1,1), plotHarc(t,j,x(:,1));
%          grid on
%          ylabel('x_1 position')
%          subplot(2,1,2), plotHarc(t,j,x(:,2));
%          grid on
%          ylabel('x_2 velocity')
%
%
%          % plot a phase plane
```

```matlab
%          figure(5) % position
%          clf
%          plotHarcColor(x(:,1),j,x(:,2),t);
%          xlabel('x_1')
%          ylabel('x_2')
%          grid on
%
%--------------------------------------------------------------------------
% Matlab M-file Project: HyEQ Toolbox @ Hybrid Dynamics and Control Lab,
% http://www.u.arizona.edu/~sricardo/index.php?n=Main.Software
% http://hybridsimulator.wordpress.com/
% Filename: HyEQsolver.m
%--------------------------------------------------------------------------
%   See also plotflows, plotHarc, plotHarcColor, plotHarcColor3D,
%   plotHybridArc, plotjumps.
%   Copyright @ Hybrid Dynamics and Control Lab,
%   Revision: 0.0.0.1 Date: 04/23/2014 10:48:24


if ~exist('rule','var')
    rule = 1;
end

if ~exist('options','var')
    options = odeset();
end

% simulation horizon
tstart = TSPAN(1);
tfinal = TSPAN(end);

% simulate
options = odeset(options,'Events',@(t,x) zeroevents(x,C,D,rule));
tout = tstart;
xout = x0.';
jout = JSPAN(1);
j = jout(end);

% Jump if jump is prioritized:
if rule == 1
    while (j<JSPAN(end))
        % Check if value it is possible to jump current position
        insideD = D(xout(end,:).');
        if insideD == 1
            [j tout jout xout] = jump(g,j,tout,jout,xout);
        else
            break;
        end
    end
end
fprintf('Completed: %3.0f%%',0);
while (j < JSPAN(end) && tout(end) < TSPAN(end))
    % Check if it is possible to flow from current position
    insideC = C(xout(end,:).');
    if insideC == 1
        [t,x] = ode45(@(t,x) f(x),[tout(end) tfinal],xout(end,:).', options);
        nt = length(t);
        tout = [tout; t];
        xout = [xout; x];
        jout = [jout; j*ones(1,nt)'];
    end

    %Check if it is possible to jump
    insideD = D(xout(end,:).');
    if insideD == 0
        break;
    else
        if rule == 1
            while (j<JSPAN(end))
                % Check if it is possible to jump from current position
                insideD = D(xout(end,:).');
                if insideD == 1
                    [j tout jout xout] = jump(g,j,tout,jout,xout);
                else
                    break;
                end
            end
        else
            [j tout jout xout] = jump(g,j,tout,jout,xout);
        end
    end
    fprintf('\b\b\b\b%3.0f%%',max(100*j/JSPAN(end),100*tout(end)/TSPAN(end)));
end
t = tout;
x = xout;
j = jout;
fprintf('\nDone\n');
end
```

3.1.1 Events Detection

```
function [value,isterminal,direction] = zeroevents(x,C,D,rule )
isterminal = 1;
direction = -1;
insideC = C(x);
if insideC == 0
    % Outside of C
    value = 0;
elseif (rule == 1)
    % If priority for jump, stop if inside D
    insideD = D(x);
    if insideD == 1
        % Inside D, inside C
        value = 0;
    else
        % outside D, inside C
        value = 1;
    end
else
    % If inside C and not priority for jump or priority of jump and outside
    % of D
    value = 1;
end
end
```

3.1.2 Jump Map

```
function [j tout jout xout] = jump(g,j,tout,jout,xout)
% Jump
j = j+1;
y = g(xout(end,:).');
% Save results
tout = [tout; tout(end)];
xout = [xout; y.'];
jout = [jout; j];
end
```

3.2 Software Requirements

In order to run simulations using the Lite HyEQ Simulator, MATLAB R13 or newer is required.

3.3 Configuration of Solver

Before a simulation is started, it is important to determine the needed integrator scheme, zero-cross detection settings, precision, and other tolerances. Using the default settings does not always give the most efficient or most accurate simulations. In the Lite HyEQ Simulator, these parameters are edited in the `run.m` file using

```
options = odeset(RelTol,1e-6,MaxStep ,.1);.
```

3.4 Initialization

The Lite HyEQ Simulator is initialized and run by calling the function `run.m`. Inside `run.m`, the initial conditions, simulation horizons `TSPAN` and `JSPAN`, a rule for jumps, and simulation tolerances are defined. After all of the parameters are defined, the function `HyEQsolver` is called, and the simulation runs. See below for sample code to initialize and run the bouncing ball example, Example 1.2.

```
% initial conditions
x1_0 = 1;
x2_0 = 0;
x0 = [x1_0;x2_0];
% simulation horizon
TSPAN=[0,10];
JSPAN = [0,20];
% rule for jumps
% rule = 1 -> priority for jumps
% rule = 2 -> priority for flows
rule = 1;
options = odeset('RelTol',1e-6,'MaxStep',.1);
% simulate
```

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

3.5 Postprocessing and Plotting solutions

The function `run.m` is also used to plot solutions after the simulations is complete. See below for sample code to plot solutions to the bouncing ball example, Example 1.2.

```
% plot solution
figure(1) % position
clf
subplot(2,1,1),plotflows(t,j,x(:,1))
grid on
ylabel('x1')
subplot(2,1,2),plotjumps(t,j,x(:,1))
grid on
ylabel('x1')
figure(2) % velocity
clf
subplot(2,1,1),plotflows(t,j,x(:,2))
grid on
ylabel('x2')
subplot(2,1,2),plotjumps(t,j,x(:,2))
grid on
ylabel('x2')
% plot hybrid arc
plotHybridArc(t,j,x)
xlabel('j')
ylabel('t')
zlabel('x1')
```

The following functions are used to generate the 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. Figure 10(a) shows a plot created with this function.

- 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). Figure 10(a) shows a plot created with this function.

- plotHybridArc(t,j,x): plots (in blue and red) 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). Figure 11 shows a plot created with this function.

- plotHarc is a function for plotting hybrid arcs (n states).
  - plotHarc(t,j,x): plots the trajectory x versus the hybrid time domain (t, j). If x is a matrix, then the time vector is plotted versus the rows or columns of the matrix, whichever line up.
  - plotHarc(t,j,x,$jstar$): plots the trajectory x versus the hybrid time domain (t, j), and the plot is cut regarding the $jstar$ interval ($jstar = [j_{initial}, j_{final}]$).
  - plotHarc(t,j,x,$jstar$,modificator): Modificator is a cell array that contains the standard matlab ploting modificators (type >> help plotHarc or >> helpwin plotHarc in the command window for more information).

- plotHarcColor plots the trajectory x (vector) on hybrid time domain with color.
  - plotHarcColor(t,j,x,L): plots the trajectory x (vector) versus the hybrid time domain (t, j). The hybrid arc is plotted with L data as color. The input vectors t, j, x, L must have the same length.
  - plotHarcColor(t,j,x,L,$jstar$): If a specific interval in j is required, $jstar = [j_{initial}, j_{final}]$ must be provided. (type >> help plotHarcColor or >> helpwin plotHarcColor in the command window for more information)
- plotHarcColor3D plots an $3D$ hybrid arc with color.
  - plotHarcColor3D(t,j,x,L) plots the trajectory x (3 states) taking into account the hybrid time domain (t, j). The hybrid arc is plotted with L data as color. The input vectors t, j, x, L must have the same length and x must have three columns.
  - plotHarcColor3D(t,j,x,L,$jstar$) If a specific interval in j is required, $jstar = [j_{initial}, j_{final}]$ must be provided.
  - plotHarcColor3D(t,j,x,L,$jstar$,modificator) Modificator is a cell array that contains the standard matlab ploting modificators (type >> help plotHarcColor3D or >> helpwin plotHarcColor3D in the command window for more information).

4 HyEQ Simulator: A Simulink implementation for simulation of single and interconnected hybrid systems with or without inputs

The HyEQ Toolbox includes three main Simulink library blocks that allow for simulation of a hybrid system $\mathcal{H} = (C, f, D, g)$ using either externally defined functions or embedded MATLAB functions, and a single hybrid system or interconnected hybrid systems with inputs using embedded MATLAB functions. Figure 3 shows these blocks in the Simulink Library Browser.

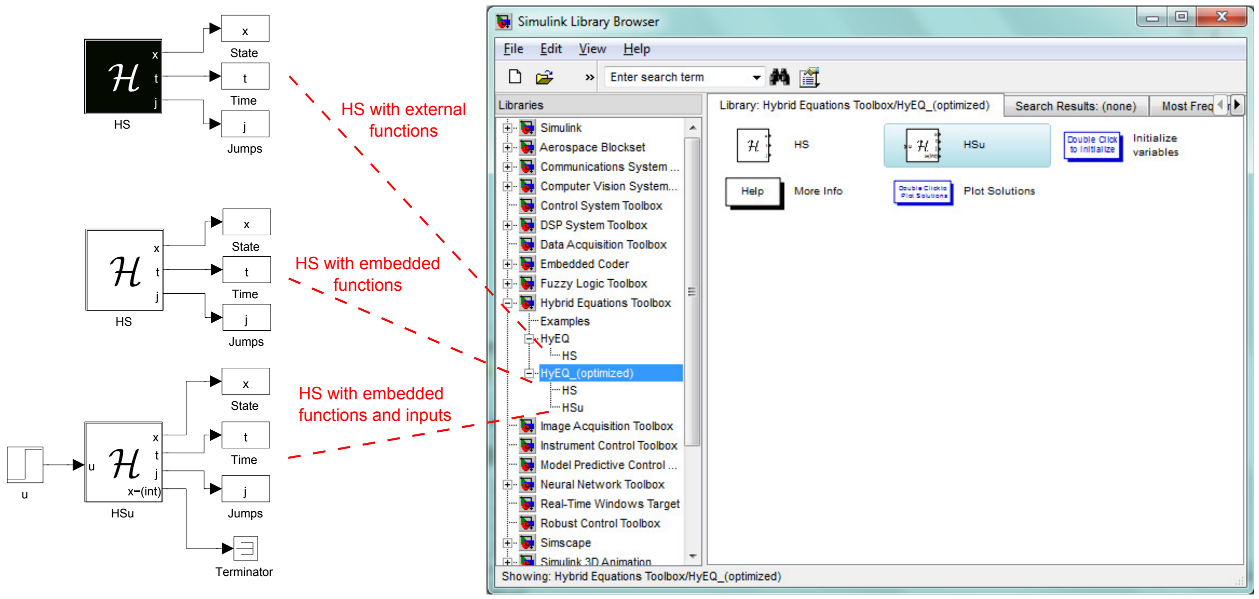


Figure 3: MATLAB/Simulink library blocks for Simulink implementation.

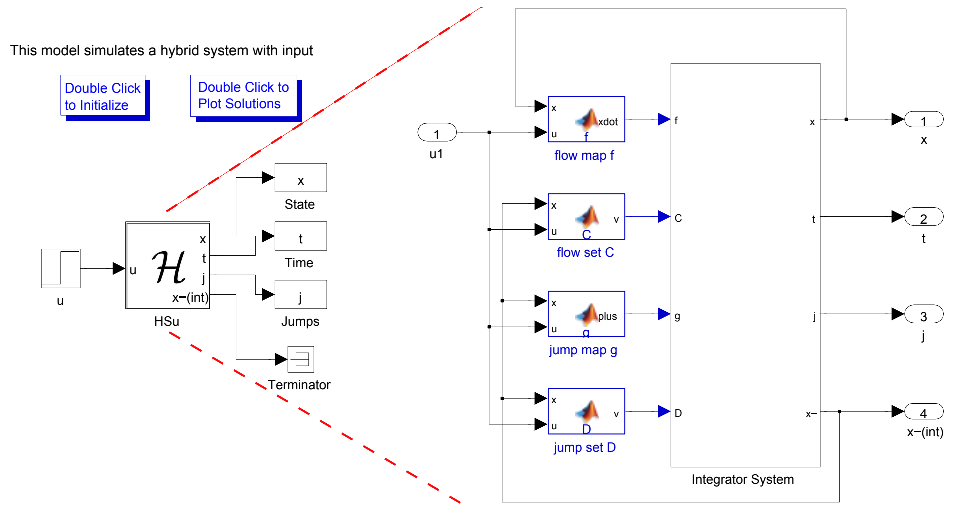


Figure 4: MATLAB/Simulink implementation of a hybrid system $\mathcal{H} = (C, f, D, g)$ with inputs.

Figure 4 shows a Simulink implementation for simulating a hybrid system with inputs using embedded MATLAB functions. In this implementation, four basic blocks are used to define the *data* of the hybrid system $\mathcal{H}$:

- The flow map is implemented in an *Embedded MATLAB function block* executing the function `f.m`. 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.

- The flow set is implemented in an *Embedded MATLAB function block* executing the function `C.m`. Its input is a vector with components x^- and input u of the *Integrator system*. Its output is equal to 1 if the state belongs to the set C or equal to 0 otherwise. The minus notation denotes the previous value of the variables (before integration). The value x^- is obtained from the state port of the integrator.

- The jump map is implemented in an *Embedded MATLAB function block* executing the function `g.m`. 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.

- 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 of the *Integrator system*. Its output is equal to 1 if the state belongs to D or equal to 0 otherwise.

In our implementation, MATLAB `.m` files are used. The file `initialization.m` is used to define initial variables before simulation. The file `postprocessing.m` is used to plot the 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 Simulink Model (see Section 4.5 for more information on these `.m` files and their use).

4.1 The Integrator System

In this section we discuss the internals of the *Integrator System* shown in Figure 5.

4.1.1 CT Dynamics

This block is shown in Figure 6. It defines the continuous-time (CT) dynamics by assembling the time derivative of the state $[t\ j\ x^\top]^\top$. States t and j are considered states of the system because they need to be

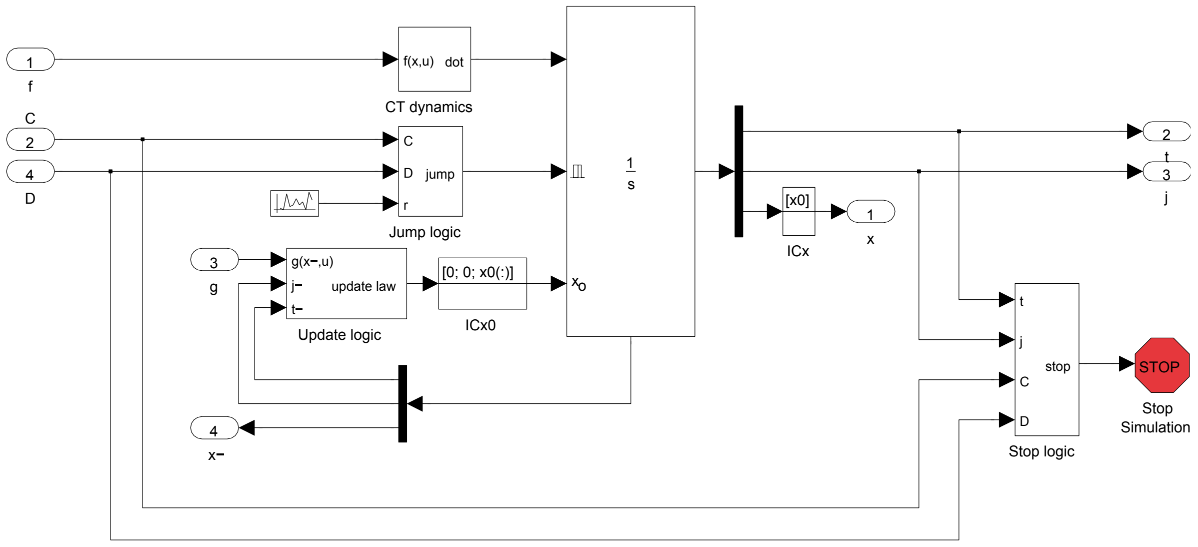


Figure 5: Integrator System

updated throughout the simulation in order to keep track of the time and number of jumps. Without t and j, solutions could not be plotted accurately. This is given by

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

Note that input port 1 takes the value of $f(x, u)$ through the output of the *Embedded MATLAB function block f* in Figure 4.

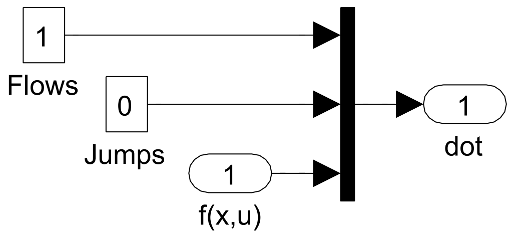


Figure 6: CT dynamics

4.1.2 Jump Logic

This block is shown in Figure 7. The inputs to the jump logic block are the output of the blocks C and D indicating whether the state is in those sets or not, and a random signal with uniform distribution in $[0, 1]$. Figure 7 shows the Simulink blocks used to implement the Jump Logic. The variable *rule* defines whether the simulator gives priority to jumps, priority to flows, or no priority. It is initialized in `initialization.m`.

The output of the Jump Logic is equal to one when:

- the output of the *D block* is equal to one and $rule = 1$,
- the output of the *C block* is equal to zero, the output of the *D block* is equal to one, and $rule = 2$,
- the output of the *C block* is equal to zero, the output of the *D block* is equal to one, and $rule = 3$,
- or the output of the *C block* is equal to one, the output of the *D block* is equal to one, $rule = 3$, and the random signal r is larger or equal than 0.5.

Under these events, the output of this block, which is connected to the integrator external reset input, triggers a reset of the integrator, that is, a jump of $\mathcal{H}$. 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.

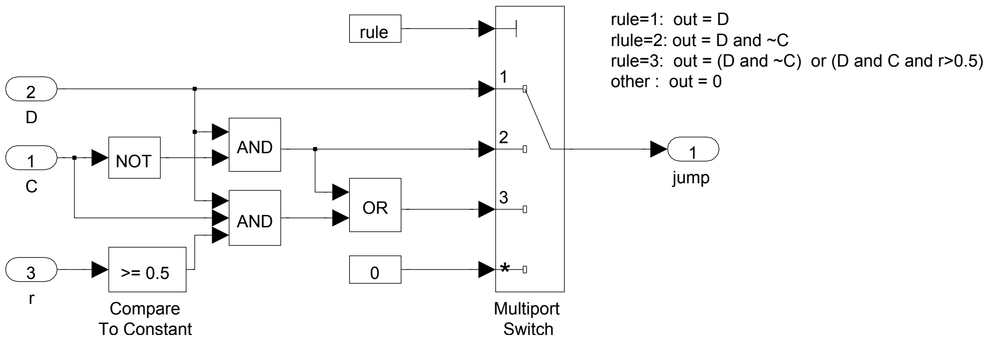


Figure 7: Jump Logic

4.1.3 Update Logic

This block is shown in Figure 8. The update logic uses the *state port* information of the integrator. This port reports the value of the state of the integrator, $[t\ j\ x^\top]^\top$, 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

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

where $[t^-\ j^-\ x^{-\top}]^\top$ is the state that triggers the jump.

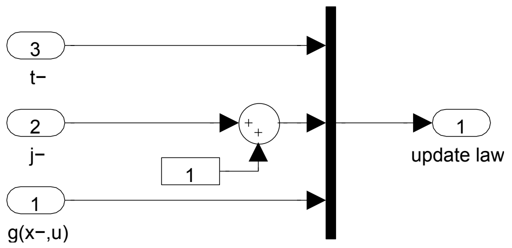


Figure 8: Update Logic

4.1.4 Stop Logic

This block is shown in Figure 9. It stops the simulation under any of the following events:

- The flow time is larger than or equal to the maximum flow time specified by T.
- The jump time is larger than or equal to the maximum number of jumps specified by J.
- 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 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^-.

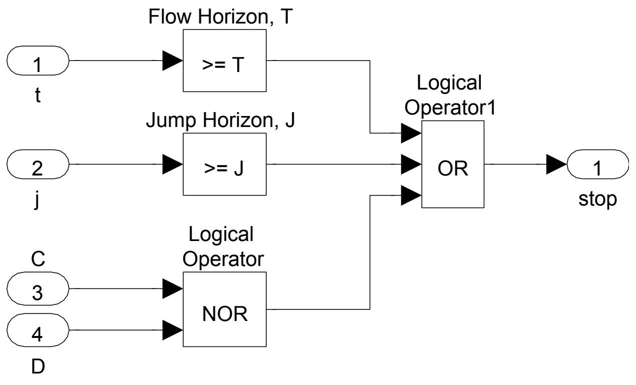


Figure 9: Stop Logic

4.2 Software Requirements

In order to run simulations of single hybrid systems using externally defined functions, MATLAB with Simulink is required.

In order to run simulations using the HyEQ Simulator with embedded MATLAB functions, MATLAB/Simulink and a supported ANSI, C, or C++ 32-bit compiler must be installed. We now briefly describe how to install necessary compilers for Windows and Mac/Linux. For more information on supported compilers, please visit `http://www.mathworks.com/support/compilers/R2012b/win64.html`.

4.2.1 Configuration of HyEQ Simulator with embedded functions for Windows

For 32-bit Windows, the LCC compiler is included with MATLAB. First, open MATLAB and then locate and choose a compiler for building MEX-files by typing

```
 >> mex -setup
```

into the MATLAB command window. Then, follow the prompts as shown below.

```
>> mex -setup

Welcome to mex -setup.  This utility will help you set up
a default compiler.  For a list of supported compilers, see
http://www.mathworks.com/support/compilers/R2012a/win32.html

Please choose your compiler for building MEX-files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1

[0] None

Compiler: 1
```

```
Please verify your choices:

Compiler: Lcc-win32 C 2.4.1

Are these correct [y]/n? y

Done . . .
```

For 64-bit Windows, a C-compiler is not supplied with MATLAB. Before running the HyEQ Toolbox in MATLAB/Simulink, please follow the following steps:

1. If you don't have *Microsoft .NET Framework 4* on your computer, download and install it from `http://www.microsoft.com/en-us/download/details.aspx?id=17851`.
2. Then download and install *Microsoft Windows SDK* from `http://www.microsoft.com/en-us/download/details.aspx?id=8279`.
3. Then perform the steps outlined above for 32-bit Windows to setup and install the compiler.

As of October 10, 2013, when installing the toolbox in Windows 8, please follow the next steps.

1. If you don't have *Microsoft .NET Framework 4* on your computer, download and install it from `http://www.microsoft.com/en-us/download/details.aspx?id=8279`.
2. Then download and install *Microsoft Windows SDK*
   - If you don't have *Visual C++ 2010 SP1* installed on your computer:
     - Download and install *Microsoft Windows SDK 7.1* from `http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=4422`
     - Apply the following patch from *Microsoft* onto the *SDK 7.1* installation: `http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=4422`
   - If you have Visual *Visual C++ 2010 SP1* or its redistributable packages installed on your computer:
     - Uninstall the *Visual C++ 2010* redistributable packages, both x64 and x86 versions. This can be done from *Control Panel / Uninstall Programs Menu*.
     - Download and install *Microsoft Windows SDK 7.1* from `http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=4422`
     - Apply the following patch from *Microsoft* onto the *SDK 7.1* installation: `http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=4422`
     - Reinstall the *Visual C++ 2010* redistributable packages:
       x86 version: `http://www.microsoft.com/en-us/download/details.aspx?id=5555`
       x64 version: `http://www.microsoft.com/en-us/download/details.aspx?id=14632`
3. Then perform the steps outlined above for 32-bit Windows to setup and install the compiler.

4.2.2 Configuration of HyEQ Simulator with embedded functions for Mac/Linux

From a terminal window, check that the file `gcc` is in the folder `/usr/bin`. If it is not there, make a symbolic link. You might require to install the latest version of `Xcode` first. In order to generate a symbolic link for gcc, that MATLAB can find to compile the simulation files (see `http://www.mathworks.com/support/sysreq/previous_releases.html`), change folder to `/usr/bin` and then

```
sudo ln -s gcc gcc-4.2
```

Then, it should be possible to setup the gcc compiler in matlab as follows:

```
>> mex -setup
    Options files control which compiler to use, the compiler and link command
    options, and the runtime libraries to link against.

    Using the 'mexsh -setup' command selects an options file that is
    placed in ~/.matlab/R2013b and used by default for 'mexsh'. An options
    file in the current working directory or specified on the command line
    overrides the default options file in ~/.matlab/R2013b.

    To override the default options file, use the 'mexsh -f' command
    (see 'mexsh -help' for more information).
```

The options files available for MEX are:

```
The options files available for mexsh are:

  1: /Applications/MATLAB_R2013b.app/bin/mexopts.sh :
      Template Options file for building MEX-files


  0: Exit with no changes

Enter the number of the compiler (0-1): 1

Overwrite ~/.matlab/R2013b/mexopts.sh ([y]/n)?: Y

/Applications/MATLAB_R2013b.app/bin/mexopts.sh is being copied to
/SOME_FOLDER/mexopts.sh
```

At this point, it is possible to check if the *gcc* is properly setup by testing any of the Simulink examples with embedded functions (see Figure 3) (e.g., Examples 1.3, 1.4, 1.5, 1.6, 1.7 or 1.8).

If an error regarding "*gmake*" and a warning "no such sysrooot directory: 'Developer/SDKs/MacOSX10.X.sdk'" is shown when compiling, it is necesary to change some lines in the file "*mexopts.sh*" (copied previously in the folder "SOME_FOLDER") .

First, locate the Xcode-SDK in your hard drive. Open a terminal window and execute the following command

```
find `xcode-select -print-path` -name MacOSX10.9.sdk
```

which returns the location of MacOSX10.9.sdk, denoted here as SDK_FOLDER. Now, in the MATLAB command window locate the file "*mexopts.sh*" by typing

```
cd /SOME_FOLDER/
```

Then, open the file

```
edit mexopts.sh
```

and edit the lines

- `SDKROOT='/Developer/SDKs/MacOSX10.X.sdk'`

  to

  `SDKROOT='SDK_FOLDER'`

- `CFLAGS="-fno-common -arch $ARCHS -isysroot $MW_SDKROOT -mmacosx-version-min=$MACOSX_DEPLOYMENT_TARGET"`

  to

  `CFLAGS="-fno-common -arch $ARCHS -isysroot $MW_SDKROOT -mmacosx-version-min=$MACOSX_DEPLOYMENT_TARGET -Dchar16_t=UINT16_T"`
- Replace all the apparitions of $10.X$ to 10.9.

Finally, restart matlab and test any of the aforementioned Simulink examples.

4.3 Configuration of Integration Scheme

Before a simulation is started, it is important to determine the needed integrator scheme, zero-cross detection settings, precision, and other tolerances. Using the default settings does not always give the most efficient or most accurate simulations. One way to edit these settings is to open the Simulink Model, select `Simulation>Configuration Parameters>Solver`, and change the settings there. We have made this simple by defining variables for configuration parameters in the `initialization.m` file. The last few lines of the `initialization.m` file look like that given below.

```
%configuration of solver
RelTol = 1e-8;
MaxStep = .001;
```

In these lines, "RelTol = 1e-8" and "MaxStep = .001" define the relative tolerance and maximum step size of the ODE solver, respectively. These parameters greatly affect the speed and accuracy of solutions.

4.4 Initialization

When the block labeled *Double Click to Initialize* at the top of the Simulink Model is double-clicked, the simulation variables are initialized by calling the script `initialization.m`. The script `initialization.m` defines the initial conditions by defining the initial values of the state components, any necessary parameters, the maximum flow time specified by T, the maximum number of jumps specified by J, and tolerances used when simulating. These can be changed by editing the script file `initialization.m`. See below for sample code to initialize the bouncing ball example, Example 1.3.

```
% initialization for bouncing ball example
clear all
% initial conditions
x0 = [1;0];
% simulation horizon
T = 10;
J = 20;
% rule for jumps
% rule = 1 -> priority for jumps
% rule = 2 -> priority for flows
% rule = 3 -> no priority, random selection when simultaneous conditions
rule = 1;
%configuration of solver
RelTol = 1e-8;
```

It is important to note that variables called in the *Embedded MATLAB function blocks* must be added as inputs and labeled as "parameters". This can be done by opening the *Embedded MATLAB function block* selecting `Tools>Edit Data/Ports` and setting the scope to `Parameter`.

After the block labeled *Double Click to Initialize* is double-clicked and the variables initialized, the simulation is run by clicking the run button or selecting `Simulation>Start`.

4.5 Postprocessing and Plotting solutions

A similar procedure is used to define the plots of solutions after the simulation is run. The solutions can be plotted by double-clicking on the block at the top of the Simulink Model labeled *Double Click to Plot Solutions* which calls the script `postprocessing.m`. The script `postprocessing.m` may be edited to include the desired postprocessing and solution plots. See below for sample code to plot solutions to the bouncing ball example, Example 1.3.

```
%postprocessing for the bouncing ball example
% plot solution
figure(1)
clf
subplot(2,1,1),plotflows(t,j,x)
grid on
ylabel('x')
subplot(2,1,2),plotjumps(t,j,x)
grid on
ylabel('x')
% plot hybrid arc
plotHybridArc(t,j,x)
xlabel('j')
ylabel('t')
zlabel('x')
```

The following functions are used to generate the 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. Figure 10(a) shows a plot created with this function.

- 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). Figure 10(a) shows a plot created with this function.

- plotHybridArc(t,j,x): plots (in blue and red) 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). Figure 11 shows a plot created with this function.

- plotHarc is a function for plotting hybrid arcs (n states).
  - plotHarc(t,j,x): plots the trajectory x versus the hybrid time domain (t, j). If x is a matrix, then the time vector is plotted versus the rows or columns of the matrix, whichever line up.
  - plotHarc(t,j,x,$jstar$): plots the trajectory x versus the hybrid time domain (t, j), and the plot is cut regarding the $jstar$ interval ($jstar = [j_{initial}, j_{final}]$).
  - plotHarc(t,j,x,$jstar$,modificator): Modificator is a cell array that contains the standard matlab ploting modificators (type >> help plotHarc or >> helpwin plotHarc in the command window for more information).

- plotHarcColor plots the trajectory x (vector) on hybrid time domain with color.
  - plotHarcColor(t,j,x,L): plots the trajectory x (vector) versus the hybrid time domain (t, j). The hybrid arc is plotted with L data as color. The input vectors t, j, x, L must have the same length.

  - plotHarcColor(t,j,x,L,$jstar$): If a specific interval in j is required, $jstar = [j_{initial}, j_{final}]$ must be provided. (type >> help plotHarcColor or >> helpwin plotHarcColor in the command window for more information)

- plotHarcColor3D plots an $3D$ hybrid arc with color.
  - plotHarcColor3D(t,j,x,L) plots the trajectory x (3 states) taking into account the hybrid time domain (t, j). The hybrid arc is plotted with L data as color. The input vectors t, j, x, L must have the same length and x must have three columns.
  - plotHarcColor3D(t,j,x,L,$jstar$) If a specific interval in j is required, $jstar = [j_{initial}, j_{final}]$ must be provided.
  - plotHarcColor3D(t,j,x,L,$jstar$,modificator) Modificator is a cell array that contains the standard matlab ploting modificators (type >> help plotHarcColor3D or >> helpwin plotHarcColor3D in the command window for more information).

5 Examples

The examples below illustrate the use of the Simulink implementation above.

Example 1.3 (bouncing ball with input) For the simulation of the bouncing ball system with a constant input and regular data given by

$$f(x,u) := \begin{bmatrix} x_2 \\ -\gamma \end{bmatrix}, C := \{(x,u) \in \mathbb{R}^2 \times \mathbb{R} \mid x_1 \geq u\} \tag{4}$$

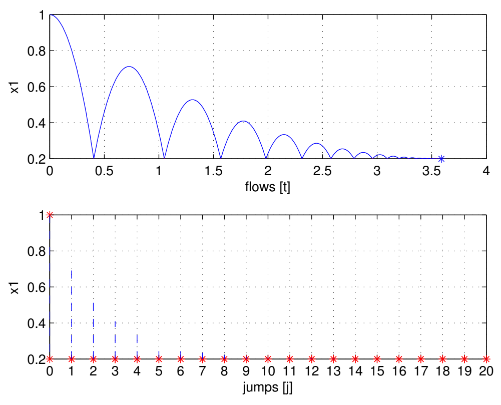
$$g(x,u) := \begin{bmatrix} u \\ -\lambda x_2 \end{bmatrix}, D := \{(x,u) \in \mathbb{R}^2 \times \mathbb{R} \mid x_1 \leq u \ , \ x_2 \leq 0\} \tag{5}$$

where $\gamma > 0$ is the gravity constant, u is the input constant, and $\lambda \in [0, 1)$ is the restitution coefficient. The MATLAB scripts in each of the function blocks of the implementation above are given as follows. An input was chosen to be $u(t, j) = 0.2$ for all (t, j). The constants for the bouncing ball system are $\gamma = 9.81$ and $\lambda = 0.8$.

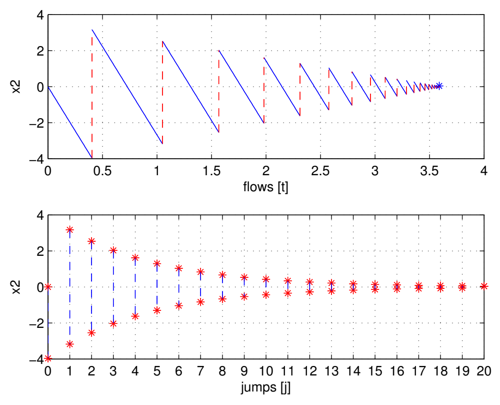
The following procedure is used to simulate this example with `HyEQsimulator.mdl`:

- `HyEQsimulator.mdl` is opened in MATLAB/Simulink.
- The *Embedded MATLAB function blocks f, C, g, D* are edited by double-clicking on the block and editing the script. In each embedded function block, parameters must be added as inputs and defined as parameters by selecting `Tools>Edit Data/Ports`, and setting the scope to `Parameter`. For this example, *gamma* and *lambda* are defined in this way.
- The initialization script `initialization.m` is edited by opening the file and editing the script. The flow time and jump horizons, T and J are defined as well as the initial conditions for the state vector, x_0, and input vector, u_0, and a rule for jumps, *rule*.
- The postprocessing script `postprocessing.m` is edited by opening the file and editing the script. Flows and jumps may be plotted by calling the functions *plotflows* and *plotjumps*, respectively. The hybrid arc may be plotted by calling the function *plotHybridArc*.
- The simulation stop time and other simulation parameters are set to the values defined in `initialization.m` by selecting `Simulation>Configuration Parameters>Solver` and inputting T, *RelTol*, $MaxStep$, etc..
- The masked integrator system is double-clicked and the simulation horizons and initial conditions are set as desired.

- The block labeled *Double Click to Initialize* is double-clicked to initialize variables.
- The simulation is run by clicking the run button or selecting `Simulation>Start`.
- The block labeled *Double Click to Plot Solutions* is double-clicked to plot the desired solutions.



(a) Height



(b) Velocity

Figure 10: Solution of Example 1.3

```
function xdot = f(x, u, gamma)
% state
x1 = x(1);
x2 = x(2);
% flow map: xdot=f(x,u);
xdot = [x(2); gamma];
```

```
function v = C(x, u)
% flow set
if (x(1) >= u(1))  % flow condition
    v = 1;  % report flow
else
    v = 0;   % do not report flow
end
```

```
function xplus = g(x, u, lambda)
% jump map
xplus = [u(1); -lambda*x(2)];
```

```
function v = D(x, u)
% jump set
if (x(1) <= u(1)) && (x(2) <= 0)  % jump condition
    v = 1;  % report jump
else
    v = 0;   % do not report jump
end
```

A solution to the bouncing ball system from $x(0,0) = [1,0]^\top$ and with $T = 10, J = 20$, $rule = 1$, is depicted in Figure 10(a) (height) and Figure 10(b) (velocity). Both the projection onto t and j are shown.

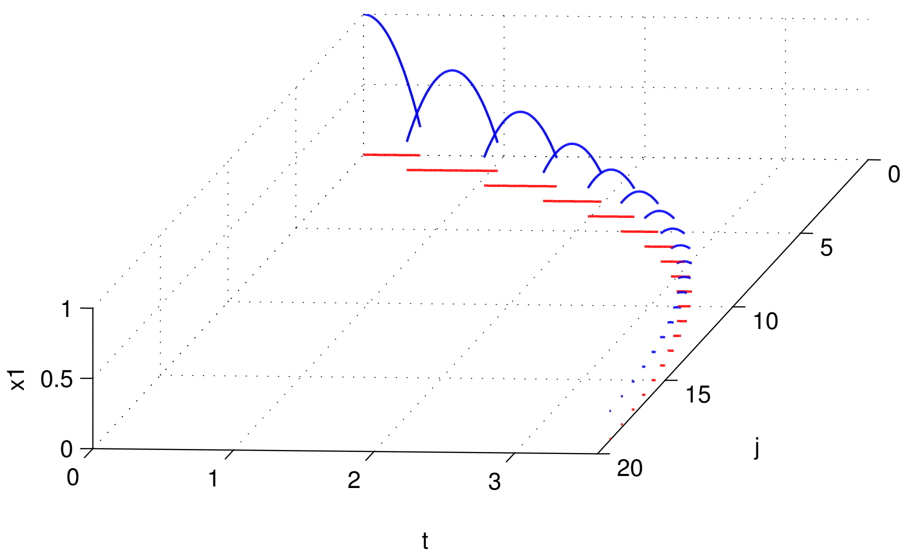


Figure 11: Hybrid arc corresponding to a solution of Example 1.3: height

Figure 11 depicts the corresponding hybrid arc for the position state.

These simulations reflect the expected behavior of the bouncing ball model. Note the only difference between this example and the example of a bouncing ball without a constant input is that, in this example, the ball bounces on a platform at a height of the chosen input value 0.2 rather than the ground at a value of 0.

For MATLAB/Simulink files of this example, see Examples/Example_1.3.

□

Example 1.4 (alternate way to simulate the bouncing ball)

Consider the bouncing ball system with a constant input and regular data as given in Example 1.3. This example shows that a MATLAB function block, such as the jump set D, can be replaced with operational blocks in Simulink. Figure 12 shows this implementation. The other functions and solutions are the same as in Example 1.3.

For MATLAB/Simulink files corresponding to this alternative implementation, see Examples/Example_1.4.

□

Example 1.5 (vehicle following a track with boundaries) Consider a vehicle modeled by a Dubins vehicle model traveling along a given track with state vector $x = [\xi_1, \xi_2, \xi_3]^\top$ with dynamics given by $\dot{\xi_1} = u \cos \xi_3$, $\dot{\xi_2} = u \sin \xi_3$, and $\dot{\xi_3} = -\xi_3 + r(q)$. The input u is the tangential velocity of the vehicle, ξ_1 and ξ_2 describe the vehicle's position on the plane, and ξ_3 is the vehicle's orientation angle. Also consider a switching controller attempting to keep the vehicle inside the boundaries of a track given by $\{(\xi_1, \xi_2) : -1 \le \xi_1 \le 1\}$. A state $q \in \{1, 2\}$ is used to define the modes of operation of the controller. When $q = 1$, the vehicle is traveling to the left, and when $q = 2$, the vehicle is traveling to the right. A logic variable r is defined in order to steer the vehicle back inside the boundary. The state of the closed-loop system is given by $x := [\xi^\top \; q]^\top$. A model

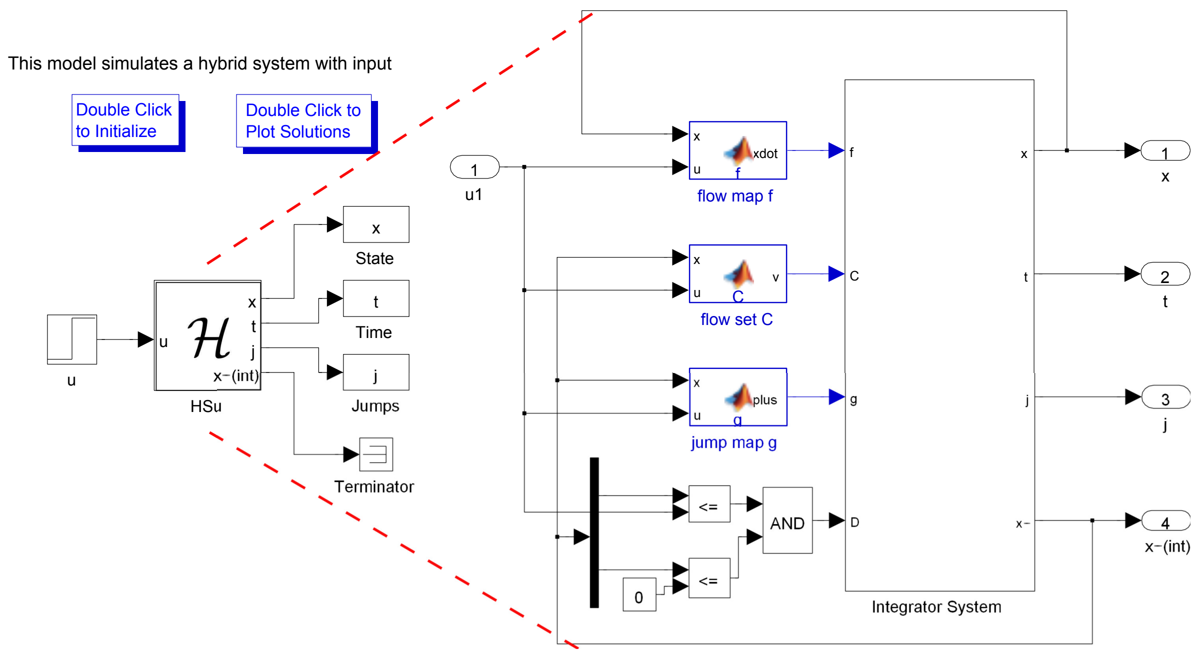


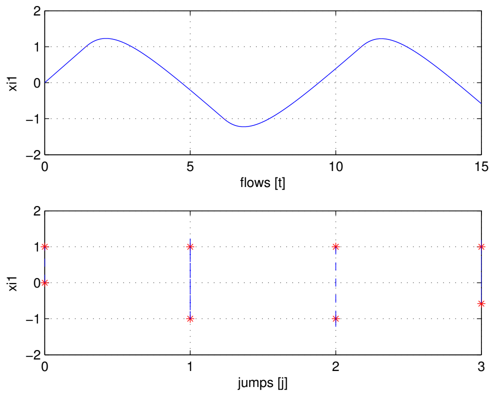
Figure 12: Simulink implementation of bouncing ball example with operator blocks

of such a closed-loop system is given by

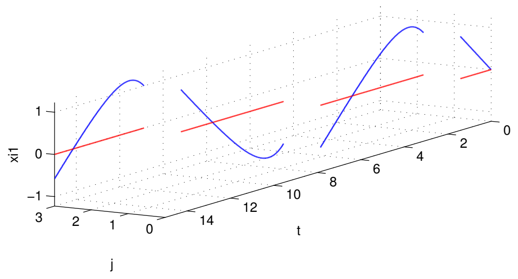
$$
\begin{aligned}
f(x,u) &:= \begin{bmatrix} \begin{bmatrix} u\cos(\xi_3) \\ u\sin(\xi_3) \\ -\xi_3 + r(q) \end{bmatrix} \\ u \end{bmatrix}, \quad r(q) := \begin{cases} \frac{3\pi}{4} & \text{if } \ q = 1 \\ \frac{\pi}{4} & \text{if } \ q = 2 \end{cases} && (6) \\
C &:= \left\{ (\xi,u) \in \mathbb{R}^3 \times \{1,2\} \times \mathbb{R} \mid (\xi_1 \leq 1, q = 2) \text{ or } (\xi_1 \geq -1, q = 1) \right\}, && (7) \\
g(\xi,u) &:= \begin{cases} \begin{bmatrix} \xi \\ 2 \end{bmatrix} & \text{if } \ \xi_1 \leq -1, \quad q = 1 \\ \begin{bmatrix} \xi \\ 1 \end{bmatrix} & \text{if } \ \xi_1 \geq 1, \quad q = 2 \end{cases}, && (8) \\
D &:= \left\{ (\xi,u) \in \mathbb{R}^3 \times \{1,2\} \times \mathbb{R} \mid (\xi_1 \geq 1, q = 2) \text{ or } (\xi_1 \leq -1, q = 1) \right\} && (9)
\end{aligned}
$$

The MATLAB scripts in each of the function blocks of the implementation above are given as follows. The tangential velocity of the vehicle is chosen to be $u = 1$, the initial position on the plane is chosen to be $(\xi_1, \xi_2) = (0, 0)$, and the initial orientation angle is chosen to be $\xi_3 = \frac{\pi}{4}$ radians.

```
function xdot = f(x, u)
% state
xi = z(statevect);
xi1 = xi(1);        %x-position
xi2 = xi(2);        %y-position
xi3 = xi(3);        %orientation angle
q = xi(4);
% q = 1 --> going left
% q = 2 --> going right
if q == 1
    r = 3*pi/4;
elseif q == 2
    r = pi/4;
```



(a) Trajectory



(b) Hybrid arc

Figure 13: Solution of Example 1.5

```
else
    r = 0;
end
% flow map: xidot=f(xi,u);
xi1dot = u*cos(xi3);  %tangential velocity in x-direction
xi2dot = u*sin(xi3);  %tangential velocity in y-direction
xi3dot = -xi3 + r;      %angular velocity
qdot = 0;
xdot = [xi1dot;xi2dot;xi3dot;qdot];
```

```
function v = C(x, u)
% state
xi = z(statevect);
xi1 = xi(1);      %x-position
xi2 = xi(2);      %y-position
xi3 = xi(3);      %orientation angle
q = xi(4);
% q = 1 --> going left
% q = 2 --> going right
% flow set
if ((xi1 < 1) && (q == 2)) || ((xi1 > -1) && (q == 1))  % flow condition
    v = 1;  % report flow
else
    v = 0;   % do not report flow
end
```

```
function xplus = g(x, u)
% state
xi = z(statevect);
xi1 = xi(1);      %x-position
xi2 = xi(2);      %y-position
xi3 = xi(3);      %orientation angle
q = xi(4);
% q = 1 --> going left
% q = 2 --> going right
xi1plus=xi1;
xi2plus=xi2;
xi3plus=xi3;
qplus=q;
% jump map
if ((xi1 >= 1) && (q == 2)) || ((xi1 <= -1) && (q == 1))
   qplus = 3-q;
else
    qplus = q;
end
xplus = [xi1plus;xi2plus;xi3plus;qplus];
```

```
function v = D(x, u)
% state
xi = z(statevect);
xi1 = xi(1);       %x-position
xi2 = xi(2);       %y-position
xi3 = xi(3);       %orientation angle
q = xi(4);
% q = 1 --> going left
% q = 2 --> going right
% jump set
if ((xi1 >= 1) && (q == 2)) || ((xi1 <= -1) && (q == 1))  % jump condition
     v = 1;  % report jump
 else
     v = 0; % do not report jump
 end
```

A solution to the system of a vehicle following a track in $\{(\xi_1, \xi_2) : -1 \leq \xi_1 \leq 1\}$, and with $T = 15$, $J = 10$, $rule = 1$, is depicted in Figure 13(a) (trajectory). Both the projection onto t and j are shown. Figure 13(b) depicts the corresponding hybrid arc.

For MATLAB/Simulink files of this example, see Examples/Example_1.5.

□

Example 1.6 (interconnection of hybrid systems $\mathcal{H}_1$ (bouncing ball) and $\mathcal{H}_2$ (moving platform)) Consider a bouncing ball ($\mathcal{H}_1$) bouncing on a platform ($\mathcal{H}_2$) at some initial height and converging to the ground at zero height. This is an interconnection problem because the current states of each system affect the behavior of the other system. In this interconnection, the bouncing ball will contact the platform, bounce back up, and cause a jump in height of the platform so that it gets closer to the ground. After some time, both the ball and the platform will converge to the ground. In order to model this system, the output of the bouncing ball becomes the input of the moving platform, and vice versa. For the simulation of the described system with regular data where $\mathcal{H}_1$ is given by

$$f_1(\xi, u_1, v_1) := \begin{bmatrix} \xi_2 \\ -\gamma - b\xi_2 + v_{11} \end{bmatrix}, C_1 := \{(\xi, u_1) \mid \xi_1 \geq u_1, u_1 \geq 0\} \tag{10}$$

$$g_1(\xi, u_1, v_1) := \begin{bmatrix} \xi_1 + \alpha_1 \xi_2^2 \\ e_1|\xi_2| + v_{12} \end{bmatrix}, D_1 := \{(\xi, u_1) \mid \xi_1 = u_1, u_1 \geq 0\}, y_1 = h_1(\xi) := \xi_1 \tag{11}$$

where $\gamma, b, \alpha_1 > 0, e_1 \in [0, 1)$, $\xi = [\xi_1, \xi_2]^\top$ is the state, $y_1 \in \mathbb{R}$ is the output, $u_1 \in \mathbb{R}$ and $v_1 = [v_{11}, v_{12}]^\top \in \mathbb{R}^2$ are the inputs, and the hybrid system $\mathcal{H}_2$ is given by

$$f_2(\eta, u_2, v_2) := \begin{bmatrix} \eta_2 \\ -\eta_1 - 2\eta_2 + v_{12} \end{bmatrix}, C_2 := \{(\eta, u_2) \mid \eta_1 \leq u_2, \eta_1 \geq 0\} \tag{12}$$

$$g_2(\eta, u_2, v_2) := \begin{bmatrix} \eta_1 - \alpha_2|\eta_2| \\ -e_2|\eta_2| + v_{22} \end{bmatrix}, D_2 := \{(\eta, u_2) \mid \eta_1 = u_2, \eta_1 \geq 0\}, y_2 = h_2(\eta) := \eta_1 \tag{13}$$

where $\alpha_2 > 0, e_2 \in [0, 1)$, $\eta = [\eta_1, \eta_2]^\top \in \mathbb{R}^2$ is the state, $y_2 \in \mathbb{R}$ is the output, and $u_2 \in \mathbb{R}$ and $v_2 = [v_{21}, v_{22}]^\top \in \mathbb{R}^2$ are the inputs.

Therefore, the interconnection may be defined by the input assignment

$$u_1 = y_2, \qquad u_2 = y_1. \tag{14}$$

The signals v_1 and v_2 are included as external inputs in the model in order to simulate the effects of environmental perturbations, such as a wind gust, on the system.

The MATLAB scripts in each of the function blocks of the implementation above are given as follows. The constants for the interconnected system are $\gamma = 0.8$, $b = 0.1$, and $\alpha_1, \alpha_2 = 0.1$.

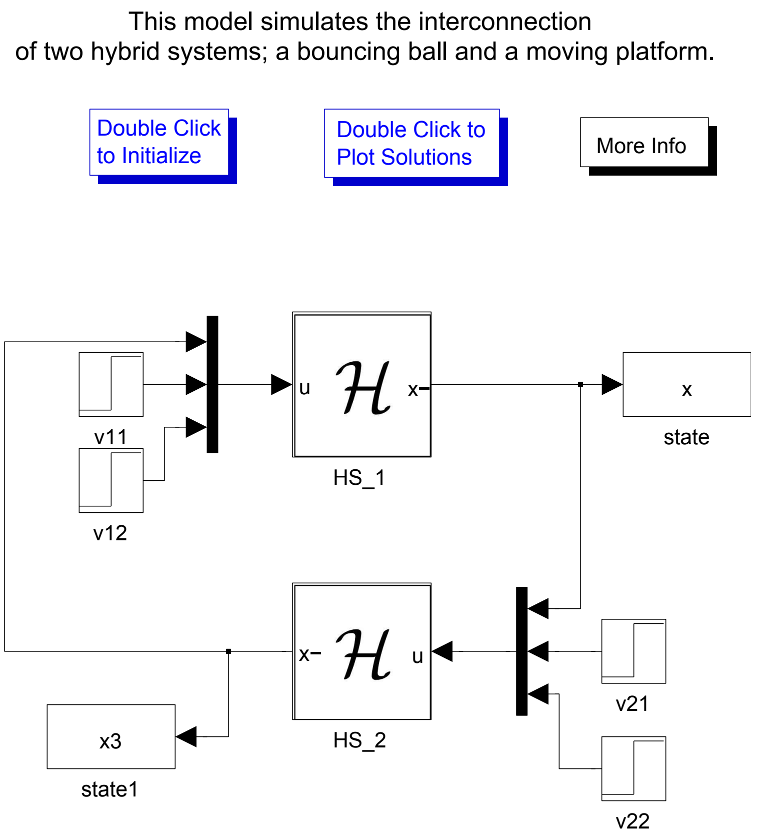

This model simulates the interconnection
of two hybrid systems; a bouncing ball and a moving platform.

Figure 14: MATLAB/Simulink implementation of interconnected hybrid systems $\mathcal{H}_1$ and $\mathcal{H}_2$

For hybrid system $\mathcal{H}_1$:

```
function xdot = f(x, u)
% state
xi1 = x(1);
xi2 = x(2);
%input
y2 = u(1);
```

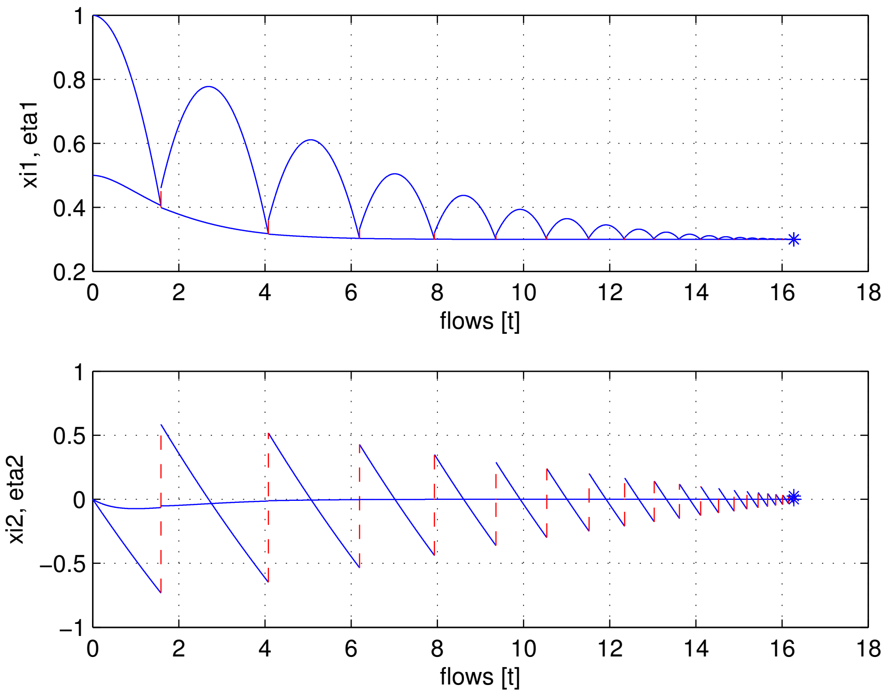
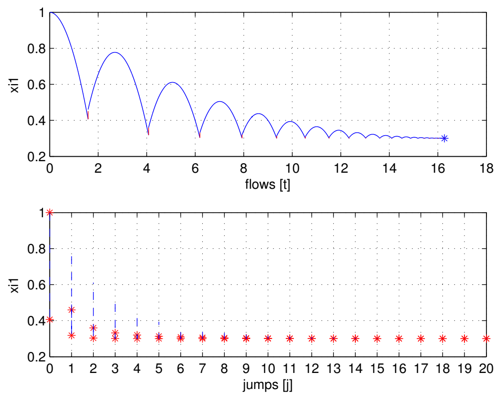


Figure 15: Solution of Example 1.6: height and velocity

```
v11 = u(2);
v12 = u(3);
% flow map
%xdot=f(x,u);
xi1dot = xi2;
xi2dot = -0.8-0.1*xi2+v11;
xdot = [xi1dot;xi2dot];
```

```
function v = C(x, u)
% state
xi1 = x(1);
xi2 = x(2);
%input
y2 = u(1);
v11 = u(2);
v12 = u(3);
if (xi1 >= y2)   % flow condition
    v = 1;  % report flow
else
    v = 0;    % do not report flow
end
```

```
function xplus = g(x, u)
% state
xi1 = x(1);
xi2 = x(2);
```


(a) Height

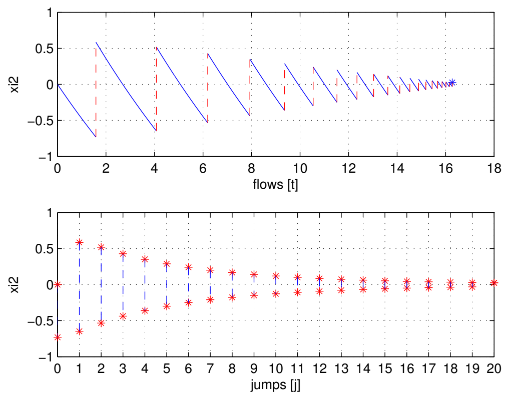

(b) Velocity

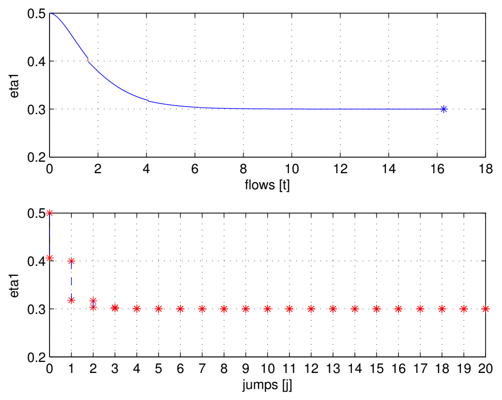
Figure 16: Solution of Example 1.6 for system $\mathcal{H}_1$

```
%input
y2 = u(1);
v11 = u(2);
v12 = u(3);
%jump map
xi1plus=y2+0.1*xi2^2;
xi2plus=0.8*abs(xi2)+v12;
xplus = [xi1plus;xi2plus];
```

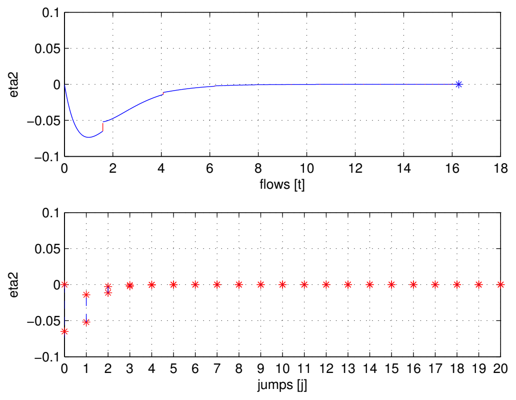
```
function v = D(x, u)
% state
xi1 = x(1);
xi2 = x(2);
%input
y2 = u(1);
v11 = u(2);
v12 = u(3);
% jump set
if (xi1 <= y2) % jump condition
    v = 1;  % report jump
else
    v = 0;   % do not report jump
end
```

For hybrid system $\mathcal{H}_2$:

```
function xdot = f(x, u)
% state
eta1 = x(1);
eta2 = x(2);
%input
y1 = u(1);
v21 = u(2);
v22 = u(3);
% flow map
```



(a) Height



(b) Velocity

Figure 17: Solution of Example 1.6 for system $\mathcal{H}_2$

```
eta1dot = eta2;
eta2dot = -eta1-2*eta2+v21;
xdot = [eta1dot;eta2dot];
```

```
function v = C(x, u)
% state
eta1 = x(1);
eta2 = x(2);
%input
y1 = u(1);
v21 = u(2);
v22 = u(3);
% flow set
if (eta1 <= y1)  % flow condition
    v = 1;  % report flow
else
    v = 0;   % do not report flow
end
```

```
function xplus = g(x, u)
% state
eta1 = x(1);
eta2 = x(2);
%input
y1 = u(1);
v21 = u(2);
v22 = u(3);
% jump map
eta1plus = y1-0.1*abs(eta2);
eta2plus = -0.8*abs(eta2)+v22;
xplus = [eta1plus;eta2plus];
```

```
function v = D(x, u)
% state
```

```
eta1 = x(1);
eta2 = x(2);
%input
y1 = u(1);
v21 = u(2);
v22 = u(3);
% jump set
if (eta1 >= y1) % jump condition
    v = 1;  % report jump
else
    v = 0;   % do not report jump
end
```

A solution to the interconnection of hybrid systems $\mathcal{H}_1$ and $\mathcal{H}_2$ with $T = 18$, $J = 20$, $rule = 1$, is depicted in Figure 15. Both the projection onto t and j are shown. A solution to the hybrid system $\mathcal{H}_1$ is depicted in Figure 16(a) (height) and Figure 16(b) (velocity). A solution to the hybrid system $\mathcal{H}_2$ is depicted in Figure 17(a) (height) and Figure 17(b) (velocity).

These simulations reflect the expected behavior of the interconnected hybrid systems.

For MATLAB/Simulink files of this example, see Examples/Example_1.6.

□

Example 1.7 (biological example: synchronization of two fireflies) Consider a biological example of the synchronization of two fireflies flashing. The fireflies can be modeled mathematically as periodic oscillators which tend to synchronize their flashing until they are flashing in phase with each other. A state value of $\tau_i = 1$ corresponds to a flash, and after each flash, the firefly automatically resets its internal timer (periodic cycle) to $\tau_i = 0$. The synchronization of the fireflies can be modeled as an interconnection of two hybrid systems because every time one firefly flashes, the other firefly notices and jumps ahead in its internal timer τ by $(1 + \varepsilon)\tau$, where ε is a biologically determined coefficient. This happens until eventually both fireflies synchronize their internal timers and are flashing simultaneously. Each firefly can be modeled as a hybrid

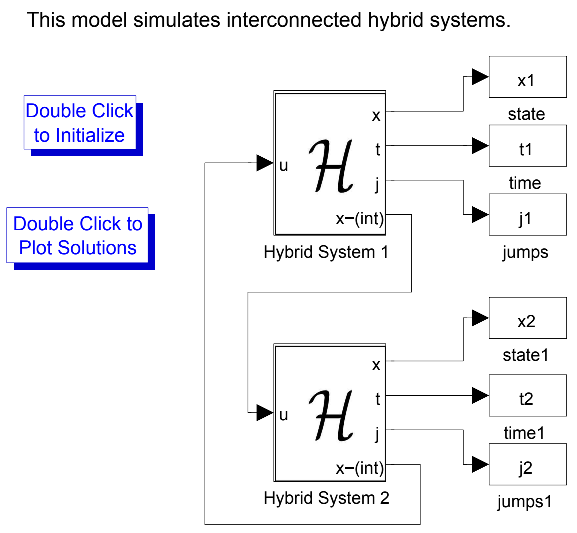
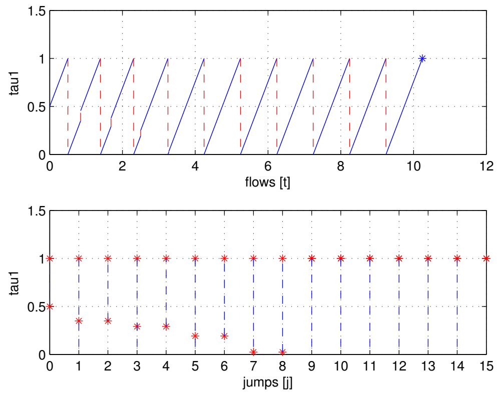

This model simulates interconnected hybrid systems.
Double Click to Initialize
Double Click to Plot Solutions
u
x
t
j
x−(int)
Hybrid System 1
Hybrid System 2
x1
state
t1
time
j1
jumps
x2
state1
t2
time1
j2
jumps1

Figure 18: Interconnection Diagram for Example 1.7

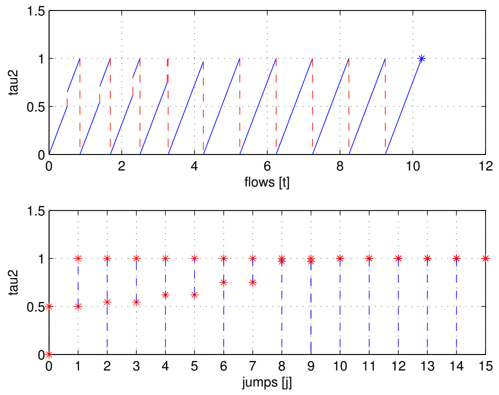
system given by

$$
\begin{aligned}
f_i(\tau_i, u_i) \;&:=\; 1, && (15)\\
C_i \;&:=\; \big\{(\tau_i, u_i) \in \mathbb{R}^2 \;\mid 0 \le \tau_i \le 1\big\} \cap \big\{(\tau_i, u_i) \in \mathbb{R}^2 \;\mid 0 \le u_i \le 1\big\} && (16)\\
g_i(\tau_i, u_i) \;&:=\; \begin{cases} (1+\varepsilon)\tau_i & (1+\varepsilon)\tau_i < 1 \\ 0 & (1+\varepsilon)\tau_i \ge 1 \end{cases} && (17)\\
D_i \;&:=\; \big\{(\tau_i, u_i) \in \mathbb{R}^2 \;\mid \tau_i = 1\big\} \cup \big\{(\tau_i, u_i) \in \mathbb{R}^2 \;\mid u_i = 1\big\}. && (18)
\end{aligned}
$$

The interconnection diagram for this example is simpler than in the previous example because now no external inputs are being considered. The only event that affects the flashing of a firefly is the flashing of the other firefly. The interconnection diagram can be seen in Figure 18.



(a) Solution for system $\mathcal{H}_1$



(b) Solution for system $\mathcal{H}_2$

Figure 19: Solution of Example 1.7

For hybrid system $\mathcal{H}_i$, $i = 1, 2$:

```
function taudot = f(tau, u)
% flow map
taudot = 1;
```

```
function v  = C(tau, u)
% flow set
if ((tau > 0) && (tau < 1)) || ((u > 0) && (u <= 1))  % flow condition
    v = 1;  % report flow
else
    v = 0;  % do not report flow
end
```

```
function tauplus = g(tau, u)
% jump map
if (1+e)*tau < 1
    tauplus = (1+e)*tau;
elseif (1+e)*tau >= 1
    tauplus = 0;
else
    tauplus = tau;
end
```

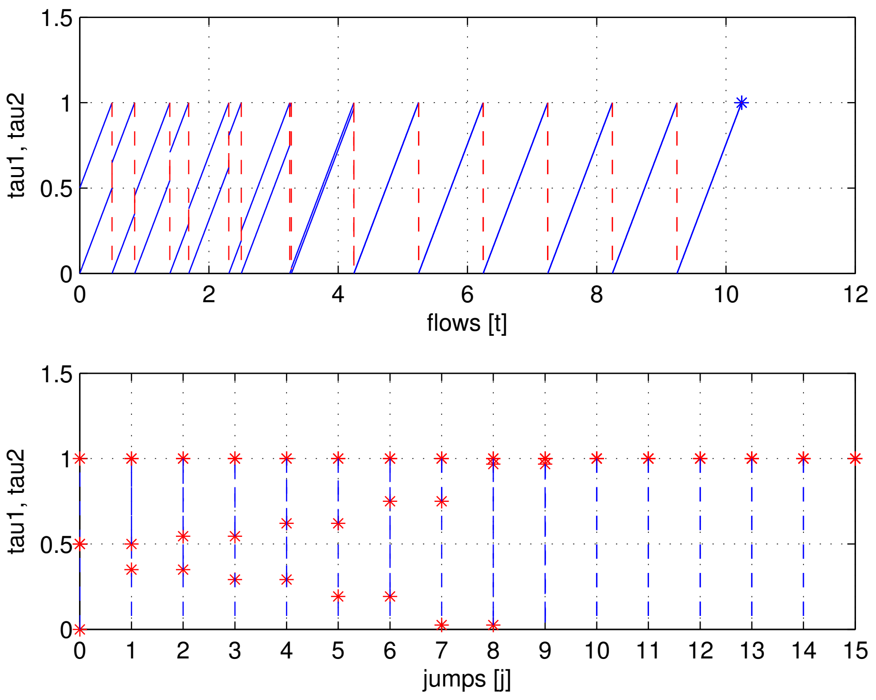


Figure 20: Solution of Example 1.7 for interconnection of $\mathcal{H}_1$ and $\mathcal{H}_2$

```
function v  = D(tau, u)
% jump set
if (u >= 1) || (tau >= 1) % jump condition
    v = 1;  % report jump
else
    v = 0;  % do not report jump
end
```

A solution to the interconnection of hybrid systems $\mathcal{H}_1$ and $\mathcal{H}_2$ with $T = 15, J = 15$, $rule = 1$, $\varepsilon = 0.3$ is depicted in Figure 20. Both the projection onto t and j are shown. A solution to the hybrid system $\mathcal{H}_1$ is depicted in Figure 19(a). A solution to the hybrid system $\mathcal{H}_2$ is depicted in Figure 19(b).

These simulations reflect the expected behavior of the interconnected hybrid systems. The fireflies initially flash out of phase with one another and then synchronize to flash in the same phase.

For MATLAB/Simulink files of this example, see Examples/Example_1.7.

□

Example 1.8 (a simple mathematical example to show different type of simulation results) Consider the hybrid system with data

$$f(x) := -x,\ C := [0,1],\ g(x) := 1 + \mathrm{mod}(x,2),\ D := \{1\} \cup \{2\}\ .$$

Note that solutions from $\xi = 1$ and $\xi = 2$ are nonunique. The following simulations show the use of the variable *rule* in the *Jump Logic block*.

Jumps enforced:

A solution from $x0 = 1$ with $T = 10, J = 20$, $rule = 1$ is depicted in Figure 21(a). The solution jumps from 1 to 2, and from 2 to 1 repetitively.

Flows enforced:

A solution from $x0 = 1$ with $T = 10, J = 20$, $rule = 2$ is depicted in Figure 21(b). The solution flows for all time and converges exponentially to zero.

Random rule:

A solution from $x0 = 1$ with $T = 10, J = 20$, $rule = 3$ is depicted in Figure 21(c). The solution jumps to 2, then jumps to 1 and flows for the rest of the time converging to zero exponentially.

Enlarging D to

$$D := [1/50, 1] \cup \{2\}$$

causes the overlap between C and D to be "thicker". The simulation result is depicted in Figure 21(d) with the same parameters used in the simulation in Figure 21(c). The plot suggests that the solution jumps several times until $x < 1/50$ from where it flows to zero. However, Figure 21(e), a zoomed version of Figure 21(d), shows that initially the solution flows and that at $(t, j) = (0.2e - 3, 0)$ it jumps. After the jump, it continues flowing, then it jumps a few times, then it flows, etc. The combination of flowing and jumping occurs while the solution is in the intersection of C and D, where the selection of whether flowing or jumping is done randomly due to using $rule = 3$.

This simulation also reveals that this implementation does not precisely generate hybrid arcs. The maximum step size was set to $0.1e - 3$. The solution flows during the first two steps of the integration of the flows with maximum step size. The value at $t = 0.1e - 3$ is very close to 1. At $t = 0.2e - 3$, instead of assuming a value given by the flow map, the value of the solution is about 0.5, which is the result of the jump occurring at $(0.2e - 3, 0)$. This is the value stored in x at such time by the integrator. Note that the value of x' at $(0.2e - 3, 0)$ is the one given by the flow map that triggers the jump, and if available for recording, it should be stored in $(0.2e - 3, 0)$. This is a limitation of the current implementation.

The following simulations show the *Stop Logic block* stopping the simulation at different events.

Solution outside $C \cup D$:

Taking $D = \{1\}$, a simulation starting from $x0 = 1$ with $T = 10, J = 20$, $rule = 1$ stops since the solution leaves $C \cup D$. Figure 22(a) shows this.

Solution reaches the boundary of C from where jumps are not possible:

Replacing the flow set by $[1/2, 1]$ a solution starting from $x0 = 1$ with $T = 10, J = 20$ and $rule = 2$ flows for all time until it reaches the boundary of C where jumps are not possible. Figure 22(b) shows this.

Note that in this implementation, the Stop Logic is such that when the state of the hybrid system is not in $(C \cup D)$, then the simulation is stopped. In particular, if this condition becomes true while flowing, then the last value of the computed solution will not belong to C. It could be desired to be able to recompute the solution so that its last point belongs to the corresponding set. From that point, it should be the case that solutions cannot be continued.

For MATLAB/Simulink files of this example, see Examples/Example_1.8. □

6 Further Reading

Installation files for the HyEQ Toolbox described in this paper can be found at MATLAB Central and at the author's website

`https://hybrid.soe.ucsc.edu/software`.

Also, resources and examples are shared by the HyEQ Toolbox users in the blog

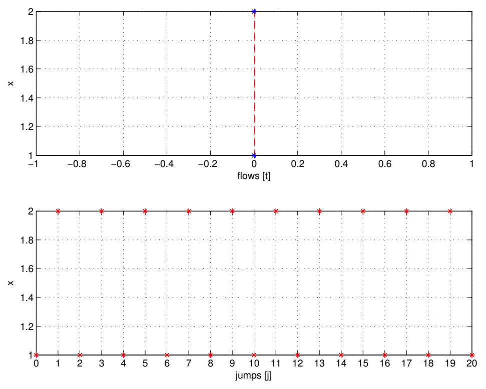
`http://hybridsimulator.wordpress.com`.

7 Acknowledgments

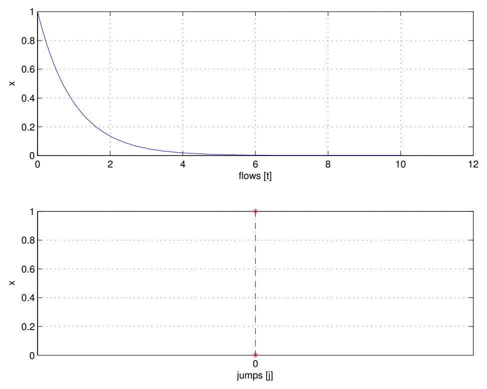
We would like to thank Giampiero Campa for his thoughtful feedback and advice as well as Torstein Ingebrigtsen Bo for his comments and initial version of the lite simulator code.

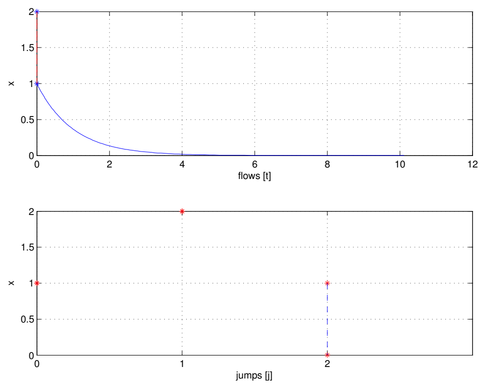
8 References

[1] R. G. Sanfelice, D. A. Copp, and P. Nanez "A Toolbox for Simulation of Hybrid Systems in Matlab/Simulink: Hybrid Equations (HyEQ) Toolbox", Proceedings of Hybrid Systems: Computation and Control Conference, pp. 101–106, 2013.
[2] `http://control.ee.ethz.ch/~ifaatic/ex/example1.m`. Institut für Automatik - Automatic Control Laboratory, ETH Zurich, 2011.
[3] R. Goebel, R. G. Sanfelice, and A. R. Teel, Hybrid dynamical systems. IEEE Control Systems Magazine, 28-93, 2009.
[4] R. G. Sanfelice and A. R. Teel, Dynamical Properties of Hybrid Systems Simulators. Automatica, 46, No. 2, 239–248, 2010.
[5] Sanfelice, R. G., Interconnections of Hybrid Systems: Some Challenges and Recent Results Journal of Nonlinear Systems and Applications, 111–121, 2011.



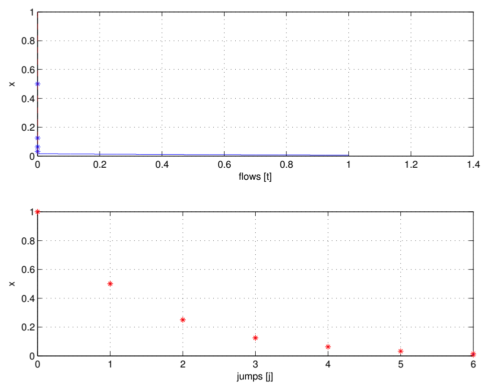
(a) Forced jumps logic.

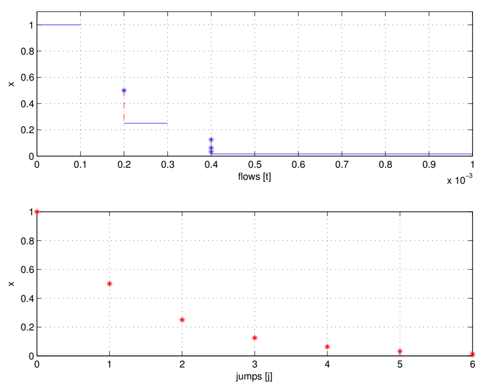




(b) Forced flows logic.

(c) Random logic for flowing/jumping.

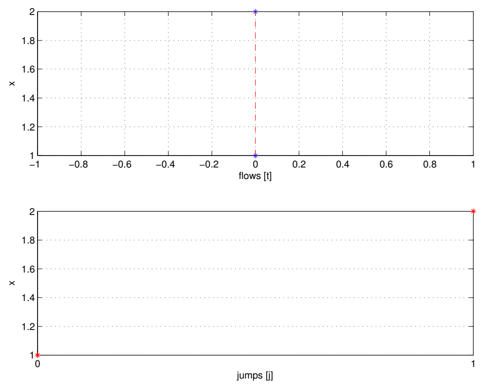




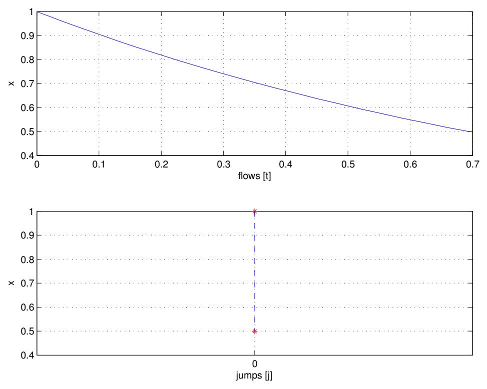
(d) Random logic for flowing/jumping.

(e) Random logic for flowing/jumping. Zoomed version.

Figure 21: Solution of Example 1.8



(a) Forced jump logic and different D.



(b) Forced flow logic.

Figure 22: Solution of Example 1.8 with premature stopping.