Simulating Hybrid Systems in Matlab/Simulink, v0.4

David A. Copp and Ricardo G. Sanfelice
Hybrid Dynamics and Control Laboratory
University of Arizona

1 A Simulink Model

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

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

The reader is referred to [2,3] for an introduction to this class of hybrid systems. Figure 1 shows a Simulink implementation proposed here.

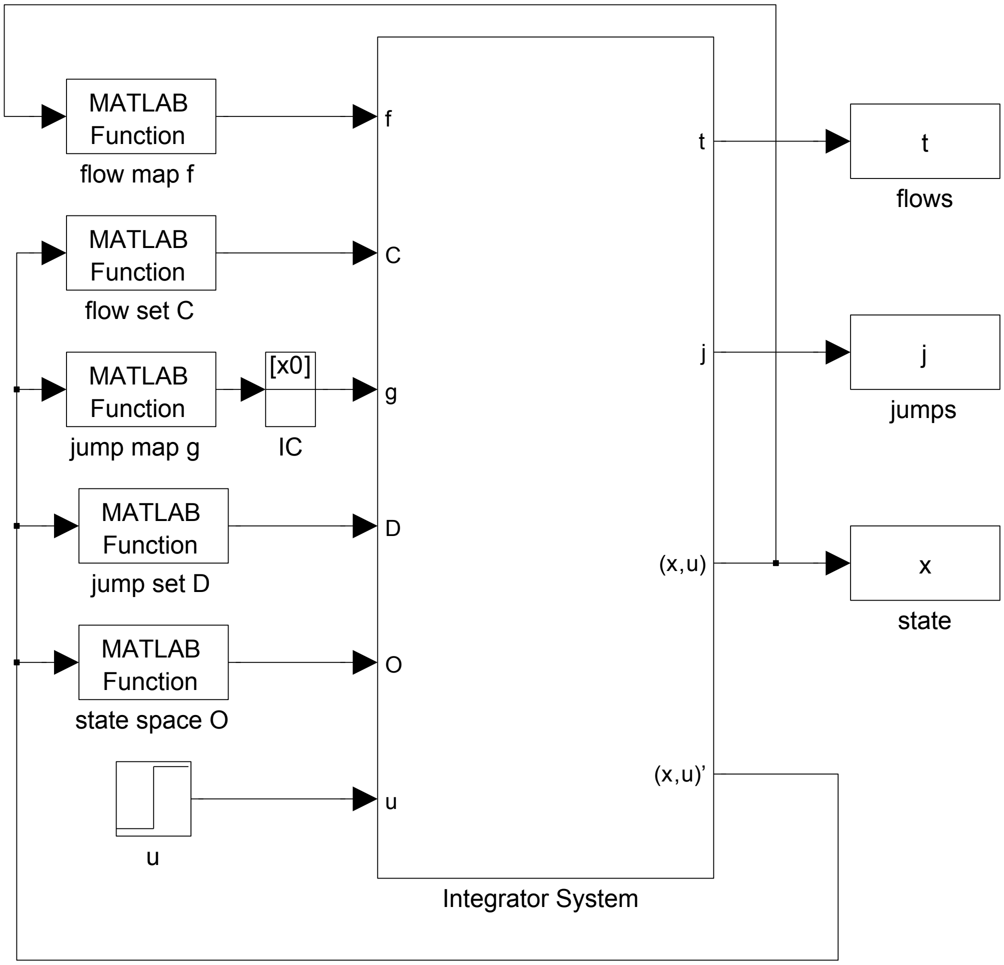

Figure 1: Matlab/Simulink implementation of a hybrid system $\mathcal{H} = (O, f, C, g, D)$ with inputs.

Five basic blocks are used to define the dynamics of the hybrid system $\mathcal{H}$:

- The flow map is implemented in a *Matlab function block* executing the function `f.m`. Its input is a vector with components defining the state of the system x, and the inputs 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 a *Matlab function block* executing the function `C.m`. Its input is a vector with components of the state of the *Integrator system* x' and the inputs u', and its output is equal to 1 if the state belongs to the set C or equal to 0 otherwise. The prime notation denotes the previous value of the variables – the value x' is obtained from the state port of the integrator.
- The jump map is implemented in a *Matlab function block* executing the function `g.m`. Its input is a vector with components of the state of the *Integrator system* x' and the input u', and its output is the value of the jump map g.
- The jump set is implemented in a *Matlab function block* executing the function `D.m`. Its input is a vector with components of the state of the *Integrator* system x' and the input u', and its output is equal to 1 if the state belongs to D or equal to 0 otherwise.
- The state space is implemented in a *Matlab function block* executing the function `O.m`. Its input is a vector with components of the state of the *Integrator system* x' and the input u', and its output is equal to 1 if the state belongs to O or equal to 0 otherwise.

A script `run.m` is used to define the simulation variables and run the simulations. It defines the initial conditions by defining the initial values of the state components, the maximum flow time specified by T, and the maximum number of jumps specified by J.

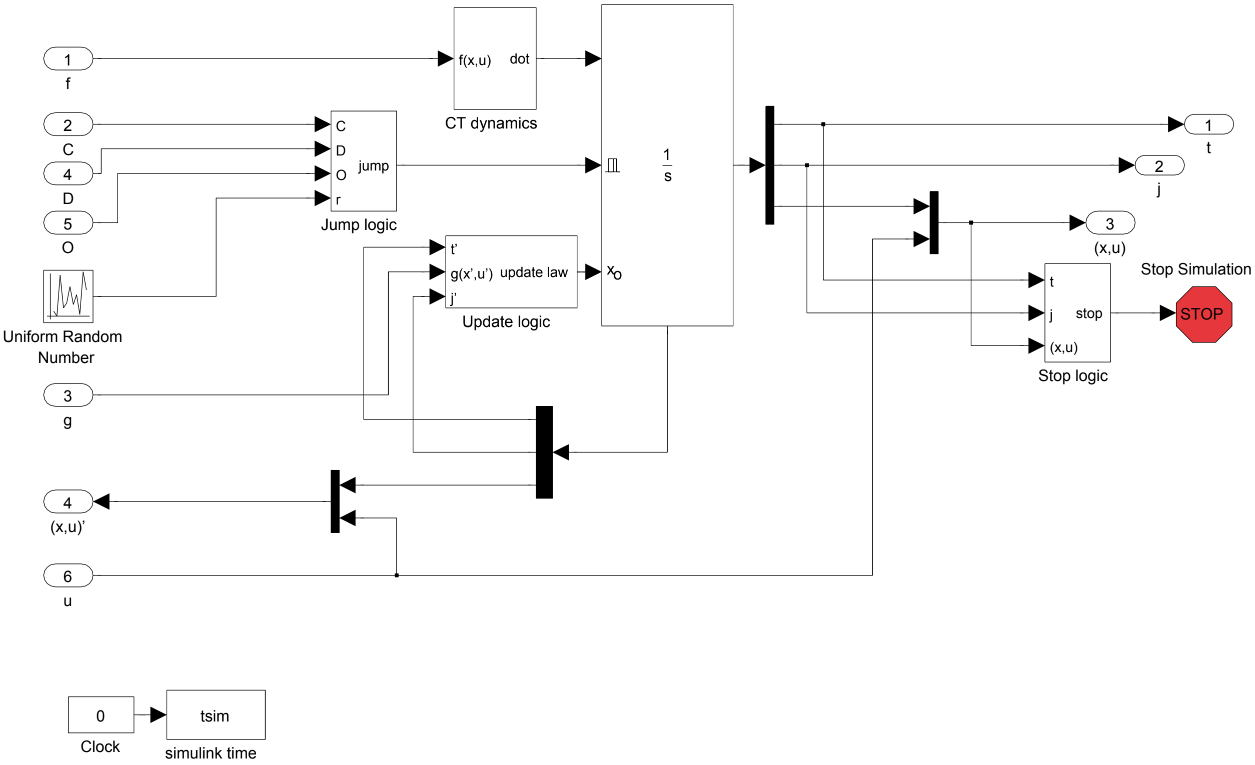

Figure 2: Integrator System

2 CT Dynamics

This block defines the continuous dynamics of the state $[t\ j\ x^T]^T$. These are given by

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

Figure 3 depicts this implementation. Note that input port 1 takes the value of $f(x,u)$ through the output of the *Matlab function block f* in Figure 1.

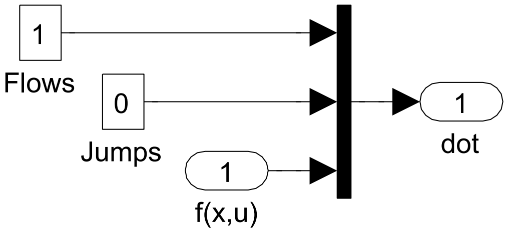

Figure 3: CT dynamics

3 Jump Logic

The inputs to the jump logic block are the output of the blocks *C, D,* and *O* indicating whether the state is in those sets or not, and a random signal with uniform distribution in $[0,1]$. Figure 4 shows that these signals are the input of a Matlab function block called *jump priority.* (The initial condition blocks set the initial value of the signals. These depend on the initial condition $z0$.) The *jump priority* block runs the following function (jumpPriority.m):

```
function out = jumpPriority(u,rule)

% state
flowFlag = u(1);
jumpFlag = u(2);
stateFlag = u(3);
randomInput = u(4);

% rule = 1 -> priority for jumps
% rule = 2 -> priority for flows
% rule = 3 -> no priority, random selection
%                   when simultaneous conditions

if (rule == 1) & (jumpFlag == 1)
   out = 1;
elseif (rule == 1) & (jumpFlag == 0)
   out = 0;
elseif (rule == 2) & (flowFlag == 1)
   out = 0;
elseif (rule == 2) & (flowFlag == 0) & (jumpFlag == 0)
   out = 0;
elseif (rule == 2) & (flowFlag == 0) & (jumpFlag == 1)
   out = 1;
elseif (rule == 3)
   if (flowFlag == 1) & (jumpFlag == 0)
      out = 0;
```

```
    elseif (flowFlag == 0) & (jumpFlag == 1)
       out = 1;
    elseif (flowFlag == 1) & (jumpFlag == 1)
       if (randomInput >= 0.5)
          out = 1;
       else
          out = 0;
       end
    else
        out = 0;
    end
end
```

The output of this function is equal to one only when the output of the *D block* is equal to one and $rule = 1$, or when 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 either event, 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 this input remains set to one, multiple resets would be triggered).

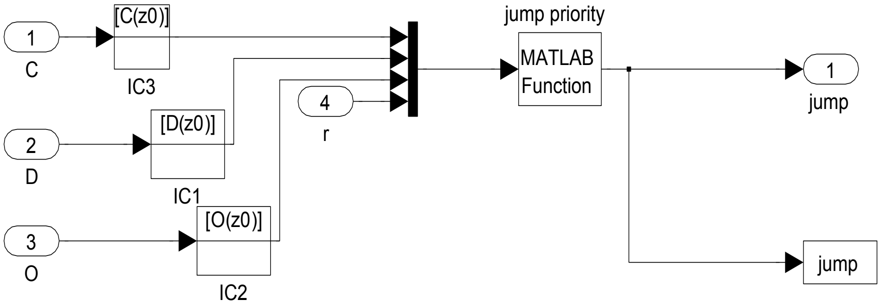

Figure 4: Jump Logic

4 Update Logic

The update logic uses the *state port* information of the integrator. This port reports the value of the state of the integrator, $[t\ j\ x^T]^T$, 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, that is, $x \in D$, while x at that same time is equal to the value assigned at the jump by the update logic. This value is given by $g(x', u')$ as Figure 5 illustrates. It also shows that the flow time t is kept constant at jumps and that j is incremented by one by the Matlab function block $j + 1$. More precisely

$$t^+ = t', \qquad j^+ = j', \qquad x^+ = g(x', u')$$

where $[t'\ j'\ x'^T]^T$ is the state that triggers the jump.

5 Stop Logic

This block, shown in Figure 6, 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.

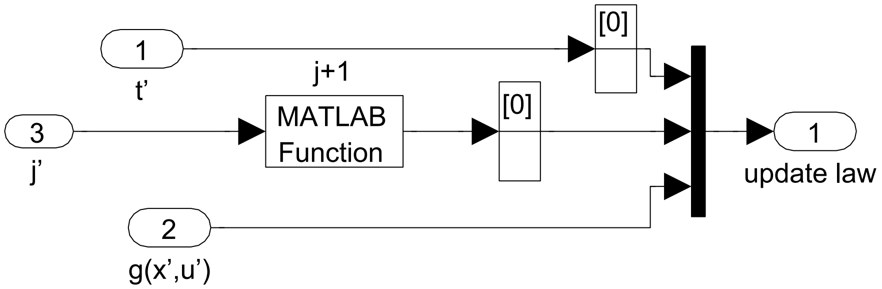

Figure 5: Update Logic

- The state of the hybrid system x is neither in C nor in D, or it is not in O.

Under any of these events, the output of the logic operator connected to the *Stop block* becomes one, stopping the simulation. Note that the blocks computing whether the state is in C, D, and O use the current value of the state x.

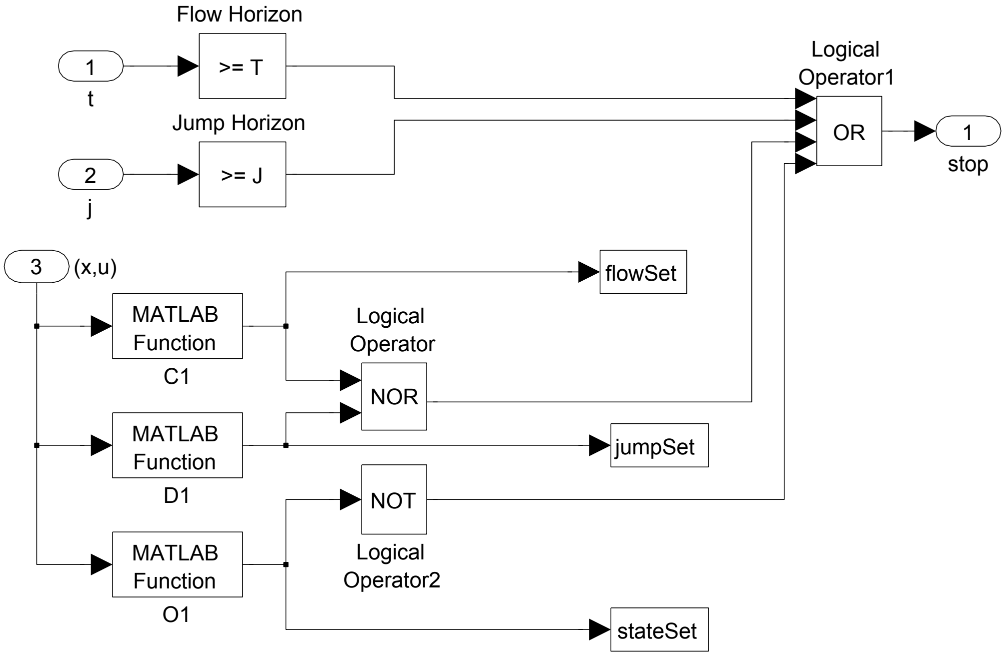

Figure 6: Stop Logic

6 Examples

The examples below illustrate the use of the implementation above. 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.
- 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).

Example 1.1 (bouncing ball) For the simulation of the bouncing ball system with regular data, and no external input, given by

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

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

where $\gamma > 0$ is the gravity constant and $\lambda \in [0,1)$ is the restitution coefficient. The Matlab scripts in each of the function blocks of the implementation above are given as follows. The constants for the bouncing ball system are $g = 9.8$ and $\lambda = 0.8$.

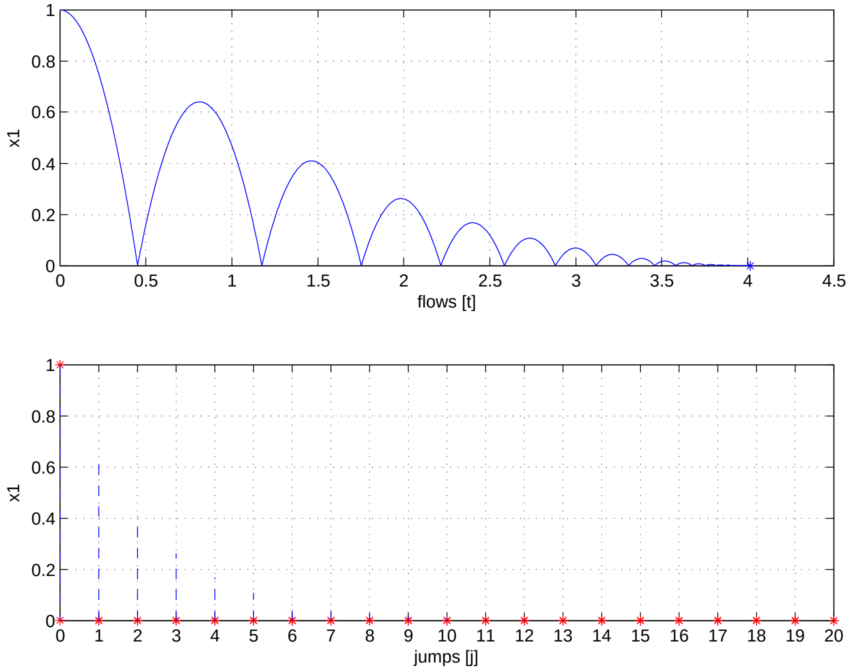

Figure 7: Solution to the bouncing ball example: height.

```
function out = f(n)
% state
x1 = n(1);
x2 = n(2);
% flow map
x1dot = x2;
x2dot = -9.8;
out = [x1dot; x2dot];
```

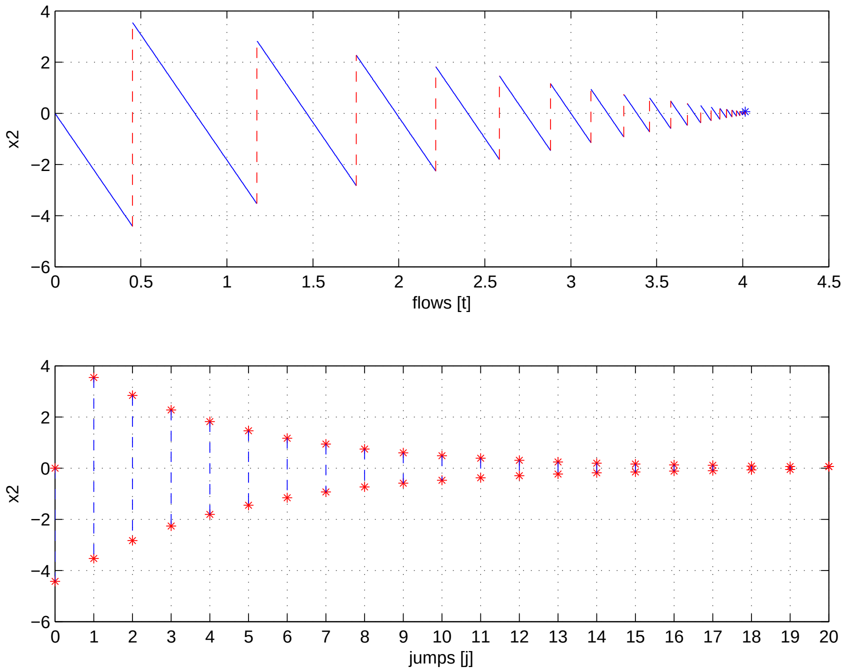

Figure 8: Solution to the bouncing ball example: velocity.

```
function [v] = C(n)
% state
x1 = n(1);
x2 = n(2);
if (x1 >= 0)  % flow condition
    v = 1;  % report flow
else
    v = 0;   % do not report flow
end

function out = g(n)
% state
x1 = n(1);
x2 = n(2);
% jump map
x1plus = 0;
x2plus = -0.8*x2;
out = [x1plus; x2plus];

function [v] = D(n)
% state
x1 = n(1);
x2 = n(2);
if (x1 <= 0 && x2 <= 0)  % jump condition
    v = 1;  % report jump
else
    v = 0;   % do not report jump
```

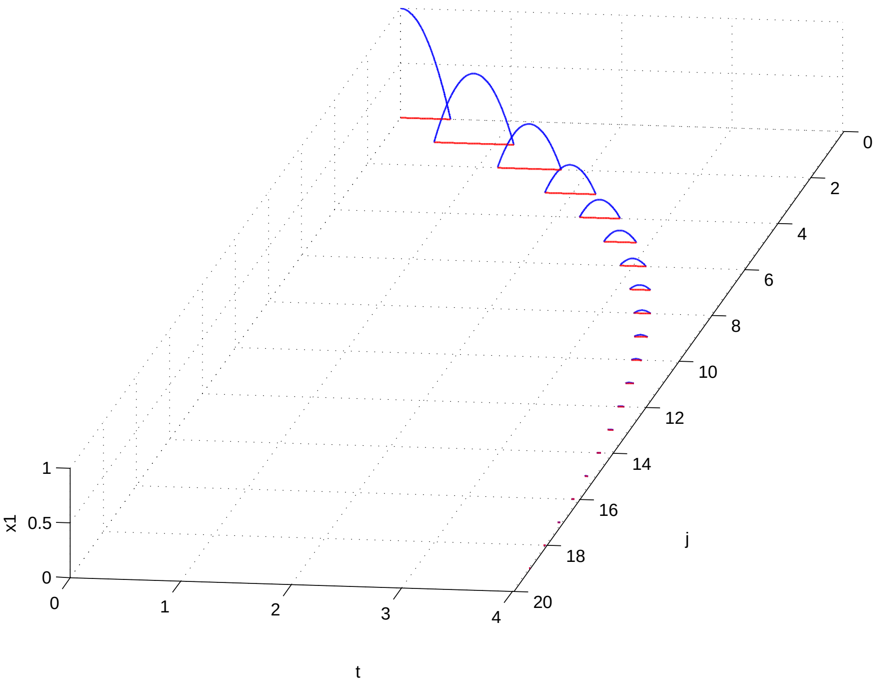

Figure 9: Hybrid arc corresponding to a solution to the bouncing ball example: height.

```
end

function [v] = O(n)
v = 1;  % in the state space
```

A solution to the bouncing ball system from $[1, 0]$ and with $T = 10, J = 20$, $rule = 1$, is depicted in Figure 7 (height) and Figure 8 (velocity). Both the projection onto t and j are shown. Figure 9 depicts the corresponding hybrid arc.

These simulations reflect the expected behavior of the bouncing ball model. However, if instead

$$\left\{x \in \mathbb{R}^2 \ \mid x_1 = 0 \ , \ x_2 \leq 0\right\}$$

is used as D, the effect of the discretization of the flows would prevent the *Jump Logic block* from detecting the jumps since it is very unlikely that the computed solution would hit $x_1 = 0$ exactly. The enlarged jump set prevents this from happening. Another way to prevent such behavior is by adding special Simulink blocks with zero-cross detection.

Also note that using $[x_1, -\gamma x_2]^T$ as the jump map would cause the simulation to stop after the first jump. In fact, at the jump, due to the discretization effect of the flows mentioned above, $x_1, x_2 < 0$. With this new jump map, $x_1^+ < 0, x_2^+ > 0$ and this state is neither in C nor in D. Hence, the simulation is stopped after the first jump.

◻

Example 1.2 (bouncing ball with input) For the simulation of the bouncing ball system with a constant input and regular data given by

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

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

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 $g = 9.81$ and $\lambda = 0.8$.

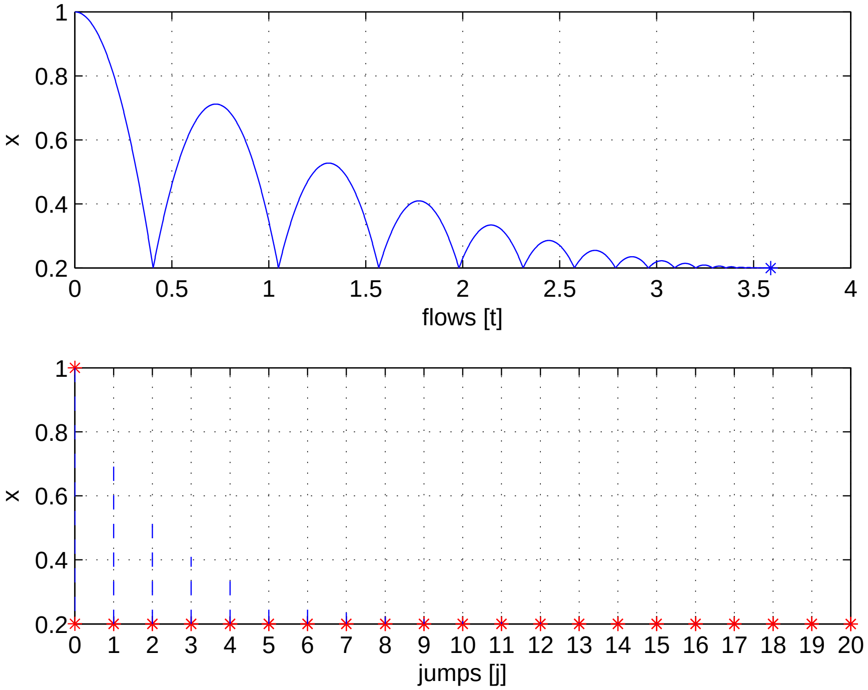

Figure 10: Solution to the bouncing ball with input example: height

```
function out = f(z)
% state
x1 = x(1);
x2 = x(2);
%input
u=u(1)
% flow map
x1dot = x2;
x2dot = -9.81;
out = [x1dot; x2dot];

function [v] = C(z)
% state
x1 = x(1);
x2 = x(2);
if (x1 >= u)  % flow condition
```

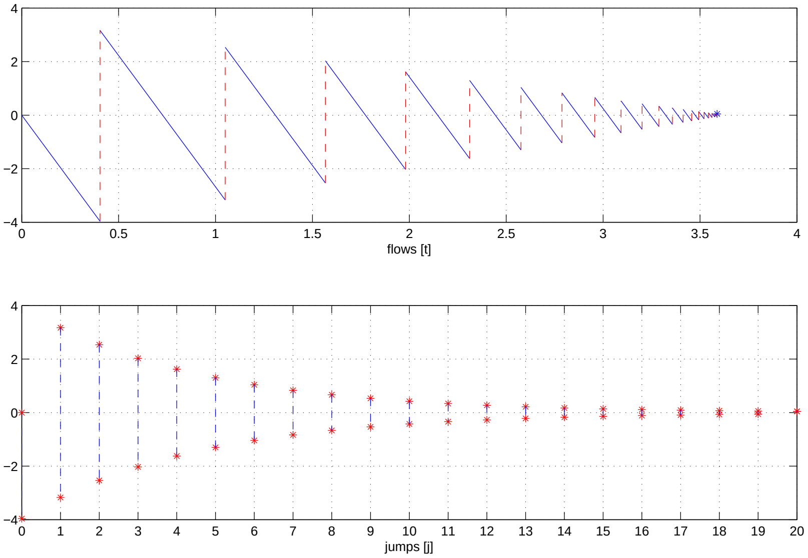

Figure 11: Solution to the bouncing ball with input example: velocity

```
    v = 1;  % report flow
else
    v = 0;   % do not report flow
end

function out = g(z)
% state
x1 = x(1);
x2 = x(2);
% jump map
x1plus = u;
x2plus = -0.8*x2;
out = [x1plus; x2plus];

function [v] = D(z)
% state
x1 = x(1);
x2 = x(2);
if (x1 <= u && x2 <= 0)  % jump condition
    v = 1;  % report jump
else
    v = 0;   % do not report jump
end

function [v] = O(z)
v = 1;  % in the state space
```

A solution to the bouncing ball system from $[1, 0]$ and with $T = 10, J = 20$, $rule = 1$, is depicted in Figure 10 (height) and Figure 11 (velocity). Both the projection onto t and j are shown. Figure 12 depicts the corresponding hybrid arc.

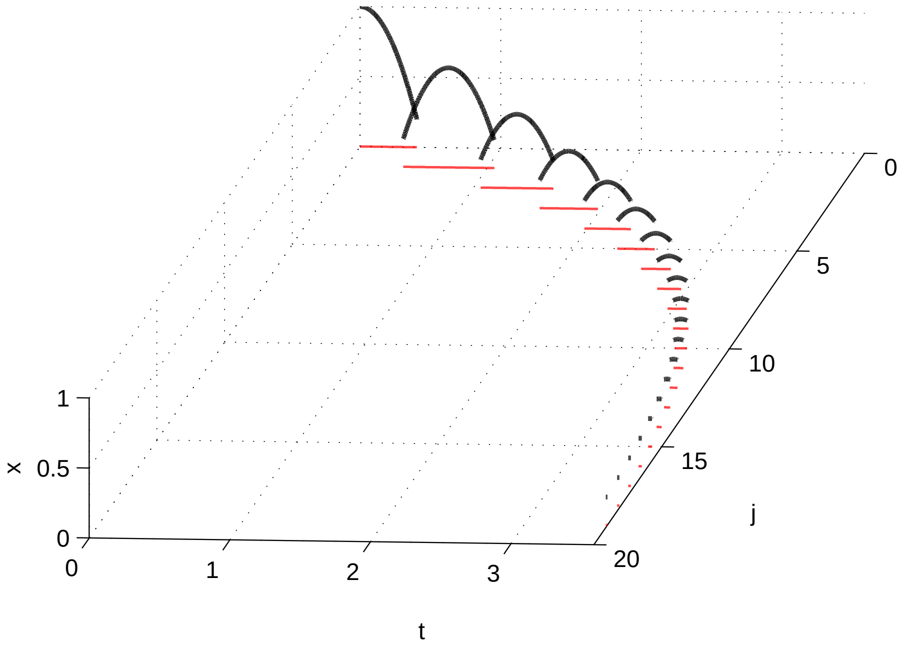

Figure 12: Hybrid arc corresponding to a solution to the bouncing ball with input example: height

These simulations reflect the expected behavior of the bouncing ball model. Note the only difference between this example and the previous is that the constant input has caused the ball to no longer bounce on a floor at the value of 0, but rather it bounces at the chosen input value of 0.2.

◻

Example 1.3 (vehicle following a track with boundaries)

Consider a vehicle traveling along a given track modeled by a Dubins vehicle model with state x where x is a vector with three components given by $\dot{\xi}_1 = v\cos\xi_3$, $\dot{\xi}_2 = v\sin\xi_3$, and $\dot{\xi}_3 = u$. v is the tangential velocity of the vehicle, ξ_1 and ξ_2 describe the vehicle's position, and ξ_3 is the vehicle's orientation angle. Also consider a switching controller attempting to keep the vehicle inside the boundaries of the track while traveling. A state $q \in \{1,2\}$ is used to define the modes of operation of the controller. The state of the closed-loop system is given by $x := [\xi^\top \ q]^\top$. For the simulation of the described system with a constant input and regular data given by

$$O := \mathbb{R}^3 \times \{1,2\} \times \mathbb{R}, f(x,u) := \begin{bmatrix} \xi \\ q \end{bmatrix}, \tag{6}$$

$$C := \big\{(x,u) \in \mathbb{R}^2 \times \mathbb{R} \mid (\xi_1 \le 1, q = 2) \cup (\xi_1 \ge -1, q = 1)\big\} \tag{7}$$

$$g(x,u) := \begin{bmatrix} \xi \\ q \end{bmatrix}, D := (\mathbb{R}^2 \times \mathbb{R}) \setminus C \tag{8}$$

When $q = 1$, the vehicle is traveling to the left, and when $q = 2$, the vehicle is traveling to the right. 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 $v = 1$, and the initial orientation angle is chosen to be $x_3 = \pi/4$ radians.

```
function out = f(z)
```

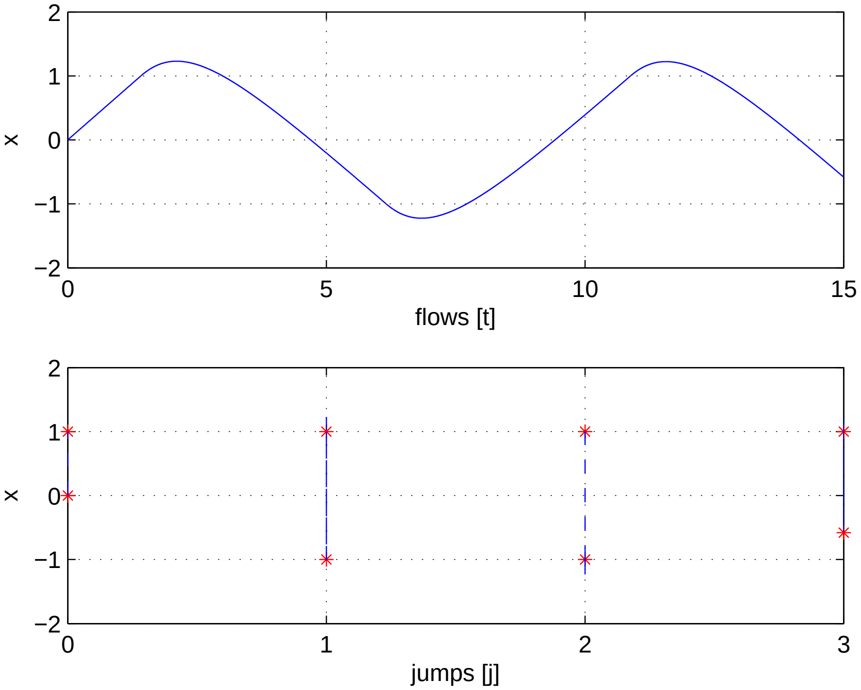

Figure 13: Solution to the Dubins path with boundaries: trajectory

```
v = 1; %tangential velocity
% state
x1 = x(1);      %x-position
x2 = x(2);      %y-position
x3 = x(3);      %orientation angle
q = x(4);
%input
u1 = u(1);
% q = 1 --> going left
% q = 2 --> going right
if q == 1
    r = 3*pi/4;
elseif q == 2
    r = pi/4;
else
    r = 0;
end
% flow map
x1dot = v*cos(x3);  %tangential velocity in x-direction
x2dot = v*sin(x3);  %tangential velocity in y-direction
x3dot = -x3 + r;    %angular velocity
qdot = 0;
out = [x1dot;x2dot;x3dot;qdot];

function [v] = C(z)
% state
```

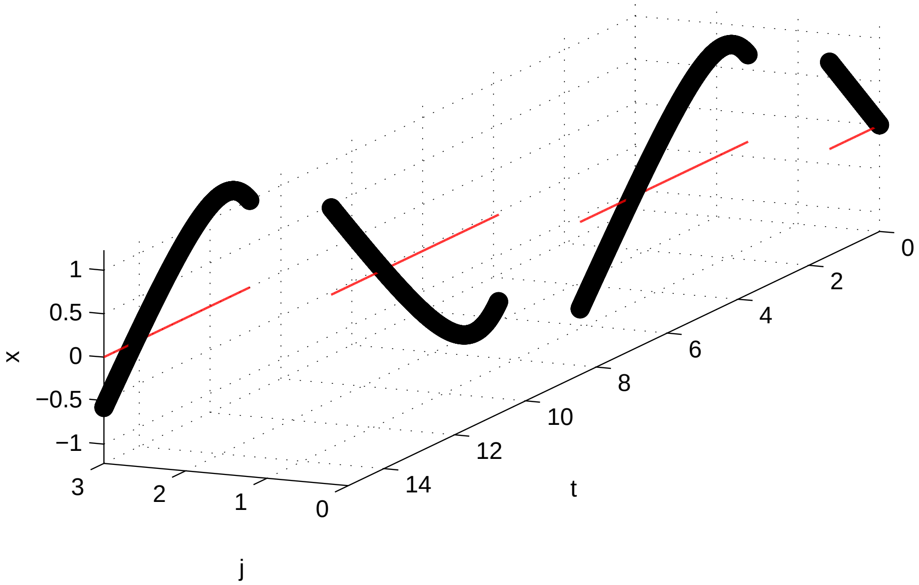

Figure 14: Hybrid arc corresponding to a Dubins path

```
x1 = x(1);       %x-position
x2 = x(2);       %y-position
x3 = x(3);       %orientation angle
q = x(4);
%input
u1=u(1);
% q = 1 --> going left
% q = 2 --> going right
if (x1 < 1) && (q == 2)  % flow condition
    v = 1;  % report flow
elseif (x1 > -1) && (q == 1) %flow condition
    v = 1;  %report flow
else
    v = 0;   % do not report flow
end

function out = g(z)
% state
x1 = x(1);       %x-position
x2 = x(2);       %y-position
x3 = x(3);       %orientation angle
q = x(4);
%input
u1=u(1);
% q = 1 --> going left
% q = 2 --> going right
```

```
x1plus=x1;
x2plus=x2;
x3plus=x3;
qplus = q;
%jump map
if (x1 >= 1) && (q == 2)
   qplus = 3-q;
elseif (x1 <= -1) && (q == 1)
   qplus = 3-q;
else
   qplus = 0;
end
out = [x1plus;x2plus;x3plus;qplus];
```

```
function [v] = D(z)
% state
x1 = x(1);      %x-position
x2 = x(2);      %y-position
x3 = x(3);      %orientation angle
q = x(4);
%input
u1=u(1);
% q = 1 --> going left
% q = 2 --> going right
 if (x1 >= 1) && (q == 2)  % jump condition
     v = 1;  % report jump
 elseif (x1 <= -1) && (q == 1)  % jump condition
     v = 1;   % report jump
 else
     v = 0; % do not report jump
 end
```

```
function [v] = O(z)
v = 1;  % in the state space
```

A solution to the system following a Dubins path with boundaries from $[-1,1]$ and with $T = 15, J = 10$, *rule* $= 1$, is depicted in Figure 13 (trajectory). Both the projection onto t and j are shown. Figure 14 depicts the corresponding hybrid arc.

□

Example 1.4 (interconnection of hybrid systems $\mathcal{H}1$ (bouncing ball) and $\mathcal{H}2$ (moving floor))

Consider a bouncing ball ($\mathcal{H}1$) bouncing on a floor and a floor ($\mathcal{H}2$) at some initial height and converging to the ground with a height equal to zero. With this interconnection, the bouncing ball will contact the floor, bounce back up, and cause a jump in height of the floor so that it gets closer to the ground. After some time, both the ball and the floor will converge to the ground. In order to model this system, the output of the bouncing ball becomes the input of the moving floor, and vice versa. For the simulation of the described system with regular data where $\mathcal{H}_1$ is given by

$$O_1 := \mathbb{R}^2 \times \mathbb{R}, f_1(\xi, u_1, v_1) := \begin{bmatrix} \xi_2 \\ -\gamma - b\xi_2 + v_{11} \end{bmatrix}, C_1 := \{(\xi, u_1) \mid \xi_1 \geq u_1, u_1 \geq 0\} \tag{9}$$

$$g_1(\xi, u_1, v_1) := \begin{bmatrix} \xi_1 + \alpha_1 \xi_2^2 \\ e_1|\xi_2| + v_{12} \end{bmatrix}, D_1 := \{(\xi, u_1) \mid \xi_1 = u_1, u_1 \geq 0\}, y_1 = h_1(\xi) := \xi_1 \tag{10}$$

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

$$O_2 := \mathbb{R}^2 \times \mathbb{R}, f_2(\eta, u_2, v_2) := \begin{bmatrix} \eta_2 \\ -\eta_1 - 2\eta_2 + v_{12} \end{bmatrix}, C_2 := \{(\eta, u_2) \mid \eta_1 \leq u_2, \eta_1 \geq 0\} \tag{11}$$

$$g_2(\eta, u_2, v_2) := \begin{bmatrix} \eta_1 - \alpha_2|\eta_2| \\ -e_2|\eta_2| + v_{22} \end{bmatrix}, D_2 := \{(\eta, u_2) \mid \eta_1 = u_2, \eta_1 \geq 0\}, y_2 = h_2(\eta) := \eta_1 \tag{12}$$

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

$$u_1 = y_2, \qquad u_2 = y_1. \tag{13}$$

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 $\alpha_1, \alpha_2 = 0.1$.

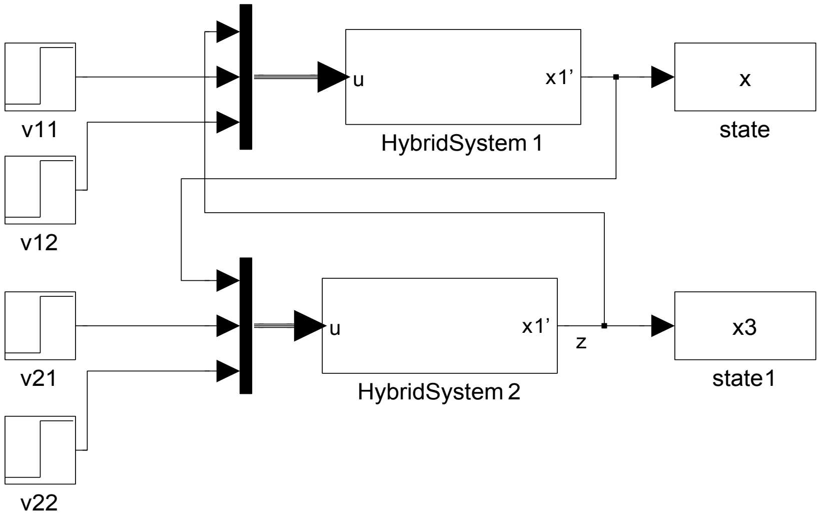

Figure 15: Matlab/Simulink implementation of interconnected hybrid systems $\mathcal{H}1$ and $\mathcal{H}2$

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

```
global n m;
% n = # of state components
% m = # of input components
function out = f1(z)
% state
x = z(1:n);
x1 = x(1);
x2 = x(2);
%input
u = z(n+1:n+m);
u1 = u(1);
u2 = u(2);
u3 = u(3);
% flow map
x1dot = x2;
x2dot = -0.8-0.1*x2+u2;
out = [x1dot;x2dot];
```

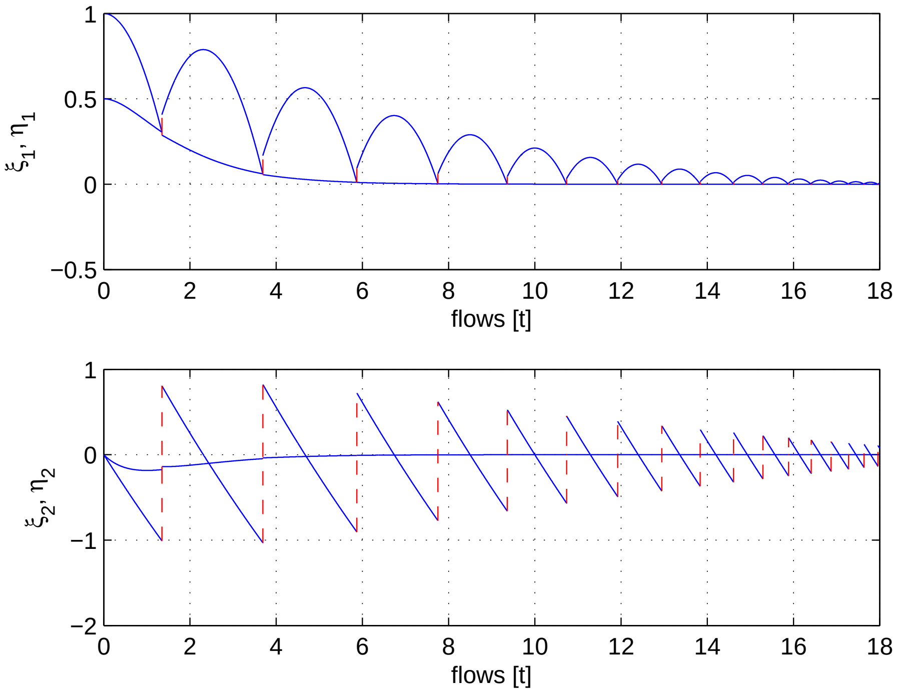

Figure 16: Solution to interconnection example: height

```
function [v] = C1(z)
% state
x = z(1:n);
x1 = x(1);
x2 = x(2);
%input
u = z(n+1:n+m);
u1 = u(1);
u2 = u(2);
u3 = u(3);
if (x1 >= u1)  % flow condition
    v = 1;  % report flow
else
    v = 0;   % do not report flow
end

function out = g1(z)
% state
x = z(1:n);
x1 = x(1);
x2 = x(2);
%input
u = z(n+1:n+m);
u1 = u(1);
u2 = u(2);
u3 = u(3);
```

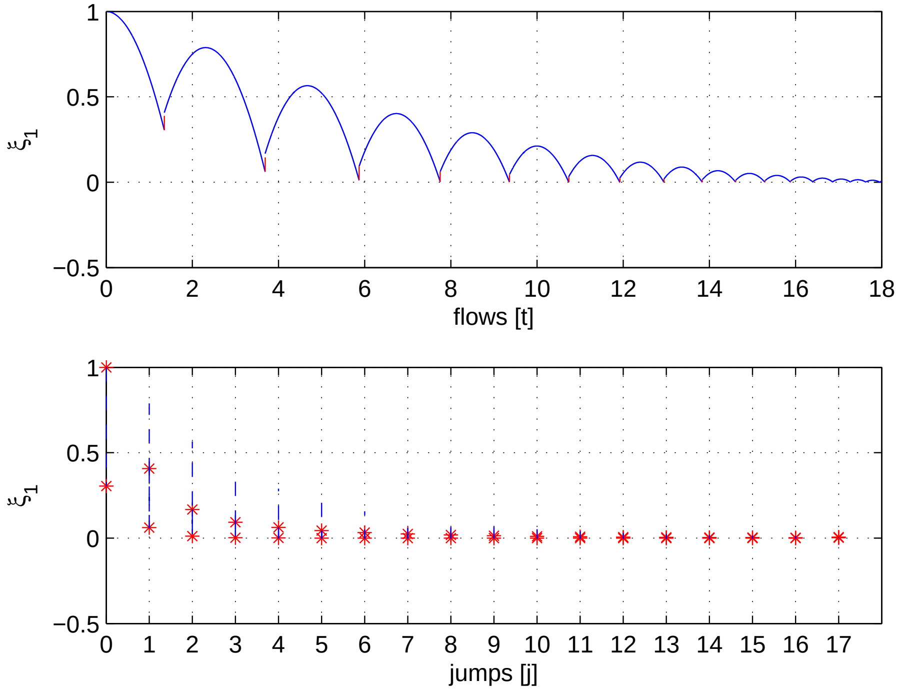

Figure 17: Solution to interconnection example for system $\mathcal{H}1$: height

```
%jump map
x1plus=u1+0.1*x2^2;
x2plus=0.8*abs(x2)+u3;
out = [x1plus;x2plus];

function [v] = D1(z)
% state
x = z(1:n);
x1 = x(1);
x2 = x(2);
%input
u = z(n+1:n+m);
u1 = u(1);
u2 = u(2);
u3 = u(3);
if (x1 <= u1) % jump condition
    v = 1;  % report jump
else
    v = 0;   % do not report jump
end

function [v] = O1(u)
v = 1;  % in the state space
```

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

```
function out = f2(z)
```

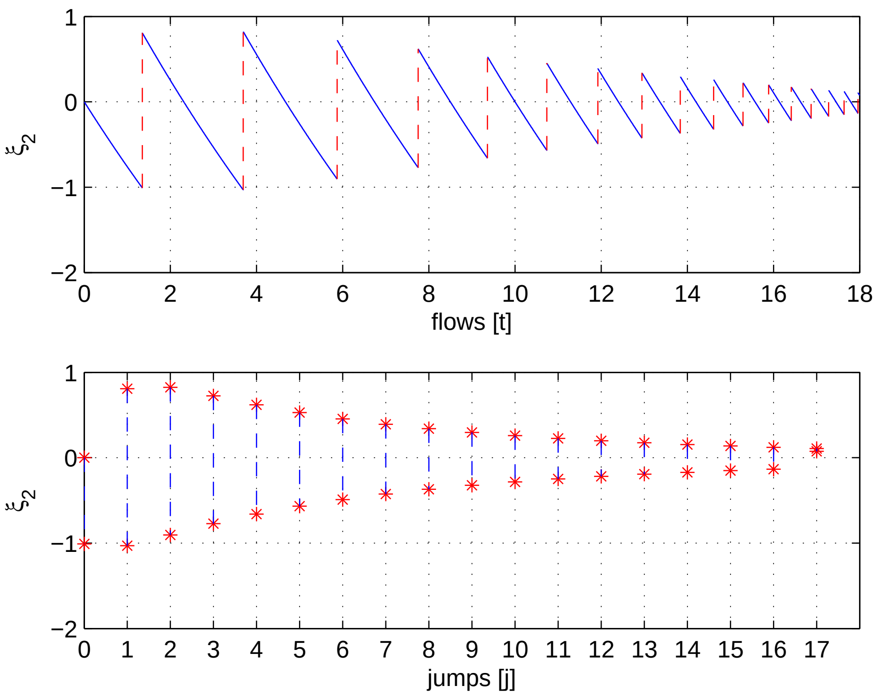

Figure 18: Solution to interconnection example for system $\mathcal{H}1$: velocity

```
% state
x = z(1:n);
x1 = x(1);
x2 = x(2);
%input
u = z(n+1:n+m);
u1 = u(1);
u2 = u(2);
u3 = u(3);
% flow map
x1dot = x2;
x2dot = -x1-2*x2+u2;
out = [x1dot;x2dot];

function [v] = C2(z)
% state
x = z(1:n);
x1 = x(1);
x2 = x(2);
%input
u = z(n+1:n+m);
u1 = u(1);
u2 = u(2);
u3 = u(3);
if (x1 <= u1)  % flow condition
    v = 1;  % report flow
```

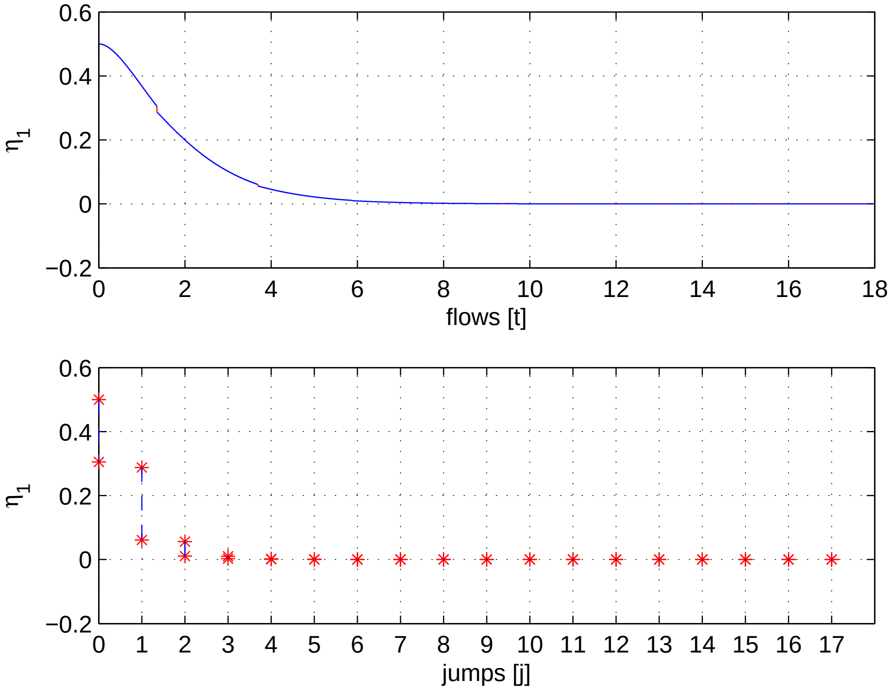

Figure 19: Solution to interconnection example for system $\mathcal{H}2$: height

```
else
    v = 0;   % do not report flow
end

function out = g2(z)
% state
x = z(1:n);
x1 = x(1);
x2 = x(2);
%input
u = z(n+1:n+m);
u1 = u(1);
u2 = u(2);
u3 = u(3);
% jump map
x1plus = u1-0.1*abs(x2);
x2plus = -0.8*abs(x2)+u3;
out = [x1plus;x2plus];

function [v] = D2(z)
% state
x = z(1:n);
x1 = x(1);
x2 = x(2);
%input
u = z(n+1:n+m);
u1 = u(1);
```

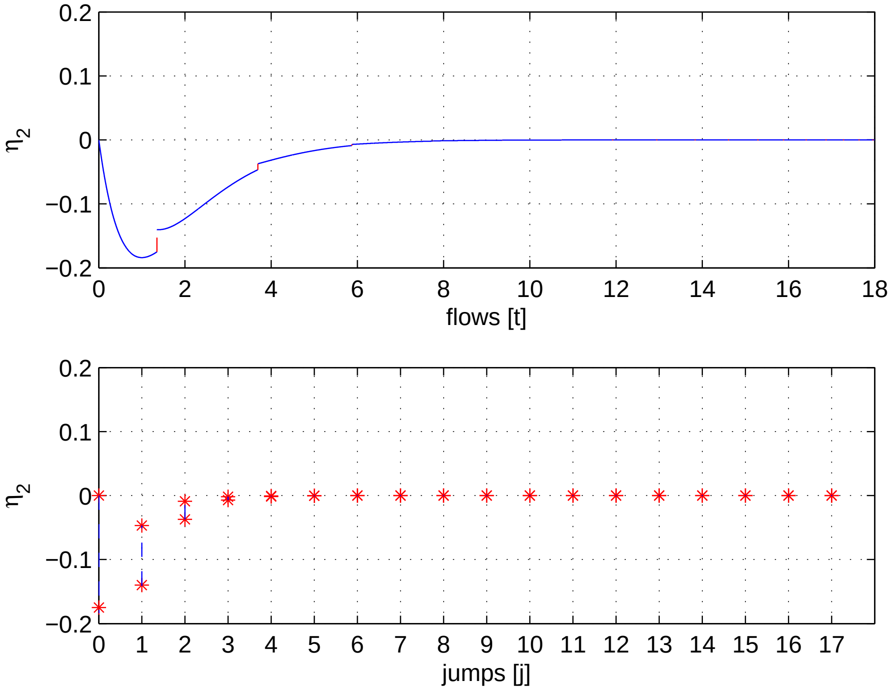

Figure 20: Solution to interconnection example for system $\mathcal{H}2$: velocity

```
u2 = u(2);
u3 = u(3);
if (x1 >= u1) % jump condition
    v = 1;  % report jump
else
    v = 0;   % do not report jump
end

function [v] = O2(u)
v = 1;  % in the state space
```

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 16. Both the projection onto t and j are shown. A solution to the hybrid system $\mathcal{H}1$ is depicted in Figure 17 (height) and Figure 18 (velocity). A solution to the hybrid system $\mathcal{H}2$ is depicted in Figure 19 (height) and Figure 20 (velocity).

These simulations reflect the expected behavior the the interconnected hybrid systems. Note that in order to implement these systems without premature stopping of the simulation, ξ_1 in g_1 and η_1 in g_2 can be changed to u_1 and u_2, respectively so that $\xi_1^+ = u_1$ and $\eta_1^+ = u_2$.

$\square$

Example 1.5 (a simple mathematical example to show different type of simulation results)
Consider the hybrid system with data

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

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. The solution jumps from 1 to 2, and from 2 to 1 repetitively.

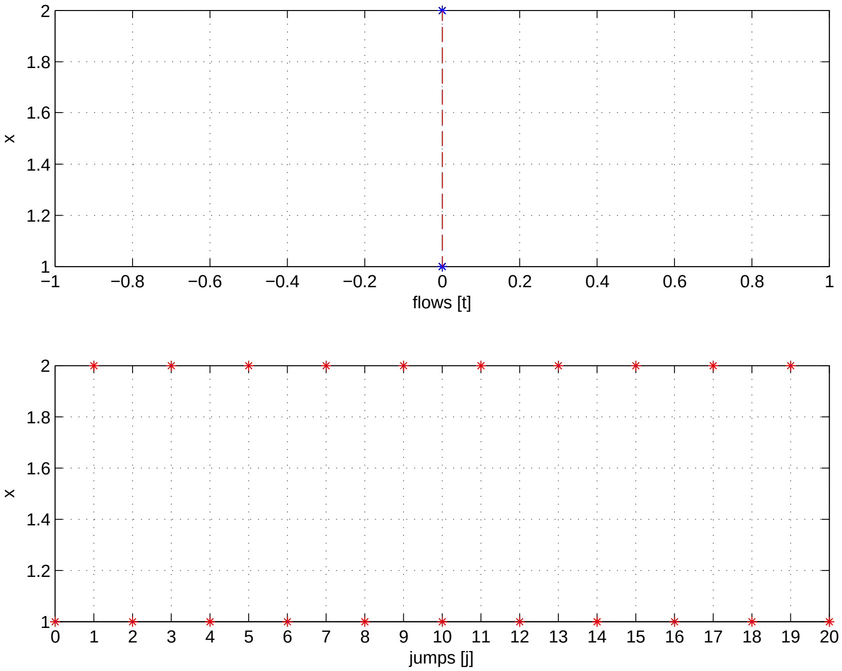

Figure 21: Solution to Example 1.5 with forced jumps logic.

Flows enforced:

A solution from $x0 = 1$ with $T = 10, J = 20$, $rule = 2$ is depicted in Figure 22. 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 23. 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 24 with the same parameters used in the simulation in Figure 23. The plot suggests that the solution jumps several times until $x < 1/50$ from where it flows to zero. However, Figure 25, a zoomed version of Figure 24, 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

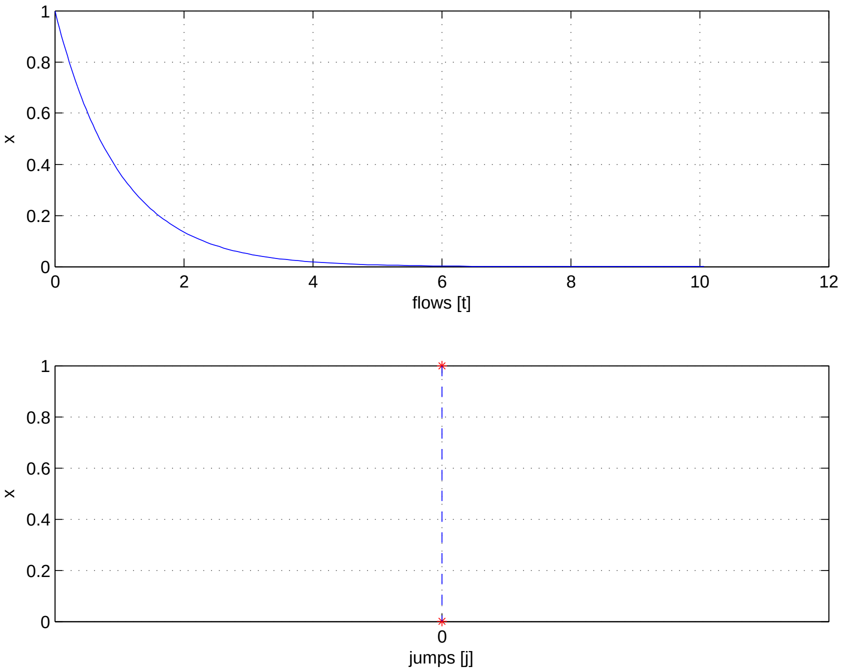

Figure 22: Solution to Example 1.5 with forced flows logic.

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.

$\square$

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

Solution outside O:

Replacing O by $(-1, 2)$, a solution starting from $x0 = 1$ with $T = 10, J = 20$, $rule = 1$ fails to exists after the first jump. This is depicted in Figure 26 (cf. Figure 21)

Solution outside $C \cup D$:

The same behavior as the one just outlined arises with $O = \mathbb{R}$ but with $D = \{1\}$. The simulation stops since the solution leaves $C \cup D$. (See also Overlap3.zip).

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

Finally, taking $O = \mathbb{R}$ and 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 27 shows this.

Note that in this implementation, the Stop Logic is such that when the state of the hybrid system is not in $(C \cup D) \cap O$, then the simulation is stopped. In particular, if this condition becomes true while flowing,

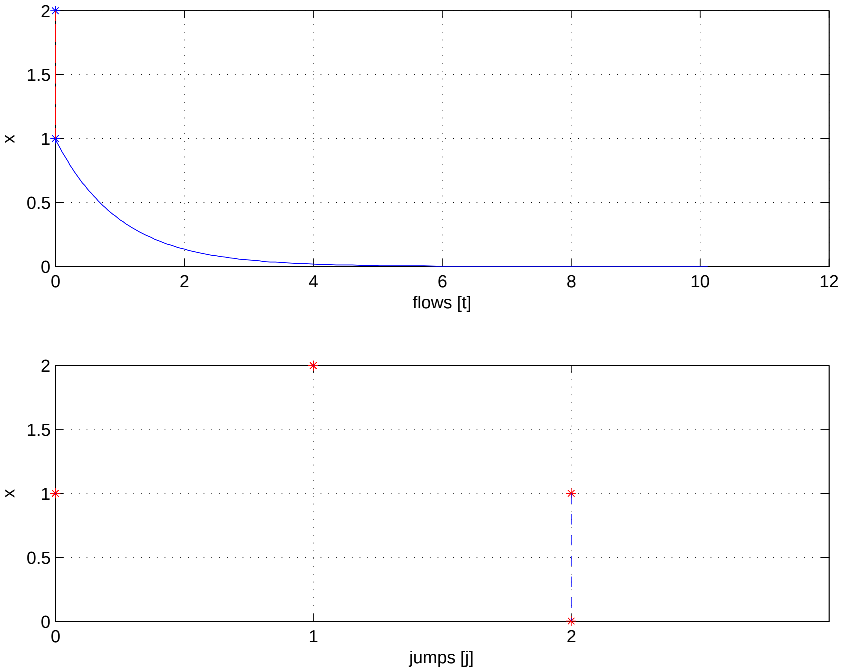

Figure 23: Solution to Example 1.5 with random logic for flowing/jumping.

then the last value of the computed solution will not belong to either C or O, or both, depending on the situation. 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.

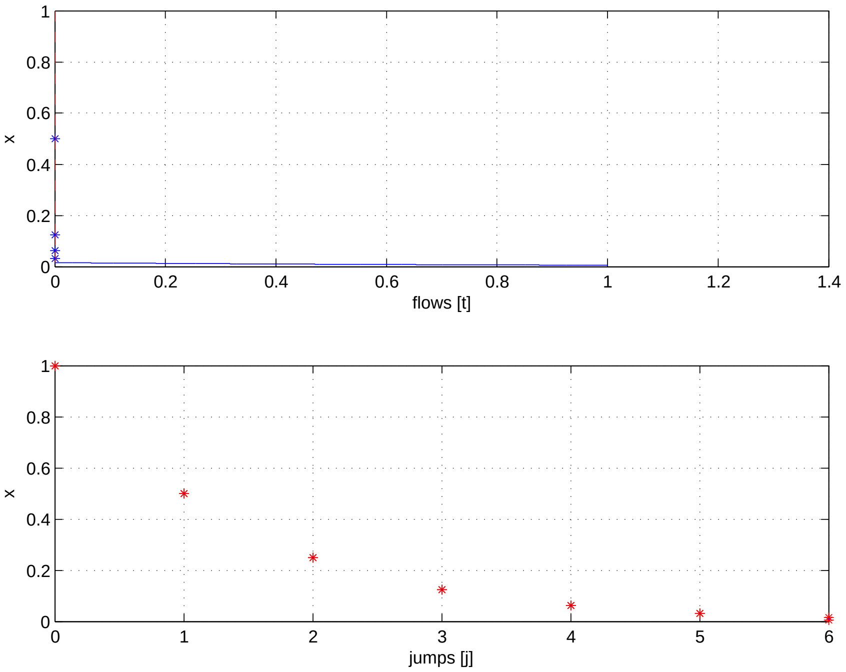

Figure 24: Solution to Example 1.5 with random logic for flowing/jumping.

7 Notes

Matlab/Simulink files corresponding to the simulation technique described in this paper can be found at Matlab Central and at the author's website

http://www.u.arizona.edu/∼sricardo/.

8 References

[1] R. G. Sanfelice. *Simulating Hybrid Systems in Matlab/Simulink, v0.3.* Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

[2] R. Goebel, R. G. Sanfelice, and A. R. Teel, Hybrid dynamical systems. IEEE Control Systems Magazine, 28-93, 2009.

[3] R. G. Sanfelice and A. R. Teel, Dynamical Properties of Hybrid Systems Simulators. Automatica, 46, No. 2, 239–248, 2010.

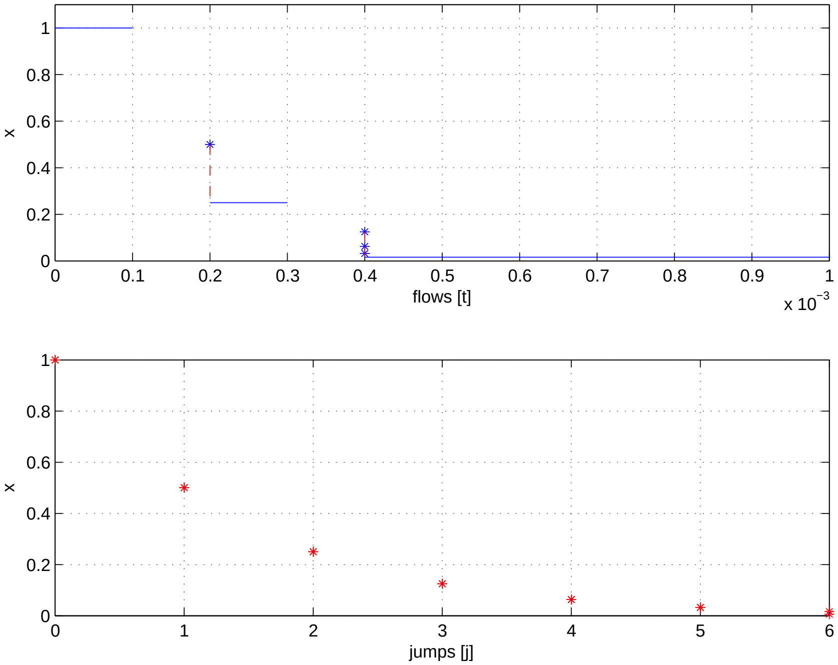

Figure 25: Solution to Example 1.5 with random logic for flowing/jumping. Zoomed version.

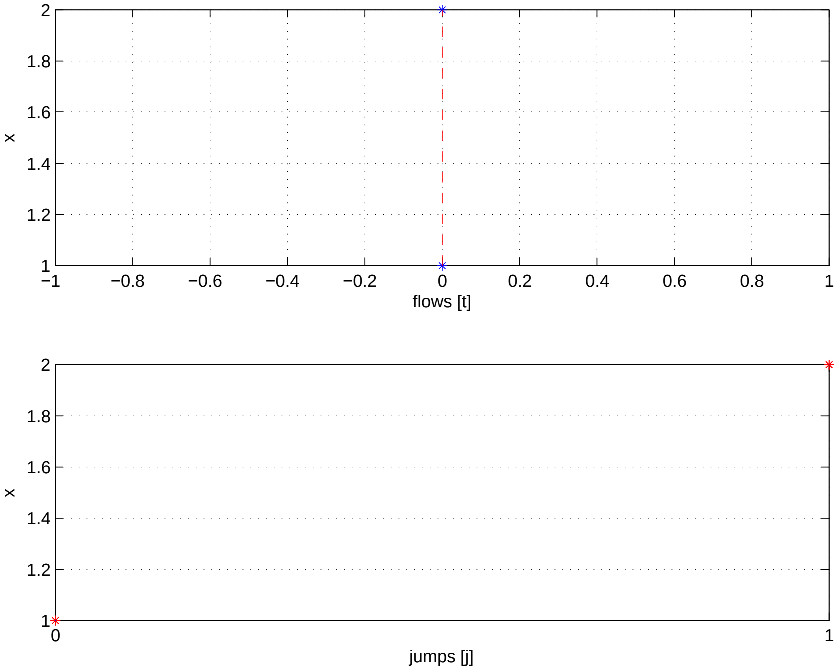

Figure 26: Solution to Example 1.5 with forced jump logic and different O.

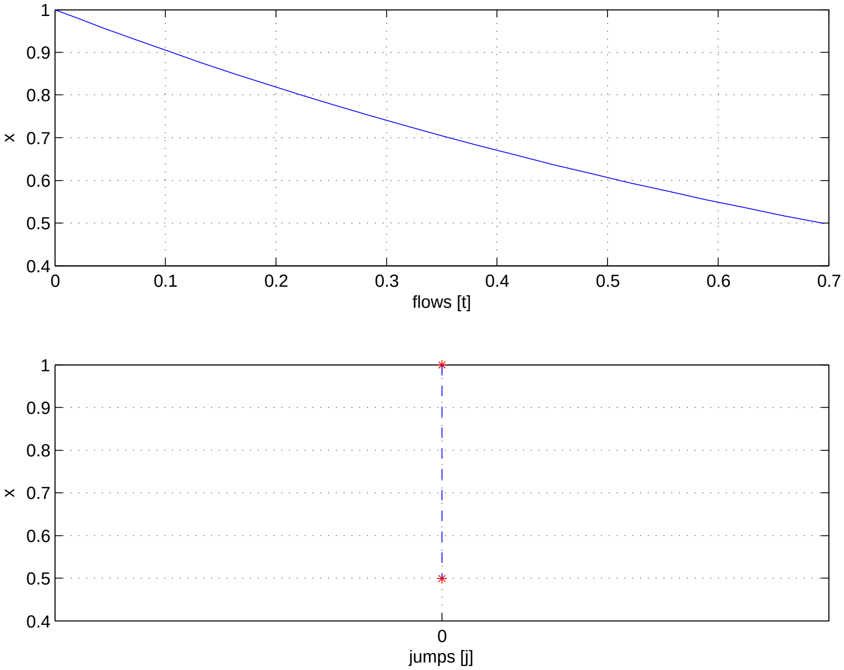

Figure 27: Solution to Example 1.5 with forced flow logic.