Numerical Algorithms (2021) 87:849–872
https://doi.org/10.1007/s11075-020-00990-x

ORIGINAL PAPER

Adaptive step size numerical integration for stochastic differential equations with discontinuous drift and diffusion

Avinash Malik[1]

Received: 3 June 2020 / Accepted: 23 July 2020 / Published online: 8 August 2020

Abstract
Stochastic hybrid systems (SHSs) are a modelling framework for a cyber-physical system (CPS), used to simulate, validate, and verify safety critical controllers under uncertainty. Popular simulation tools can miss detecting discontinuities when simulating SHS, thereby producing incorrect outputs during simulation. We propose a novel adaptive step size simulation/integration technique for a subset of SHS—stochastic differential equations (SDEs) with discontinuous drift and diffusion coefficients. Each integration step, of the Euler-Maruyama numerical solution of the SDEs, is made dependent upon the values of the continuous variables inducing the discontinuity. This in turn guarantees convergence of the system trajectory towards the discontinuity without missing it. A thorough analysis and extensive benchmarking of the proposed integration technique shows the efficacy of the approach when simulating complex SHSs.

Keywords Stochastic hybrid systems · Stochastic differential equations · Numerical integration algorithms · Convergence analysis

1 Introduction

Stochastic hybrid systems (SHSs) [1, 2] are a subset of cyber-physical systems [3] where the physical plant, with uncertainty, is captured using stochastic differential equations (SDEs), while control switches between different plant modes are captured as instantaneous transitions. SHSs have been used to model air traffic control [4], robust sliding mode control [5], communication networks [6], etc.

✉ Avinash Malik
avinash.malik@auckland.ac.nz

1 Department of Electrical, Computer and Software Engineering, University of Auckland, Auckland, New Zealand

Significant recent research literature exists elucidating the formal semantics [2, 7], and formal verification of SHS [8–10]. In contrast, very little research literature exists for efficient (or even functionally correct) numerical simulation techniques for SHS. The standard technique for numerical integration of SDEs is the Euler-Maruyama [11] fixed step size integration. Euler-Maruyama technique combines standard Euler technique for integrating ordinary differential equations (ODEs) with Itô' chain-rule [12] to compute the integral of a given SDE. Adaptive step size integration of SDEs, without discontinuities, has shown to perform efficiently compared to fixed step size integration [13, 14]. However, to the best of our knowledge, there exists no adaptive step size numerical integration/simulation algorithm for SHS, where the drift and diffusion coefficients of the SDE change with changing plant modes. In this paper, we remedy this situation: we first present the problem with fixed step size numerical integration/simulation for SHS in the de facto, industry standard, modelling tool Simulink® [15]. Next, we present an adaptive step size numerical integration/simulation technique for SHS.

1.1 Running example and problem description

Figure 1 shows an example of simulating sliding mode control (SMC) for an autonomous vehicle under uncertainty in Simulink®. Figure 1a shows the steering wheel of a car. The continuous variable $x(t)$ indicates the position of the steering wheel at any given point in time $t \in \mathbb{R}^{\geq 0}$. The steering wheel can move left or right from the centre, marked with the up arrow. By convention, the centre position is considered to be $\pi/2$ radians. The aim of the SMC is to maintain the steering wheel in the centre position, irrespective of the starting position ($x(0) \neq \pi/2$), even in case of uncertainty.

$$dx(t) = -sgn(x(t) - \pi/2)dt \tag{1}$$

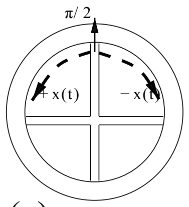
$$sgn(x(t)) = \begin{cases} 1, & x(t) > 0 \\ -1, & x(t) < 0 \\ 0, & otherwise \end{cases} \tag{2}$$

$$\frac{d}{dt}x(t) = \begin{cases} 1, & x(t) < \pi/2 \\ -1, & x(t) > \pi/2 + \epsilon \\ 0, & \pi/2 \leq x(t) \leq \pi/2 + \epsilon \end{cases} \tag{3}$$

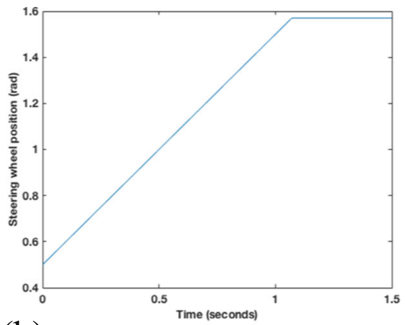
$$dx(t) = -sgn(x(t) - \pi/2)dt + sgn(x(t) - \pi/2)dW(t) \tag{4}$$

$$dx(t) = \begin{cases} (1) \times dt + (-1) \times dW(t), & x(t) < \pi/2 \\ (-1) \times dt + (1) \times dW(t), & x(t) > \pi/2 + \epsilon \\ 0, & \pi/2 \leq x(t) \leq \pi/2 + \epsilon \end{cases} \tag{5}$$

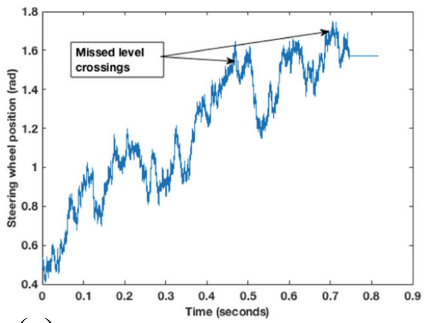
Equation (1) shows the SMC model without uncertainty, where the idealised *sgn* function is given in (2). A non-chattering implementation of the SMC is a hybrid automaton (HA) [16], as shown in (3). This HA has three locations, with invariant bounding $x(t)$ on each location as specified in (3). In order to avoid chattering, the



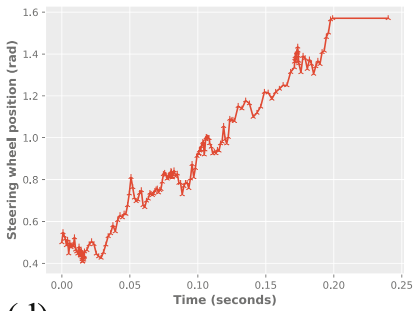
(a) Steering wheel



(b) SMC output without uncertainty



(c) Incorrect SHS output from Simulink



(d) Correct SHS output from the proposed technique

Fig. 1 Sliding mode control for maintaining steering wheel position of an autonomous car under uncertainty. **a** shows the steering wheel model. **b** shows the correct steering wheel position output, from simulink, without uncertainty. **c** shows incorrect steering wheel position output, from simulink, under uncertainty. **d** shows the correct steering wheel position output, from the proposed technique, under uncertainty

continuous variable $x(t)$ stops evolving once $\pi/2 \leq x(t) \leq \pi/2 + \epsilon$, where $\epsilon \in \mathbb{R}^{>0}$ is an arbitrarily small value close to the sliding surface — point $\pi/2$.

Figure 1b shows the output from Simulink® when the HA in (3) is implemented as Stateflow with $x(0) = 0.5$ radians and $\epsilon = 1e - 7$. This simulation used the well-known ODE45 solver, from Simulink®, with adaptive step size level crossing detection. The output is as expected. The steering wheel position increases at a rate of 1, until it reaches close to $\pi/2$ radians and then remains there forever. We cannot see any chattering, since the Stateflow has entered the third location in (3).

Next, we add uncertainty to the SMC as shown in (4). Equation (4) states that $x(t)$ evolves stochastically, where $W(t)$ is the standard Wiener process. The equivalent stochastic hybrid automaton (SHA) [2] is shown in (5). Notice that the so-called *drift* (dt) and *diffusion* ($dW(t)$) coefficients change in each of the three locations in (5). Equation (5) is implemented as a Stateflow chart with $x(0) = 0.5$ and $\epsilon = 1e - 7$ as before. The Stateflow chart is simulated using the fixed step Euler-Maruyama technique with a step size of $1e - 7$. The output is shown in Fig. 1c. The output is incorrect. Simulink has missed the level crossing (when $\pi/2 \leq x(t) \leq \pi/2 + \epsilon$) multiple times. Reducing the step size does not help.

To the best of our knowledge, no simulation tools such as Simulink® OpenModelica [17] support adaptive step size simulation of SHS. The difficulty lies in correctly implementing level crossing detection in the presence of Wiener processes. Level crossing detection in numerical simulation tools works using variants of binary search (e.g. bracketing in Simulink) as follows: ① from any given point in time $T \in \mathbb{R}^{\geq 0}$, take an integration step $\delta \in \mathbb{R}^{>0}$. ② Check if the level crossing guard shows a sign change. For the running example, check if $sgn(x(t) - \pi/2)$ changes sign. ③ If sign change is not detected, then continue to take the next integration step δ'. ④ If sign change is detected, then search for time $\tau \in [T, T + \delta]$, where the sign change happened. There are two major problems with this level crossing detection approach: ① using binary search between T and $T + \delta$ requires generating arbitrary Wiener increments, which depend upon the step size taken. Levy construction of Wiener increments for an arbitrary step size exists [13, 18]. However, this is a very inefficient construction. ② Furthermore, it is difficult to choose a value of ϵ, to enforce a non-chattering implementation, in the presence of Wiener processes. This is because it is impossible to distinguish between an integration step overshooting the level crossing bounded by arbitrary ϵ and a random Wiener increment. Hence, we need an adaptive step size simulation algorithm for SHS, which *converges to the level crossing without overshooting it.*

We propose just such an adaptive step size simulation algorithm. The correct output trace for the example in (4), generated from our simulation technique, is shown in Fig. 1d. As we can see, the level crossing is correctly detected at $x(0.2) = \pi/2$.

1.2 Contribution

Our main contribution in this work is an adaptive step size integration/simulation technique for stochastic hybrid systems. In particular:

1. Our integration/simulation technique *guarantees*, within floating point error bounds, convergence to the guard inducing the level crossing.
2. Does not require specification of an arbitrary ϵ to avoid chattering.
3. Our integration technique is efficient since it does not require construction of arbitrary Wiener increments.

The rest of the paper is arranged as follows: Section 2 gives the background information needed to read the rest of the paper. Section 3 describes the formal syntax and semantics of our construction of the SHS. Section 4 describes the main simulation algorithm. The algorithmic properties are analysed in Section 5. Section 6 then presents the experimental results. The presented work is placed in context with current state-of-the-art in Section 7. Finally, we conclude and provide the future work directions in Section 8.

2 Preliminaries

In this section, we give the preliminaries required to understand the rest of the paper.

2.1 Wiener process

A Wiener process $W(t)$ is a random variable that depends continuously on $t \in [0, T]$, where $T \in \mathbb{R}^{\geq 0}$ such that:

1. $W(0) = 0$, with probability 1.
2. $W(t) - W(s) \sim \sqrt{t-s} \times \mathcal{N}(0, 1)$, where $\mathcal{N}(0, 1)$ is a sample from a normal distribution with mean zero and variance one and $t > s$.
3. For $0 \leq s < t < u < v \leq T$, increments $W(t) - W(s)$ and $W(v) - W(u)$ are independent.

Dividing the interval $[0, T]$, into discrete steps of size δ, such that $\delta = T/N$, for some $N \in \mathbb{N}^{\geq 1}$, we can define a *discrete* Wiener process as in Definition 1.

Definition 1 For some s, such that $0 < s \leq T$, and $s = \delta \times j$, where $j \in \{1, \ldots, N\}$, a discrete Wiener process $W[s]$ is given in (6).

$$\begin{aligned}
W[1] &= W[0] + \sqrt{\delta} \times \mathcal{N}(0, 1), \text{ from (2) above} \\
\therefore W[s] &= W[0] + \sum_{i=1}^{j} (\sqrt{\delta} \times \mathcal{N}_i(0, 1)), \text{ from (3) above} \\
\therefore W[s] &= \sqrt{\delta} \times \sum_{i=1}^{j} (\mathcal{N}_i(0, 1)), \text{ from (1) above}
\end{aligned} \tag{6}$$

Given $0 < s < T$ and $s < s+\Delta \leq T$, where $s = M \times \delta$ and $s+\Delta = (M+R) \times \delta$, $M, R \in \mathbb{N}^{\geq 1}$. We can define $W[s + \Delta] - W[s]$ as shown in (7).

$$\begin{aligned}
&\text{from (6) we have} \\
W[s + \Delta] - W[s] &= (\sqrt{\delta} \sum_{i=1}^{M+R} \mathcal{N}_i(0, 1)) - (\sqrt{\delta} \sum_{i=1}^{M} \mathcal{N}_i(0, 1)) \\
\therefore W[s + \Delta] - W[s] &= \sqrt{\delta} \times (\sum_{i=M}^{M+R} \mathcal{N}_i(0, 1)), \\
\therefore W[s + \Delta] - W[s] &= \sqrt{\delta} \times (\sum_{i=1}^{R} \mathcal{N}_i(0, 1))
\end{aligned} \tag{7}$$

2.2 Fixed step Euler-Maruyama solution to SDE

A scalar, autonomous SDE is shown in (8) and its solution in (9), where $f(x(t))$—called the drift coefficient, is the slope of the continuous variable $x(t)$ changing with time t. Slope $g(x(t))$—called the diffusion coefficient, on the other hand shows the change in the continuous variable with the change in the one-dimensional Wiener process $W(t)$. If $g(x(t)) = 0$, then (8) is an ODE.

$$dx(t) = f(x(t))\,dt + g(x(t))\,dW(t) \tag{8}$$

$$\therefore x(t) = x(0) + \int_0^t f(x(s))\,ds + \int_0^t g(x(s))\,dW(s), \forall t \in [0, T] \tag{9}$$

The numerical Euler-Maruyama solution [11] to (9), at time T, with $x(0)$ as the initial value, is given in (10). In (10), Δ is the fixed step size such that $T/Y = \Delta$, $Y \in \mathbb{N}^{\geq 1}$. If $g(x[j-1])$ is 0 then (10) devolves to standard forward Euler solution of an ODE.

$$x[T] = x(0) + \sum_{j=1}^{Y}(f(x[j-1])\Delta + g(x[j-1])(W[s+\Delta] - W[s]))$$

from (7) we have for $\Delta = \delta \times R$

$$\therefore x[T] = x(0) + \sum_{j=1}^{Y}(f(x[j-1])\Delta + g(x[j-1])(\sqrt{\delta}\sum_{i=1}^{R}\mathcal{N}_i(0,1))) \tag{10}$$

3 Formal syntax and semantics

Stochastic hybrid systems (SHSs) can capture stochastic behaviour in a plethora of different ways [2]. The most common option is using piecewise stochastic differential equations (SDEs). However, Poisson processes are another technique to capture stochasticity. In this section, we formally describe the syntax, well-formed criteria, and semantics of the subset of SHS that we simulate.

3.1 Syntax

This paper describes the adaptive step size simulation algorithm for SHS expressed as SDEs with discontinuous drift and diffusion coefficients as defined in Definition 2.

Definition 2 A stochastic hybrid system (SHS) is defined in (11).

$$d\mathbf{x}(t) = \sum_{q=1}^{L} \mathbb{1}_{\theta_q}((A_q\mathbf{x}(t) + B_q)\,dt + B'_q\,dW(t)) \tag{11}$$

In (11), $\mathbf{x}(t) \in \mathbb{R}^n$ is a vector of continuous variables, $A_q \in \mathbb{R}^{n\times n}$ is a matrix, and $B_q, B'_q \in \mathbb{R}^n$ are vectors. $\theta_q \subset \mathbb{R}^n$ indicates the invariants for each location $q \in \{1, \dots, L\}$, and $\mathbb{1}$ is the unit vector. If B'_q is a zero vector, then (11) becomes a hybrid automaton. Furthermore, SDEs in *each* location are linear and time-invariant, and consequently locally Lipschitz continuous.

Remark 1 SHS defined as (11) is well formed if:

1. $\bigcup_{q=1}^{L} \theta_q = \mathbb{R}^n$ and $\bigcap_{q=1}^{L} \theta_q = \emptyset$. Informally, the location invariants cover all scenarios and are disjoint for each continuous variable $x(t)$ in vector $\mathbf{x}(t)$.
2. Given a location invariant of the form $l \preceq \theta_q \preceq u$ for some location q, and for some variable $x(t)$ in vector $\mathbf{x}(t)$. With $\preceq \in \{<, \leq\}$, and $l, u \in \mathbb{R} \setminus \{-\infty, +\infty\}$. Then, the drift coefficients $((A_q\mathbf{x}(t) + B_q))$ of the SDE in location q evolve the continuous variable x towards l or u.

$$
\begin{aligned}
dx(t) = {} & \mathbb{1}_{(-\infty,\pi/2)}(([0][x(t)] + [1])\,dt + [-1]\,dW(t)) \\
& + \mathbb{1}_{(\pi/2,+\infty)}(([0][x(t)] + [-1])\,dt + [1]\,dW(t)) \\
& + \mathbb{1}_{[\pi/2,\pi/2]}(([0][x(t)] + [0])\,dt + [0]\,dW(t))
\end{aligned} \tag{12}
$$

The stochastic sliding mode control (SMC) running example in (4) can be expressed within our definition of SHS (Definition 2) as shown in (12). There are three locations $q \in \{1, 2, 3\}$ in (12). The invariants on the three locations are $(-\infty, \pi/2)$, $(\pi/2, +\infty)$, and $[\pi/2, \pi/2]$, respectively. The matrix $A_q = [0]$ in all three locations. For the three locations, we also have $B_1 = [1]$, $B_1' = [-1]$, $B_2 = [-1]$, $B_2' = [1]$, and $B_3 = B_3' = [0]$, respectively. Equation (12) satisfies the well-formed criteria. The union of location invariants covers the entire real number line and the intersection of the location invariants is empty, thereby satisfying the first criteria. The second criteria is satisfied, because in all three locations, the SDEs evolve $x(t)$ to $\pi/2$. As long as $x(t) \in (-\infty, \pi/2)$, q remains as one, and the system trajectory moves towards $\pi/2$. As soon as $x(t) = \pi/2$, q becomes three. Same when $x(t) \in (\pi/2, +\infty)$ and q remains as two during this period. Notice that the behaviour is the same as in (5). However, we do not specify the arbitrary ϵ to avoid chattering.

3.2 Semantics

Formally, a well-formed SHS in (11) follows the same stochastic execution as in [7]. The execution semantics of our SHS is given in Definition 3.

Definition 3 A well-formed SHS expressed as (11) has a stochastic execution $(\mathbf{x}(t), q(t)) \in (\mathbb{R}^n, \mathbb{N})$, iff, there exists a sequence of stop times $\tau_0 = 0 \leq \tau_1 \leq \tau_2 \ldots, \tau_m, \ldots$, where $\tau_m \in \mathbb{R}^{\geq 0}$ such that:

– There exists exactly, one $q(\tau_0) \in \{1, \ldots, L\}$, such that $\mathbf{x}(\tau_0) \in \theta_{q(\tau_0)}$
– In each interval $t \in [\tau_n, \tau_{n+1})$, $q(t) = q(\tau_n)$ remains constant and $\mathbf{x}(t)$ is the solution to the SDE in (13).

$$
d\mathbf{x}(t) = (A_{q(\tau_n)}\mathbf{x}(t) + B_{q(\tau_n)})\,dt + B'_{q(\tau_n)}\,dW(t) \tag{13}
$$

– $\tau_{n+1} = \inf\{t \geq \tau_n | \mathbf{x}(t) \notin \theta_{q(\tau_n)}\}$. Informally, a *level crossing* is made from location $q(\tau_n)$ to $q(\tau_{n+1})$ as soon as the location invariant is violated. At the level crossing stop time τ_{n+1}, the SDE stops evolving, i.e. drift and diffusion coefficients can be considered to be zero at instant τ_{n+1}.
– $\mathbf{x}(\tau_{n+1}) = \mathbf{x}(\tau_{n+1}^-)$, where $\mathbf{x}(\tau_{n+1}^-)$ denotes the left limit: $\lim_{t \nearrow \tau_{n+1}} \mathbf{x}(t)$. Informally, the value of the continuous variables are carried over into the new location.

The output trace generated from (12) with initial value $x(0) = 0.5$ until $x(t) = \pi/2$ is shown in Fig. 1d.

4 Simulation algorithm

In this section, we describe the proposed SHS simulation algorithm in a top-down manner. We first describe the overall algorithm for simulating a well-formed SHS (Definition 2) and then describe the integration step size computation algorithm. Before explaining any of the algorithms, we set up a few cursory variable definitions with their constraints.

4.1 Definitions and constraints for each continuous variable in the system

For every $x(t)$ in vector $\mathbf{x}(t)$, let $x[T]$, denote the value of the continuous variable, at simulation time $T \in [0, T_{sim}]$ where T_{sim} is the user specified simulation end time. Moreover, $\tau_m \leq T_{sim}$ is some stop time (Definition 3), such that $x[\tau_m] \notin \theta_{q(\tau_{m-1})}$, and $x[T] \in \theta_{q(\tau_{m-1})}$, where $x[\tau_m]$ is computed using Euler-Maruyama numerical solution. Finally, let $R \in \mathbb{N}^{\geq 1}$ be a user-specified constant, then we have the following:

Definition 4 Level crossing step: $\Delta^x_{\tau_m} = \tau_m – T$ and $\Delta^x_{\tau_m} = \delta^x_{\tau_m} \times R$, where $\delta^x_{\tau_m} \in \mathbb{R}^{>0}$.

Definition 5 Error bounded step: $\Delta^x_{\epsilon} = \tau' - T$ for some $T < \tau' < \tau_m$ and $\Delta^x_{\epsilon} = \delta^x_{\epsilon} \times R$, where $\delta^x_{\epsilon} \in \mathbb{R}^{>0}$.

Definition 6 Random sample vector: $dWt \in \mathbb{R}^R$ is a vector of R samples from $\mathcal{N}(0, 1)$.

Definition 7 Change satisfying level crossing: given $l \preceq \theta_{q(\tau_{m-1})} \preceq u$, $l, u \in \mathbb{R} \setminus \{-\infty, +\infty\}$ and $\preceq \in \{<, \leq\}$. We have $\gamma_l = |l - x[T]|$, $\gamma_u = |u - x[T]|$ and $\gamma \in \{\gamma_l, \gamma_u\}$.

4.2 Adaptive step size simulation of a stochastic hybrid system

Figure 2 gives the **key idea** of our adaptive step size simulation technique. In Fig. 2, we have a two continuous variable system with their initial values: $\mathbf{x}(0) = [x1(0), x2(0)]$. The x-axis shows progression of time and the y-axis shows the value of the continuous variables. Values τ_1, τ_2, and τ_3 in Fig. 2 show the stop times as defined in Definition 3.

The algorithm, in the very first step, **Sim step-0**, computes integration steps $\Delta^{x1}_{\tau_1}$ and $\Delta^{x2}_{\tau_1}$, such that numerical solutions $x1[T + \Delta^{x1}_{\tau_1}]$ and $x2[T + \Delta^{x2}_{\tau_1}]$ in a single step violate their individual location invariants. During **Sim step-0**, $T = 0$ is the current simulation time. In Fig. 2, variable $x2(t)$ violates the location invariant before $x1(t)$. Next, the algorithm, in **Sim step-0**, also computes steps Δ^{x1}_{ϵ} and Δ^{x2}_{ϵ} such that the outputs $x1[T + \Delta^{x1}_{\epsilon}]$ and $x2[T + \Delta^{x2}_{\epsilon}]$ are within some user-specified error bound

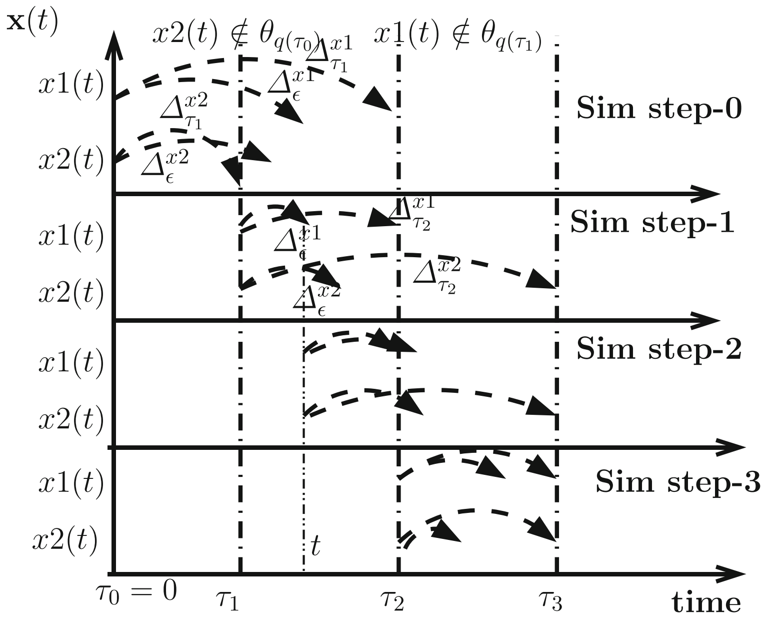


Fig. 2 Key idea of the simulation algorithm

$\epsilon > \mathbb{R}^{>0}$ from the exact solutions. Finally, the algorithm **commits** the smallest from amongst these computed step sizes as the integration step. Integration is carried out using the Euler-Maruyama numerical solution. In the case of Fig. 2, the minimum is $\Delta^{x2}_{\tau_1}$ in **Sim step-0**. Hence, in **Sim step-1**, all computation starts from τ_1. Again the algorithm computes the four possible time steps in **Sim step-1**. In this simulation step, the smallest step size is Δ^{x1}_{ϵ} and hence, in **Sim step-2** all computation starts from some time between τ_1 and τ_2. This process continues until the end of user-specified simulation time.

Algorithm 1 shows the pseudo-code implementing the key idea. The algorithm takes as input the well-formed SHS from Definition 2, e.g. the SHS in (12) and gives the output trace—the values of continuous variables back. First, the timer T is initialised to zero (line 1) and the initial values of the continuous variables are stored (line 2). For the running example, in (12), with $x(0) = 0.5$ this entails storing the initial value 0.5. Next, lines 3–24 run the simulation until user-specified end time T_{sim}. On line 5, the current location is obtained. For the running example, in (12), for the very first time, q is one, because $x(0) = 0.5 \in (-\infty, \pi/2)$. Next, the algorithm gets the potential step sizes to be committed, for each variable in $\mathbf{x}(t)$ (lines 8–20). The running example, in (12), only has a single variable and hence, we compute two possible time steps Δ^x_{ϵ} and $\Delta^x_{\tau_1}$, such that $x[T + \Delta^x_{\epsilon}]$ is bounded by the user specified error bound ϵ and $x[T + \Delta^x_{\tau_1}]$ is equal to $\pi/2$, thereby violating its location invariant, respectively. Finally, lines 22 and 23 commit the integration step via Euler-Maruyama numerical solution, using the minimum from amongst these time steps in Δs (line 21) and increment the timer, respectively.

Algorithm 1 Pseudo-code describing the simulation of well-formed SHS.

```
   Input  : A well-formed SHS $\mathcal{S}$
   Input  : Initial values $\mathbf{x}(0)$
   Output : Output Trace $\mathcal{X}$
1  $T \leftarrow 0$ ; // Global clock initialisation
2  $\mathbf{x}[T] \leftarrow \mathbf{x}(0)$ ; // Set current values
3  while $T \leq T_{sim}$ do
4      $\mathcal{X} \leftarrow$ Record($T, \mathbf{x}[T]$) ;            // record system output
       /* Get current location $q$, such that $\mathbf{x}[T]$ satisfies
          $q'$s invariant                                          */
5      $q \leftarrow$ Getloc($\mathcal{S}, \mathbf{x}$[T]) ;
6      $\Delta s \leftarrow \emptyset$ ; // Initialise a set of time steps
7      $dWt \leftarrow$ randn($R$) ; // Create $R$ samples from $\mathcal{N}(0, 1)$
       /* For each continuous variable                               */
8      foreach $x \in \mathbf{x}$ do
           /* Get its invariant set in location $q$, $\theta^x$ is a
              tuple with left and right bounds of the
              invariant for variable $x$                             */
9          $\theta^x \leftarrow |j| j \in \theta[q][x] \wedge j \neq \pm\infty|$ ;
10         $\Delta^x \leftarrow (\infty, \infty)$ ; /* Initialise steps for left and
           right invariant bounds for variable $x$                   */
11         foreach $j \in \theta^x$ do
               /* Get the difference between current value
                  of variable and the invariant bound               */
12             $\gamma \leftarrow |j - \mathbf{x}[T][x]|$ ;
13             if $\gamma \neq 0$ then
                   /* Get the integration step                      */
14                 $\Delta^x[j] \leftarrow$ step($\mathbf{x}$[T], x, q, A[q], B[q], B'[q], $\gamma$, dWt) ;
15             else
16                 $\Delta^x[j] \leftarrow 0$
17             end
18         end
19         $\Delta s \leftarrow \Delta s \cup \Delta^x$ ; /* Union the step for each variable
           into the overall set                                      */
20     end
21     $\Delta \leftarrow \min(\Delta s)$ ; // Final step is the minimum of them
       all
22     $x[T+\Delta] = x[T]+(A[q]\bullet x[T]+B[q]) \times \Delta_s + B'[q] \times \sqrt{\delta} \times (\sum_{i=1}^{R} dWt[i])$
       ; /* Update the continuous variables using
       Euler-Maruyama numerical solution, $\Delta = R \times \delta$       */
23     $T \leftarrow T + \Delta$ ; // Increment time
24 end
25 return $\mathcal{X}$
```

The `step` function call on line 14 gives back the minimum from amongst Δ_τ^x and Δ_ϵ^x for all $x(t)$ in vector $\mathbf{x}(t)$. We describe this function in Algorithm 2.

Algorithm 2 Integration step size computation, considering level crossing and error bounds.

```
   Input   : Current values $\mathbf{x}[T]$
   Input   : Continuous variable $x$
   Input   : Current Location $q$
   Input   : System matrix in $q$, $A$
   Input   : Constant vector in $q$, $B$
   Input   : Diffusion constant vector in $q$, $B'$
   Input   : Change satisfying level crossing $\gamma$
   Input   : Random samples $dWt$
   Output  : Integration step size $\Delta$
1  $R \leftarrow \text{len}(dWt)$ ; // Store the number of samples in $R$
2  while $True$ do
       /* Store the current value of continuous variables
          temporarily                                              */
3      $xtemp \leftarrow \mathbf{x}[T]$ ;
4      $xtemph \leftarrow \mathbf{x}[T]$ ;
       /* Compute the drift and diffusion coefficients for $x$   */
5      $\mu \leftarrow (A \bullet \mathbf{x}[T] + B)[x]$ ; // $\bullet$ is dot product
6      $\sigma \leftarrow B'[x]$ ;
       /* Call to **Algorithm** 3                                  */
7      $\Delta \leftarrow \text{compute}(\mu, \sigma, \gamma, dWt)$ ;
       /* Compute the total area under Wiener process             */
8      $Winc \leftarrow \sqrt{\Delta/R} \times \sum_{j=1}^{R}(dWt[j])$ ;
       /* Integration result of taking the step $T$ to $T + \Delta$ for
          all variables                                            */
9      $Fxts \leftarrow (A \bullet xtemp + B) \times \Delta$ ;
10     $Gxts \leftarrow B' \times Winc$ ;
11     $xtemp \leftarrow xtemp + Fxts + Gxts$ ;
       /* Integration result of taking the step $T$ to $T + \Delta/2$ for
          all variables                                            */
12     $Winc' \leftarrow \sqrt{\Delta/R} \times \sum_{j=1}^{R/2}(dWt[j])$ ;
13     $Fxts \leftarrow (A \bullet xtemph + B) \times \Delta/2$ ;
14     $Gxts \leftarrow B' \times Winc'$ ;
15     $xtemp \leftarrow xtemp + Fxts + Gxts$ ;
       /* Integration result of taking the step $T + \Delta/2$ to $T + \Delta$
          for all variables                                        */
16     $Winc' \leftarrow \sqrt{\Delta/R} \times \sum_{j=R/2+1}^{R}(dWt[j])$ ;
17     $Fxts \leftarrow (A \bullet xtemph + B) \times \Delta/2$ ;
18     $Gxts \leftarrow B' \times Winc'$ ;
19     $xtemp \leftarrow xtemp + Fxts + Gxts$ ;
       /* Check the error between taking single step to $\Delta$ vs.
          two half steps to $\Delta$                               */
20     if $error(xtemp, xtemph) \geq \epsilon$ then
21         $\gamma \leftarrow \gamma/2$ ; /* If error is beyond user specified $\epsilon$ then
           half $\gamma$                                           */
22     else
23         break
24     end
25 end
26 return $\Delta$
```

4.3 Computing the integration time steps

Algorithm 2 takes the current value of all the continuous variables in the system: $\mathbf{x}[T]$, the current continuous variable, $x(t)$, whose integration step needs to be computed, the current location q, the system matrices in the current location q: A, B, B', the change satisfying the level crossing γ (Definition 7), and the dWt random sample vector as inputs and returns the minimum from amongst the time steps $\Delta^x_{\tau_m}$, Δ^x_ϵ.

The **key idea** is shown in Fig. 3. Algorithm 2 first computes the time step $\Delta^x_{\tau_m}$, such that the numerical solution violates the location invariant, and consequently the level crossing is satisfied (line 7). Consider the running example in (12), with $T = 0$ and $x[T] = 0.5$. Then, the call to Algorithm 3 on line 7 will return a time step $\Delta^x_{\tau_m}$, such that the location invariant $\theta_{q(\tau_m)} \notin (-\infty, \pi/2)$. In fact, $x[T + \Delta^x_{\tau_m}] = \pi/2$ meeting the level crossing.

Next, the algorithm gets the value for *all* continuous variables ($\mathbf{x}[T + \Delta^x_{\tau_m}]$) for the time step $\Delta^x_{\tau_m}$ by applying Euler-Maruyama numerical integration (lines 8–11). Algorithm 2, then takes two half steps: first from T to $\Delta^x_{\tau_m}/2$ computing $\mathbf{x}'[T+\Delta^x_{\tau_m}/2]$ and then from $T+\Delta^x_{\tau_m}/2$ to $T+\Delta^x_{\tau_m}$ computing the vector $\mathbf{x}'[T+\Delta^x_{\tau_m}]$ (lines 12—19). Finally, the **error** function (lines 20–24 and (14)) checks if the difference between the two vectors, $\mathbf{x}[T + \Delta^x_{\tau_m}]$ and $\mathbf{x}'[T + \Delta^x_{\tau_m}]$, is bounded by some user-specified $\epsilon \in \mathbb{R}^{>0}$. If the difference is bounded, then the step $\Delta^x_{\tau_m}$ is accepted and returned (line 26). Else, γ is halved (line 21) and the process is iterated. The next iteration will give a new time step $\Delta^x_\epsilon < \Delta^x_{\tau_m}$, from Algorithm 3, which satisfies Definition 5. Algorithm 2 keeps on iterating, halving γ, until the error bound is satisfied.

$$error(\mathbf{x}[T + \Delta^x], \mathbf{x}'[T + \Delta^x]) = \sum_{\forall x \in \mathbf{x}} |(x'[T + \Delta^x] - x[T + \Delta^x])/x'[T + \Delta^x]|,\ \Delta^x \in \{\Delta^x_{\tau_m}, \Delta^x_\epsilon\} \tag{14}$$

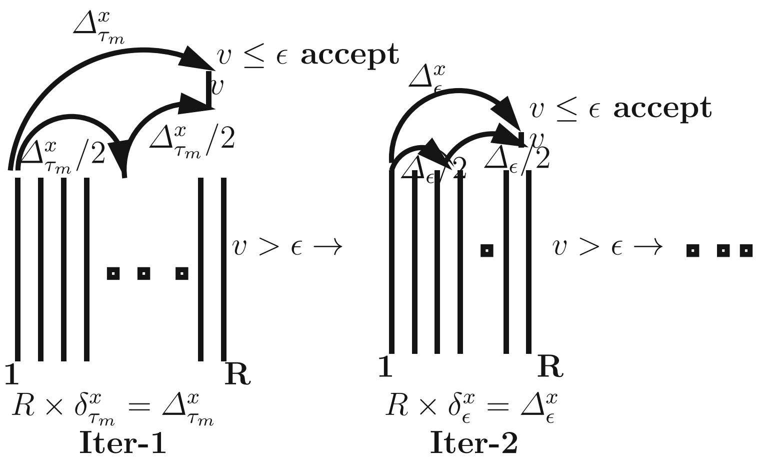


Fig. 3 Computing the integration step size for each variable

Algorithm 3 Algorithm to detect the step size that converges to the level crossing.

```
   Input  : Drift coefficient $\mu$
   Input  : Diffusion coefficient $\sigma$
   Input  : Change $\gamma$
   Input  : Random samples $dWt$
   Output : Step size $\Delta$
 1 $R \leftarrow \text{len}(dWt)$ ;
 2 $\Gamma \leftarrow \sigma \times \sum_{j=1}^{R}(dWt[j])$ ;
 3 $a \leftarrow R \times \mu^2$ ;
 4 $b \leftarrow (2 \times \mu \times \gamma \times R) - (\Gamma^2)$ ;
 5 $c \leftarrow R \times \gamma^2$ ;
   /* Root of quadratic equation $ax^2 + bx + c$ */
 6 $\Delta 1 \leftarrow \text{root}(a, b, c)$ ;
 7 $b \leftarrow (2 \times \mu \times \gamma \times R) + (\Gamma^2)$ ;
   /* Root of quadratic equation $ax^2 - bx + c$ */
 8 $\Delta 2 \leftarrow root(a, b, c)$ ;
 9 return $\min(\Delta 1, \Delta 2)$
```

In the case of the running example, (12), with $T = 0$ and $x[T] = 0.5$, Algorithm 2 calls Algorithm 3, the very first time with $\gamma = |\pi/2 - 0.5|$, following Definition 7. The returned step size $\Delta^x_{\tau_1}$ will be accepted if the value returned by (14) is bounded by some user-specified ϵ. Else, Algorithm 3 will be called recursively, reducing γ by half until the error is bounded and obtaining the step size Δ^x_ϵ.

From Definitions 4 and 5, $\Delta^x_{\tau_m}$ (respectively Δ^x_ϵ) consists of constant R steps of size $\delta^x_{\tau_m}$ (respectively δ^x_ϵ). Solution $\mathbf{x}'[T + \Delta^x_{\tau_m}]$ (respectively $\mathbf{x}'[T + \Delta^x_\epsilon]$), obtained by taking two half steps, is considered the reference solution, which bounds the integration time step $\Delta^x_{\tau_m}$ (respectively Δ^x_ϵ). For a large R, the reference solution can be computed, in the worst case, by taking R steps of size $\delta^x_{\tau_m}$ (respectively δ^x_ϵ). However, this can substantially increase the algorithm runtime. Experimental results show that taking two half steps to compute the reference solution is enough to bound the integration step, so that the final output is close to the reference solution taking R individual steps of size $\delta^x_{\tau_m}$ (respectively δ^x_ϵ).

Algorithm 3 forms the core of our level crossing detection algorithm. Let $\Delta^x \in \{\Delta^x_{\tau_m}, \Delta^x_\epsilon\}$ be a time step returned back from Algorithm 3, depending upon the γ value being input from Algorithm 2. Algorithm 3 works for a single continuous variable and computes a single integration step size. Hence, (10) can be rewritten as (15). In (15), μ and σ are the constant drift and diffusion coefficients, respectively, at simulation time T. Moreover, R and $\delta^x \in \{\delta^x_{\tau_m}, \delta^x_\epsilon\}$ are as defined in Section 4.1.

$$x[T + \Delta^x] = x[T] + f(x[T])\Delta^x + g(x[T])(\sqrt{\delta^x}\sum_{i=1}^{R}\mathcal{N}_i(0,1))$$

$$\therefore x[T + \Delta^x] = x[T] + \mu \times \Delta^x + \sigma \times (\sqrt{\delta^x}\sum_{i=1}^{R}\mathcal{N}_i(0,1))$$

$$\therefore x[T+\Delta^x] = x[T] + \mu \times \Delta^x + \sqrt{\delta^x} \times (\sigma \sum_{i=1}^{R} \mathcal{N}_i(0,1))$$
$$\therefore x[T+\Delta^x] = x[T] + \mu \times \Delta^x + \sqrt{\delta^x} \times \Gamma$$
$$\therefore x[T+\Delta^x] - x[T] = \mu \times \Delta^x + \sqrt{\Delta^x/R} \times \Gamma \tag{15}$$

However, since $|x[T+\Delta^x] - x[T]|$ is equal to γ, we have (16) and (17).

$$\mu \times \Delta^x + \sqrt{\Delta^x/R} \times \Gamma = \pm\gamma$$
$$\therefore \mu \times \Delta^x + \gamma = -\sqrt{\Delta^x/R} \times \Gamma \tag{16}$$
$$\mu \times \Delta^x - \gamma = -\sqrt{\Delta^x/R} \times \Gamma \tag{17}$$

Expanding (16) and (17), we get quadratics in (18) and (19) whose roots give us values for Δ^x. The minimum of these values is the final value Δ^x that is returned. The two (18) and (19) handle the separate case of the continuous variable with positive and negative drift coefficients. Algorithm 3 calculates the roots of these two quadratics, within some floating point error bound, on lines 3–8, given the required inputs. It returns the minimum amongst these roots on line 9.

$$R\mu^2(\Delta^x)^2 + (2\mu\gamma R - \Gamma^2)\Delta^x + R\gamma^2 = 0 \tag{18}$$
$$R\mu^2(\Delta^x)^2 - (2\mu\gamma R + \Gamma^2)\Delta^x + R\gamma^2 = 0 \tag{19}$$

5 Analysis of the algorithm

This section analyses the following properties: ① existence of a real positive integration step guaranteeing that the algorithm always makes progress—called simulation progress—and ② the simulation algorithm should converge to the level crossing without overshooting it. The strong convergence rate, of the proposed algorithm, is the same as standard Euler-Maruyama fixed step technique—$O(\Delta^{1/2})$ [14, 19], where Δ is the step size, because we use Euler-Maruyama numerical solution to solve the SDEs at each integration step.

5.1 Simulation progress—existence of a real positive integration step

Algorithm 3 computes the roots of two quadratic equations. These roots are the possible simulation/integration steps. A quadratic equation, with real coefficients, might have two complex roots, from the fundamental theorem of algebra, in which case simulation progress will halt. Hence, we need to guarantee that we get real positive roots when solving the quadratic equations. This guarantee is given by Theorem 1. Lemma 1 is the supporting lemma for Theorem 1. It describes the conditions for a real positive root for the quadratics ((18) and (19)) used in Algorithm 3.

Lemma 1 *There exists a* $\gamma \in \mathbb{R}^{>0}$ *such that* $R\mu^2(\Delta^x)^2 + (2\mu\gamma R - \Gamma^2)\Delta^x + R\gamma^2 = 0$ *or* $R\mu^2(\Delta^x)^2 - (2\mu\gamma R + \Gamma^2)\Delta^x + R\gamma^2 = 0$ *has at least one real positive root.*

Proof This lemma can be proven by first showing that there always exists a $\gamma > \mathbb{R}^{>0}$ such that the discriminant is always non-negative. Considering the discriminants of the two quadratics, we have (20) and (21).

$$\gamma \leq \frac{\Gamma^2}{4\mu R} \tag{20}$$

$$\gamma \geq \frac{-\Gamma^2}{4\mu R} \tag{21}$$

The two equations are real positive for $\mu > 0$ and $\mu < 0$, accounting for the positive and negative drifts, respectively. Next, given the non-negative discriminant, we can show that there is at least one real positive root for the quadratics. This can be done by considering the general formula for solutions of quadratic equations, and consider the two cases of the discriminant being equal to zero and greater than zero, respectively. $\square$

Theorem 1 *Algorithms 1, 2, and 3 together always get a real positive root, i.e.* $\Delta^x \in \mathbb{R}^{>0}$ *for* $R\mu^2(\Delta^x)^2 + (2\mu\gamma R - \Gamma^2)\Delta^x + R\gamma^2 = 0$ *or* $R\mu^2(\Delta^x)^2 - (2\mu\gamma R + \Gamma^2)\Delta^x + R\gamma^2 = 0$.

Proof $\gamma \in \{\gamma_l, \gamma_u\}$ where $\gamma_l, \gamma_u \in \mathbb{R}^{\geq 0}$ as defined in Definition 7. Algorithm 2 keeps on halving γ until some error bound is satisfied. So, we need to show that there will eventually be a γ, which satisfies the conditions in (20) or (21) and the real positive root will follow from Lemma 1. Let $\gamma' \in \mathbb{R}^{>0}$ be the value satisfying Lemma 1. We consider different cases for γ and μ.

1. $\gamma = 0$ (equivalently $\gamma_l = 0$ or $\gamma_u = 0$), this means we are already at the level crossing and $\mu = 0$.
2. $\gamma = \gamma'$ (equivalently $\gamma_l = \gamma'$ or $\gamma_u = \gamma'$), $\mu > 0$ or $\mu < 0$, either of (20) or (21) are satisfied and we get a real positive root.
3. $\gamma < \gamma'$ (equivalently $\gamma_l < \gamma'$ or $\gamma_u < \gamma'$) and $\mu > 0$, in this case (20) will be satisfied so we will get a real positive root.
4. $\gamma > \gamma'$ (equivalently $\gamma_l > \gamma'$ or $\gamma_u > \gamma'$) and $\mu < 0$, in this case (21) will be satisfied, so we will get a real positive root.
5. $\gamma < \gamma'$ and $\mu < 0$. Here we have three cases:
   (a) $\gamma_l < \gamma'$, but $\gamma_u > \gamma'$, $\mu < 0$. In this case, (21) will be satisfied by evolving the continuous variable towards the right bound of the location invariant.
   (b) $\gamma_u < \gamma'$, but $\gamma_l > \gamma'$, $\mu < 0$. In this case, (21) will be satisfied by evolving the continuous variable towards the left bound of the location invariant.

(c) $\gamma_u < \gamma'$, and $\gamma_l < \gamma'$, $\mu < 0$. In this case since $\mu < 0$, it implies (21), (19), and (17) need to be satisfied. Thus, we are considering the case of $|\gamma| = \gamma$. From Definition 7, then lower bound of current location invariant $l > x[T]$, which cannot happen. In the case of the upper bound of the current location invariant, $u > x[T]$. However, since the level crossing u is greater than current value $x[T]$ and the continuous variable x is evolving away from the level crossing, because $\mu < 0$. This is **not** a well-formed SHS according to Remark 1.

6. Finally, case $\gamma > \gamma'$ and $\mu > 0$ is the dual of the one above. □

5.2 Convergence to level crossing

In Theorem 2, we show that the proposed numerical simulation technique forces the numerical solutions to converge to the level crossing, within floating point error bound, without overshooting it.

Lemma 2 *Let Δ^x be the real positive root of $R\mu^2(\Delta^x)^2+(2\mu\gamma R-\Gamma^2)\Delta^x+R\gamma^2 = 0$ or $R\mu^2(\Delta^x)^2 - (2\mu\gamma R + \Gamma^2)\Delta^x + R\gamma^2 = 0$ for some value γ. Then, $\forall\gamma' < \gamma$, root $\Delta' < \Delta^x$.*

Proof Considering $R\mu^2(\Delta^x)^2 + (2\mu\gamma R - \Gamma^2)\Delta^x + R\gamma^2 = 0$. Let $a = R\mu^2$, $b = (2\mu\gamma R - \Gamma^2)$, and $c = R\gamma^2$. Then from Lemma 1, we have $\Delta^x = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$ with possibly the discriminant of zero. The value of Δ^x only varies with γ, since μ, R, and Γ are constants. Moreover, a is independent of γ. Hence, reducing γ reduces Δ^x. Same for the other quadratic equation. □

Theorem 2 *For a given SHS $\mathcal{S}$, defined in (11), there exists a stop time $\tau_m \in \mathbb{R}^{>0}$, such that $\forall T \in [\tau_{m-1}, \tau_m)$, $\mathbf{x}[T] \in \theta_{q(\tau_{m-1})}$. Then, $\int_T^{T+\Delta}(A_{q(\tau_{m-1})} \bullet \mathbf{x}(\tau) + B_{q(\tau_{m-1})})d\tau$ $+ \int_T^{T+\Delta} B'_{q(\tau_{m-1})}dW(\tau) \notin \theta_{q(\tau_{m-1})}$, i.e. our simulation technique can take an integration step Δ, such that the level crossing is satisfied and $T + \Delta = \tau_m$.*

Proof From Remark 1, drift coefficients evolve the continuous variables towards the level crossing. Hence, there exists a step Δ such that level crossing is satisfied. $\Delta = \min\{\{\Delta_{\tau_m}^{x_1}, \ldots \Delta_{\tau_m}^{x_n}\} \cup \{\Delta_\epsilon^{x_1}, \ldots, \Delta_\epsilon^{x_n}\}\}$, where for any $x_n(t) \in \mathbf{x}(t)$, $\Delta_{\tau_m}^{x_n}$ and $\Delta_\epsilon^{x_n}$ are the roots of (18) or (19). Base step: if $\Delta = \Delta_{\tau_m}^{x_n}$, then by (16) or (17), and Definition 7, level crossing is satisfied and $T + \Delta_{\tau_m}^{x_n} = \tau_m$. Recursive step: $\Delta = \Delta_\epsilon^{x_n}$, then by Lemma 2 and (Algorithm 2, line 21), we have $\Delta_\epsilon^{x_n} < \Delta_{\tau_m}^{x_n}$ and the level crossing is *not* satisfied at time $T+\Delta_\epsilon^{x_n} \in [\tau_{m-1}, \tau_m)$. By setting $T = T+\Delta_\epsilon^{x_n}$, we need to show that there exists a step Δ, $\int_T^{T+\Delta}(A_{q(\tau_{m-1})} \bullet \mathbf{x}(\tau) + B_{q(\tau_{m-1})})d\tau + \int_T^{T+\Delta} B'_{q(\tau_{m-1})}dW(\tau) \notin \theta_{q(\tau_{m-1})}$. However, this is the statement of the theorem itself. □

6 Experimental results

In this section, we simulate a number of different examples to show the efficacy of the proposed simulation algorithm. Section 6.1 describes the benchmarks used. Section 6.2 describes the experimental setup and the comparison metrics. Finally, Section 6.3 presents the results along with the output traces.

6.1 Benchmark description

We select four different benchmarks from embedded control.

1. Steering wheel (SW): This is the running example as defined in (12). It has three locations and a single continuous variable $x(t)$. The objective is to maintain the steering wheel position at the stable point $\pi/2$.
2. Twisting controller (TC): A TC is a higher order sliding mode control (SMC) technique [20], which is a very popular control technique used in robotics [21], drug delivery systems [22], etc. The stochastic version of the TC is given in (22) and (23), where $b > s > 0$ are constants.

$$dy(t) = (\frac{-(b+s)}{2} sgn(x(t)) - \frac{b-s}{2} sgn(y(t)))dt + dW(t) \tag{22}$$
$$dx(t) = y(t)dt + dW(t) \tag{23}$$

   Rewriting (22) and (23) into the form defined in (11) gives 9 locations, with two continuous variables $x(t)$ and $y(t)$. Note that derivative of $x(t)$ is equal to $y(t)$ ((23)), while the derivative of $y(t)$ changes signs ((22)). Hence, overall the second derivative of $x(t)$ changes signs. Thus, the name higher order sliding mode control. The objective is to *twist* to the stable point (0, 0).
3. Steering wheel with continuous diffusion (SWD): This is a modified version of the running example, where the diffusion coefficient is never zero. It is defined in (24).

$$\begin{aligned} dx(t) = {} & \mathbb{1}_{(-\infty,\pi/2)}(([0][x(t)] + [1])\, dt + [-1]\, dW(t)) \\ & + \mathbb{1}_{(\pi/2,+\infty)}(([0][x(t)] + [-1])\, dt + [1]\, dW(t)) \\ & + \mathbb{1}_{[\pi/2,\pi/2]}(([0][x(t)] + [0])\, dt + [1]\, dW(t)) \end{aligned} \tag{24}$$

   There are three locations. In the third location, the diffusion coefficient is non-zero, in contrast to SW. The objective is to maintain the steering wheel position at $\pi/2$ (Fig. 1a), while there is continuous disturbance.
4. Coupled dynamical system (CDS): This is a sliding mode control example, with continuous variables evolving with coupled SDEs. The benchmark is shown in (25). There are 9 locations, and in (25), $\mathbf{x}(t) = [x1(t), x2(t)]^T$. The aim of this

benchmark is to drive the two variables onto the stable point (5, 0), irrespective of the starting values and under disturbance.

$$\begin{aligned}
d\mathbf{x}(t) = \; & \mathbb{1}_{([5,5],[0,0])}(([0,0;0,0]\mathbf{x}(t) + [0;0])dt + [0;0]dW(t)) \\
& + \mathbb{1}_{([5,5],(0,+\infty))}(([0,0;1,1]\mathbf{x}(t) + [0;-10])dt + [0;-1]dW(t)) \\
& + \mathbb{1}_{([5,5],(-\infty,0))}(([0,0;1,1]\mathbf{x}(t) + [0;10])dt + [0;1]dW(t)) \\
& + \mathbb{1}_{((-\infty,5),[0,0])}(([0,1;0,0]\mathbf{x}(t) + [4;0])dt + [1;0]dW(t)) \\
& + \mathbb{1}_{((5,+\infty),[0,0])}(([0,1;0,0]\mathbf{x}(t) + [-4;0])dt + [-1;0]dW(t)) \\
& + \mathbb{1}_{((-\infty,5),(-\infty,0))}(([0,1;1,1]\mathbf{x}(t) + [4;10])dt + [1;1]dW(t)) \\
& + \mathbb{1}_{((5,+\infty),(0,+\infty))}(([0,1;1,1]\mathbf{x}(t) + [-4;-10])dt \\
& + [-1;-1]dW(t)) \\
& + \mathbb{1}_{((-\infty,5),(0,+\infty))}(([0,1;1,1]\mathbf{x}(t) + [4;-10])dt \\
& + [1;-1]dW(t)) \\
& + \mathbb{1}_{((5,+\infty),(-\infty,0))}(([0,1;1,1]\mathbf{x}(t) \\
& + [-4;10])dt + [-1;1]dW(t))
\end{aligned} \tag{25}$$

6.2 Experimental setup

We show two different measures of accuracy of strong convergence, following [18, 23] given in (26) and (27), respectively, comparing the numerical solution with a reference solution.

$$\begin{aligned}
e &= (\mathbb{E}|(\mathbf{x}(T_{sim}) - \mathbf{x}[T_{sim}])|) \\
e1 &= \sqrt{1/M \times \sum_{m=1}^{M} (\sum_{\forall x \in \mathbf{x}} (x(T_{sim}) - x[T_{sim}])^2)}
\end{aligned} \tag{26}$$

$$e2 = 100/M \times (\sum_{m=1}^{M} (\sum_{\forall x \in \mathbf{x}} |(x(T_{sim}) - x[T_{sim}])/x(T_{sim})|)) \tag{27}$$

Equation (26) computes the expected value ($\mathbb{E}$) of the error between the numerical solution ($x[T_{sim}]$) and the reference solution ($x(T_{sim})$) as the root mean square error (RMSE). Equation (27) on the other hand computes the error as the mean absolute percentage error (MAPE). Both these error metrics are computed over M Monte Carlo runs.

In the general case, there is no analytical solution for a SHS. Hence, the reference solution $x(T_{sim})$ is also computed numerically. In our case, the simulation time T_{sim} is divided into Y steps, such that $T_{sim} = \sum_{s=1}^{Y} \Delta_s = \sum_{s=1}^{Y} \delta_s \times R$, where simulation/integration step Δ_s—either the level crossing step $\Delta^x_{\tau_m}$ or an error bound step Δ^x_{ϵ}—is computed dynamically by Algorithm 1 (line 21). Each step Δ_s is divided into finer steps of size δ_s, such that $\Delta_s = \delta_s \times R$ as defined in Section 4.1. For each integration step Δ_s, $x[T + \Delta_s]$ is computed by Algorithm 1, line 23 as shown in (28), where $T \in [0, T_{sim})$. We have an equivalent reference solution, computing $x(T + \Delta_s)$

using a finer time step δ_s, on the same Wiener path, as shown in (29). Hence, the reference solution $x(T_{sim}) = x(T + \Delta_s)$, when $s = Y$.

$$x[T + \Delta_s] = x[T] + (A_{q(T)} \bullet x[T] + B_{q(T)}) \times \Delta_s \\ +B'_{q(T)} \times \sqrt{\delta}_s \times (\sum_{i=1}^{R} \mathcal{N}_i(0, 1)) \tag{28}$$

$$x(T + \Delta_s) = x(T) + \sum_{i=1}^{R}(A_{q(T)} \bullet x(T) + B_{q(T)})\delta_s \\ +B'_{q(T)} \times \sqrt{\delta}_s \times N_i(0, 1) \tag{29}$$

6.3 Results

All experiments were carried out on Intel® Core i5 2.9GHz CPU with 8GB RAM, running OSX 10.11.6. For all experiments, we set $T_{sim} = 1$ s and $M = 1000$ Monte Carlo runs. The Python implementation of the solver and the benchmarks are available from [24].

We vary $R \in \{2, 4, 8\}$ and $\epsilon \in \{1e-4, 1e-8\}$, which controls each dynamic step size (Algorithm 2, lines 20–24). The RMSE and MAPE errors computed using (26) and (27), respectively, between the output from the proposed algorithm and the reference trace are listed in Table 1. For SW and SWD benchmarks, RMSE and MAPE are almost zero (in the order 10^{-14}) in all cases. For the TC example, the maximum error is 0.035% between the numerical and the reference solution. However, this error decreases to the order of 10^{-6} when $\epsilon = 1e-8$. Similar results are seen for the CDS example. Overall, for all our benchmarks, the numerical solution performs well with the expected error never exceeding 0.2% for 1000 Monte Carlo runs.

The output trace, for one Monte Carlo run, for the running example (SW) has been shown previously in Fig. 1d. The output traces, for one of the Monte Carlo run, for the rest of the benchmarks is shown in Fig. 4. Figure 4a shows the TC benchmark twisting towards the (0, 0) stable point. Unlike the SW benchmark, the SWD benchmark (Fig. 4b) reaches the $\pi/2$ position, but is then pushed away from this position, because the diffusion coefficient is 1 in the third location of (24). Hence, in Fig. 4b, we see the SMC trying to bring the position of the steering wheel back to $\pi/2$ continuously. Figure 4d and c show two unique solutions for the CDS benchmark. The objective of the CDS benchmark is to guide $x1(t)$ and $x2(t)$ to the stable point (5, 0). In Fig. 4c, $x1(t)$ reaches point 5, earlier than $x2(t)$ reaches 0. Hence, the system starts sliding on the line $5 \pm dW(t) \times x2(t)$ until $x2(t)$ becomes 0 and the system reaches the stable point. Figure 4d is the dual of Fig. 4c, where $x1(t)$ reaches 0 before $x2(t)$ reaches 5 and we see the expected sliding behaviour adhering to the dynamics described in (25).

Table 1 RMSE and MAPE errors for generated output vs. reference output with varying R and ϵ

(a) RMSE and MAPE for SW

$R \backslash \epsilon$	1e-4		1e-8	
	$e1$	$e2$	$e1$	$e2$
2	8.8e-14	5.6e-14	2.2e-16	1.4e-14
4	1.7e-15	1.1e-14	4.4e-16	2.8e-14
8	1.1e-15	7.1e-14	2.2e-15	1.4e-13

(b) RMSE and MAPE for TC

$R \backslash \epsilon$	1e-4		1e-8	
	$e1$	$e2$	$e1$	$e2$
2	1.3e-4	2.2e-3	1.5e-7	4.8e-6
4	3.9e-4	0.01	1.3e-7	4.2e-6
8	2.6e-3	0.035	2e-7	6.6e-6

(c) RMSE and MAPE for SWD

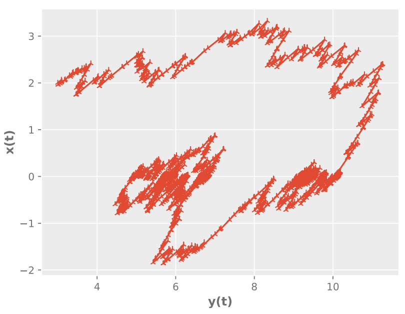
$R \backslash \epsilon$	1e-4		1e-8	
	$e1$	$e2$	$e1$	$e2$
2	6e-16	5e-14	2.4e-16	1.5e-14
4	1.3e-15	8.7e-14	1.1e-15	3e-14
8	1.1e-15	6.9e-14	4.4e-16	1.6e-14

(d) RMSE and MAPE for CDS

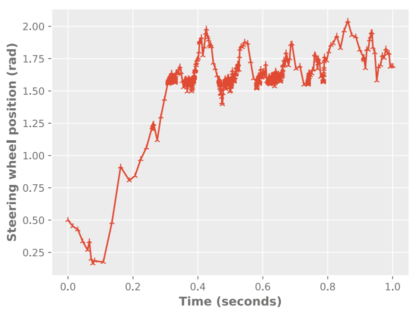
$R \backslash \epsilon$	1e-4		1e-8	
	$e1$	$e2$	$e1$	$e2$
2	5e-2	3.5e-2	5.9e-5	1.1e-5
4	1.4e-2	0.11	3.4e-7	4.1e-5
8	4.8e-3	0.1	4e-7	7e-5

7 Discussion and related work

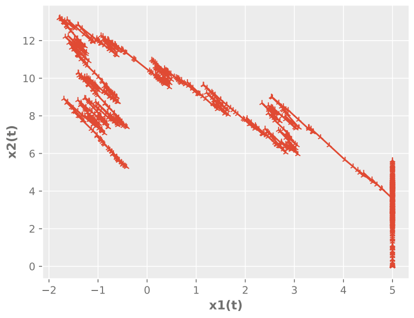
The numerical analysis community has developed adaptive step size numerical integration techniques for SDEs *without* discontinuities [13, 18, 25, 26]. Many of these techniques require a higher order numerical integration technique, e.g. Heun's or Milstein's method embedded with Itô' Taylor expansion [12] to compute solutions to SDEs at each integration step. Recent work on strong convergence analysis of SDEs with discontinuities [19, 23, 27] shows that higher order numerical integration techniques do not work when considering SDEs with discontinuities. Furthermore, strong



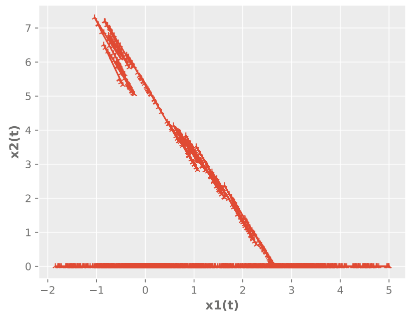
(a) Trace for TC, $y(0) = 2$, $x(0) = 3$, $b = 3$, $s = 1$.



(b) Trace for SWD, $x(0) = 0.5$.



(c) First trace for CDS, $x1(0) = -0.5$, $x2(0) = 5.4$



(d) Second trace for CDS, $x1(0) = -0.5$, $x2(0) = 5.4$

Fig. 4 Example traces for various benchmarks. $T_{sim} = 1$ s, $R = 4$, and $\epsilon = 1e - 4$

convergence guarantees can only be given, for SDEs with discontinuities, if they are piecewise Lipschitz continuous. The subset of the SHS that we handle (see (11)), in this work, are linear and time-invariant SDEs in *each* location of the SHS. The linear and time-invariant construction in (11) guarantees local Lipschitz continuity and hence, a strong unique solution for each location.

The SDEs in (11) have constant diffusion coefficients. Given a SDE with single-dimensional state-dependent diffusion coefficients, the Lamperti transformation [28, 29] can be used to transform state-dependent diffusion coefficients to constant or unit diffusion coefficients. The resultant SDE can be integrated using the proposed technique. Furthermore, the original Lamperti transform in [28] is extended to work with a subset of SDEs with multi-dimensional[1] diffusion coefficients in [29]. In particular, applying the Lamperti transform from Theorems 4 and 5 in [29] makes each continuous variable dependent only upon an individual Wiener process with unit diffusion coefficient. The resultant SDEs can then be integrated using the proposed technique. The only change required is updating Definition 6. Instead of a random sample vector, we need a random sample matrix, with each row being the R dimensional random sample vector for each Wiener process.

[1] More than one Wiener process.

The proposed numerical integration technique computes each integration step such that the difference between the current value of the continuous variables and the values satisfying the discontinuity reduces by some quantum at each step. This is similar to quantised state system (QSS) integration [30] techniques used for simulating HA [31] and standard ODEs. In contrast to the proposed numerical integration technique that handles SDEs with discontinuous drift and diffusion, the recent numerical integration technique in [27] handles SDEs with discontinuous drifts, but continuous diffusion. Moreover, the work in [27] does not describe an explicit algorithm to compute the integration step. It is rather geared towards computing the asymptotic complexity of convergence for given integration step size(s).

The CPS community has developed different variants of SHS [2]. Of particular interest have been SHSs which allow spontaneous jumps, from a location, via Poisson process and SDE with single-dimensional Wiener process [2, 32]. The CPS community has also developed formal semantics based on continuous and discrete time Markov processes for model checking SHS [10, 33, 34]. The proposed work on numerical simulation of SHS can be considered orthogonal to verification and model checking techniques in the CPS community. Many of the model checking techniques internally use Euler-Maruyama numerical solution for solving SDEs in each location and hence, we expect that the proposed work would benefit the verification community in the future.

8 Conclusion and future work

This work presents an adaptive step size simulation/integration technique for stochastic differential equations (SDEs) with discontinuous drift and diffusion coefficients. Convergence to the discontinuity is achieved by making each integration step dependent upon the value of the continuous variables inducing the discontinuity. At any given integration step, the Euler-Maruyama numerical solution is set less than or equal to the value of the continuous variable inducing the discontinuity, and then taking the root of the resultant polynomial as the integration step size. Analysis and extensive benchmarking shows the efficacy of the proposed integration technique in correctly simulating complex stochastic hybrid systems (SHSs).

The current approach is restricted to a subset of SHSs, where SDEs, in each location of the SHS, are linear and time-invariant. Furthermore, discontinuities are also explicitly defined. Many open questions about handling transcendental guards, non-linear SDEs, and SHSs with Poisson processes remain to be explored in the future.

References

1. Pola, G., Bujorianu, M.L., Lygeros, J., Di Benedetto, M.D.: Stochastic hybrid models: an overview. In: ADHS, pp. 45–50 (2003)
2. Cassandras, C.G., Lygeros, J.: Stochastic Hybrid Systems. CRC Press, Boca Raton (2018)
3. Alur, R.: Principles of Cyber-Physical Systems. MIT Press, Cambridge (2015)

4. Glover, W., Lygeros, J.: A stochastic hybrid model for air traffic control simulation. In: International Workshop on Hybrid Systems: Computation and Control, pp. 372–386. Springer (2004)
5. Poznyak, A.: Stochastic sliding mode control: what is this? In: 2016 14th International Workshop on Variable Structure Systems (VSS), pp. 328–333. IEEE (2016)
6. Hespanha, J.P.: Stochastic hybrid systems: application to communication networks. In: International Workshop on Hybrid Systems: Computation and Control, pp. 387–401. Springer (2004)
7. Hu, J., Lygeros, J., Sastry, S.: Towards a theory of stochastic hybrid systems. In: International Workshop on Hybrid Systems: Computation and Control, pp. 160–173. Springer (2000)
8. Fränzle, M., Gao, Y., Gerwinn, S.: Constraint-solving techniques for the analysis of stochastic hybrid systems. In: Provably Correct Systems, pp. 9–38. Springer (2017)
9. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.-P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods in System Design **43**(2), 191–232 (2013)
10. David, A., Du, D., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. arXiv:1208.3856 (2012)
11. Kloeden, P.E., Platen, E.: Numerical Solution of Stochastic Differential Equations, vol. 23. Springer, Berlin (2013)
12. Itô, K.: 109. Stochastic integral. Proceedings of the Imperial Academy **20**(8), 519–524 (1944)
13. Gaines, J.G., Lyons, T.J.: Variable step size control in the numerical solution of stochastic differential equations. SIAM J. Appl. Math. **57**(5), 1455–1484 (1997)
14. Rackauckas, C., Nie, Q.: Adaptive methods for stochastic differential equations via natural embeddings and rejection sampling with memory. Discrete and Continuous Dynamical Systems, Series B **22**(7), 2731 (2017)
15. MathWorks Corp: Simulink: Simulation and Model-Based Design. https://www.mathworks.com/products/simulink.html/ (2019)
16. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Hybrid Systems, pp. 209–229. Springer, London (1993)
17. Open Source Modelica Consortium (OSMC): Openmodelica. https://openmodelica.org/index.php/ (2019)
18. Burrage, P.M., Burrage, K.: A variable stepsize implementation for stochastic differential equations. SIAM J. Sci. Comput. **24**(3), 848–864 (2003)
19. Ngo, H.-L., Taguchi, D.: On the Euler–Maruyama approximation for one-dimensional stochastic differential equations with irregular coefficients. IMA J. Numer. Anal. **37**(4), 1864–1883 (2017)
20. Fridman, L., Levant, A., et al.: Higher order sliding modes. Sliding Mode Control in Engineering **11**, 53–102 (2002)
21. Cruz, G.L., Alazki, H., Hernández, R.G.: Super twisting control for thermo's catalyst-5 robotic arm. IFAC-PapersOnLine **51**(13), 303–308 (2018)
22. Kumari, K., Chalanga, A., Bandyopadhyay, B.: Implementation of super-twisting control on higher order perturbed integrator system using higher order sliding mode observer. IFAC-PapersOnLine **49**(18), 873–878 (2016)
23. Göttlich, S., Lux, K., Neuenkirch, A.: The Euler scheme for stochastic differential equations with discontinuous drift coefficient: a numerical study of the convergence rate. Advances in Difference Equations **2019**(1), 429 (2019)
24. Malik, A.: Benchmarks. https://github.com/amal029/eha. Last Accessed 05 April 2020 (2020)
25. Lamba, H.: An adaptive timestepping algorithm for stochastic differential equations. J. Comput. Appl. Math. **161**(2), 417–430 (2003)
26. Ilie, S., Jackson, K.R., Enright, W.H.: Adaptive time-stepping for the strong numerical solution of stochastic differential equations. Numerical Algorithms **68**(4), 791–812 (2015)
27. Neuenkirch, A., Szolgyenyi, M., Szpruch, L.: An adaptive Euler–Maruyama scheme for stochastic differential equations with discontinuous drift and its convergence analysis. SIAM J. Numer. Anal. **57**(1), 378–403 (2019)
28. Lamperti, J.: A simple construction of certain diffusion processes. J. Math. Kyoto Univ. **4**(1), 161–170 (1964)
29. Møller, J.K., Madsen, H.: From State Dependent Diffusion to Constant Diffusion in Stochastic Differential Equations by the Lamperti Transform, ser. IMM-Technical Report-2010-16. Technical University of Denmark, DTU Informatics, Building 321 (2010)

30. Kofman, E., Junco, S.: Quantized-state systems: a DEVS approach for continuous system simulation. Transactions of The Society for Modeling and Simulation International **18**(3), 123–132 (2001)
31. Malik, A., Roop, P.: A dynamic quantized state system execution framework for hybrid automata. Nonlinear Analysis: Hybrid Systems **36**, 100870 (2020)
32. Lygeros, J., Prandini, M.: Stochastic hybrid systems: a powerful framework for complex, large scale applications. Eur. J. Control. **16**(6), 583–594 (2010)
33. Abate, A., Katoen, J.-P., Lygeros, J., Prandini, M.: Approximate model checking of stochastic hybrid systems. Eur. J. Control. **16**(6), 624–641 (2010)
34. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control, pp. 43–52 (2011)