Nonlinear Analysis: Hybrid Systems 46 (2022) 101234

Contents lists available at ScienceDirect

Nonlinear Analysis: Hybrid Systems

journal homepage: www.elsevier.com/locate/nahs

Efficient simulation of general stochastic hybrid systems

Avinash Malik

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

ARTICLE INFO

Article history:
Received 5 January 2021
Received in revised form 22 February 2022
Accepted 6 June 2022
Available online 22 June 2022

Keywords:
Numerical algorithms
Stochastic hybrid systems
Synchronous semantics
Wiener process
Jump diffusion process

ABSTRACT

General Stochastic Hybrid System (SHS) are characterised by Stochastic Differential Equations (SDEs) with discontinuities and Poisson jump processes. SHS are useful in model based design of Cyber-Physical System (CPS) controllers under uncertainty. Industry standard model based design tools such as Simulink/Stateflow® are inefficient when simulating, testing, and validating SHS, because of dependence on fixed-step Euler–Maruyama (EM) integration and discontinuity detection. We present a novel efficient adaptive step-size simulation/integration technique for general SHSs modelled as a network of Stochastic Hybrid Automatons (SHAs). We propose a simulation algorithm where each SHA in the network executes synchronously with the other, at an integration step-size computed using adaptive step-size integration. Ito' multi-dimensional lemma and the inverse sampling theorem are leveraged to compute the integration step-size by making the SDEs and Poisson jump rate integration dependent upon discontinuities. Existence and convergence analysis along with experimental results show that the proposed technique is substantially faster than Simulink/Stateflow® when simulating general SHSs.

1. Introduction

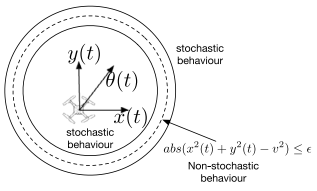
Hybrid systems are a modelling framework, which are often used for model based design of Cyber-Physical Systems (CPSs) [1]. Stochastic Hybrid Systems (SHSs) are a subset of hybrid systems where a continuous time phenomenon (usually termed the plant) with uncertainty is controlled using a discrete controller. Stochastic Hybrid Automaton (SHA) [2] is a formal framework designed to capture SHS phenomenon. SHA provides a clean syntax and associated semantics [2], which enables model based design [3] of stochastic CPS controllers. Model based design of SHS starts with first describing the behaviour of the plant and the controller (SHA) within a simulation environment such as Simulink/Stateflow®. The required controller adjustments are made within the simulation environment and the controller is extracted for execution on an embedded processor.

The main challenge in simulation of SHA is correctly capturing the sudden discontinuities induced by controller switches [4,5]. In particular, the stochastic nature of the plant may lead to multiple level crossings (discontinuities) occurring very close to each other. In such scenarios it is paramount that the very first level crossing (discontinuity) is captured else, the simulation trace can significantly diverge from the real solution. The problem of discontinuity detection becomes even more challenging in the presence of concurrently executing SHAs inducing their own discontinuities and a non-chattering controller implementation.

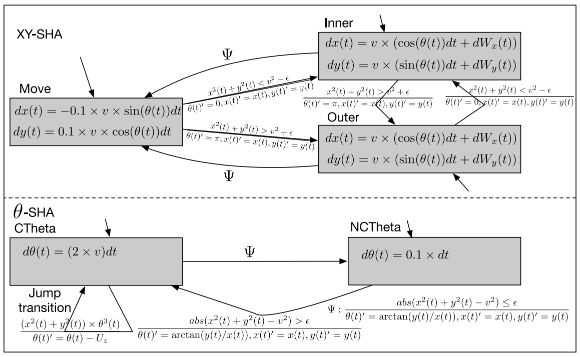
Fixed step Euler–Maruyama (EM) [6] is the standard numerical integration technique employed for integrating Stochastic Differential Equations (SDEs). The fixed step EM technique is known to converge to a unique solution [6].

E-mail address: avinash.malik@auckland.ac.nz.

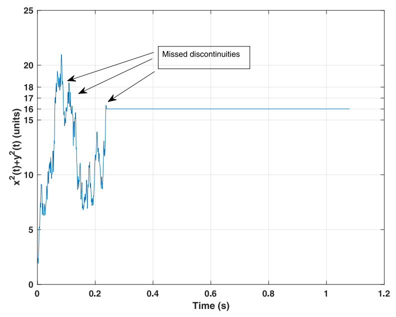
https://doi.org/10.1016/j.nahs.2022.101234
1751-570X/© 2022 Elsevier Ltd. All rights reserved.



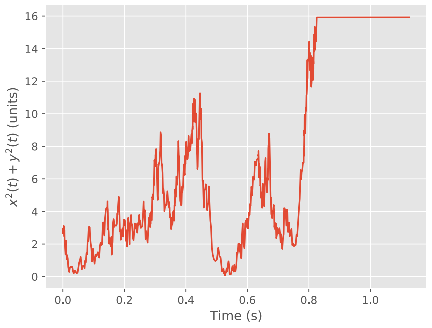
(a) Stochastic point UAV


XY-SHA
Inner
$dx(t) = v \times (\cos(\theta(t))dt + dW_x(t))$
$dy(t) = v \times (\sin(\theta(t))dt + dW_y(t))$
Move
$dx(t) = -0.1 \times v \times \sin(\theta(t))dt$
$dy(t) = 0.1 \times v \times \cos(\theta(t))dt$
Outer
$dx(t) = v \times (\cos(\theta(t))dt + dW_x(t))$
$dy(t) = v \times (\sin(\theta(t))dt + dW_y(t))$
Ψ
Ψ
$\frac{x^2(t) + y^2(t) < v^2 - \epsilon}{\theta(t)' = 0, x(t)' = x(t), y(t)' = y(t)}$
$\frac{x^2(t) + y^2(t) > v^2 + \epsilon}{\theta(t)' = \pi, x(t)' = x(t), y(t)' = y(t)}$
$\frac{x^2(t) + y^2(t) > v^2 + \epsilon}{\theta(t)' = \pi, x(t)' = x(t), y(t)' = y(t)}$
$\frac{x^2(t) + y^2(t) < v^2 - \epsilon}{\theta(t)' = 0, x(t)' = x(t), y(t)' = y(t)}$
θ-SHA
CTheta
$d\theta(t) = (2 \times v)dt$
NCTheta
$d\theta(t) = 0.1 \times dt$
Ψ
Jump transition
$\frac{(x^2(t) + y^2(t)) \times \theta^3(t)}{\theta(t)' = \theta(t) - U_z}$
$\frac{abs(x^2(t) + y^2(t) - v^2) > \epsilon}{\theta(t)' = \arctan(y(t)/x(t)), x(t)' = x(t), y(t)' = y(t)}$
$\Psi : \frac{abs(x^2(t) + y^2(t) - v^2) \leq \epsilon}{\theta(t)' = \arctan(y(t)/x(t)), x(t)' = x(t), y(t)' = y(t)}$

(b) The SHA model of the point UAV. Note that edges with the same guard and resets are labelled with the shared Ψ label.



(c) SHA model output from Simulink/Stateflow® with integration step-size=10^{-3}



(d) The SHA model output from the proposed technique with adaptive integration step-size

Fig. 1. SHA modelling a point UAV and its output via Simulink/Stateflow® and the proposed technique. The outputs are generated for a single Montecarlo run with $x(0) = y(0) = 1$, $\theta(0) = \arctan(y(0)/x(0))$, $v = 4$, and $\epsilon = 10^{-1}$.

However, the strong rate of convergence is proportional to the square-root of the integration step-size [6]. In presence of discontinuities, a very small step-size might be necessary to correctly simulate the SHA. Thereby making the simulation extremely inefficient. In the next subsection we show the problem with a stochastic UAV example and then state our contributions.

1.1. Running example and problem description

Fig. 1 is the running example with Fig. 1(a) showing the stochastic model of a point UAV [7]. The continuous variables $x(t)$, $y(t)$, and $\theta(t)$ show the 2D position and the heading of the UAV, respectively. The UAV moves stochastically outside the bounds of the two solid circles described by the equation: $abs(x^2(t) + y^2(t) - v^2) \leq \epsilon$. As soon as the UAV hits this discontinuity (circle); the UAV starts moving non-stochastically.

The UAV motion is modelled as a SHA in Fig. 1(b). There are two concurrent SHAs. SHA XY-SHA models the 2D position, and the θ-SHA models the heading, respectively. The XY-SHA has three locations, Inner, Outer, and Move indicating the movement of the UAV inside, outside, and within the circular bounds, respectively. Either of these locations might be the initial location depending upon the initial conditions. The model moves from location Inner to Move once the discontinuity is hit, else if the UAV goes past the discontinuity undetected, then the model moves into the Outer location. The heading is reset in each of these cases such that UAV moves towards the discontinuity. Same for the Outer location. In case of the θ-SHA, we have two locations indicating the stochastic changes in the heading (location CTheta) and the non-stochastic change in the heading (location NCTheta), respectively. It is worth noting that the dynamics of the $x(t)$ and $y(t)$ variables are Wiener processes [6], whereas the stochastic change of the heading $\theta(t)$ is modelled as a Poisson jump process in location CTheta. The rate of the Poisson jump is $(x^2(t) + y^2(t)) \times \theta^3(t)$. Upon taking this jump transition, the heading is reset by some random value drawn from a uniform $\mathcal{U}(0, 1)$ distribution; U_z.

Fig. 1(c) shows the output trace generated from the stochastic UAV model implemented in Simulink/Stateflow®, for the default integration step-size of 10^{-3} using the fixed step-size EM integration technique. As we can see, the discontinuity is missed a number of times. The UAV starts from the initial position $x(0) = y(0) = 1$ and heading $\theta(0) = \arctan(y(0)/x(0)) = 0.78$ radians. For $v = 4$ and $\epsilon = 10^{-1}$, if the discontinuity is not missed, then it is expected that the value of $x^2(t)+y^2(t)$ would become 16 ± 10^{-1}. However, as we can see, the UAV crosses from Inner to the Outer location and vice-versa multiple times before settling on the discontinuity. Correct discontinuity (level-crossing) detection is a well-known problem in hybrid system design [4,5].

The correct output from the proposed technique is shown in Fig. 1(d). The proposed simulation technique correctly detects the very first level-crossing and enters the Move location as soon as the level-crossing is detected. Moreover, the proposed technique only takes 645 integration steps to simulate 1.1 s of the model, compared to Simulink/Stateflow® taking 1100 integration/simulation steps to produce the incorrect output. Correct behaviour can be generated from Simulink/Stateflow® with a step-size of 10^{-5}. However, this significantly increases the simulation time to 110,000 simulation steps, which is a 170× performance degradation compared to the proposed technique.

The overarching **contribution** of this paper is to propose an *adaptive* step-size simulation/integration technique for concurrent SHAs with improved simulation performance. Furthermore, our technical contributions can be outlined as follows:

1. The proposed simulation technique can correctly simulate linear, non-linear, closed, and open edge guards inducing level-crossings.
2. The proposed simulation technique is compositional, in that, the concurrent SHAs, in a network, do not need to be syntactically composed. Instead a synchronous compositional simulation technique is provided.
3. An analysis of the proposed simulation technique is provided that guarantees simulation progress and convergence to the very first level crossing for well-formed SHAs.
4. Finally, a number of experimental results show the efficacy of the proposed technique compared to the industry standard Simulink/Stateflow® simulation tool.

The paper is organised as follows; Section 2 gives the preliminaries required to read the rest of the paper. Section 3 then gives the standard syntax and semantics of the SHA. The numerical simulation algorithm and synchronous composition is provided in Section 4, followed by the analysis of the algorithm in Section 5. Experimental results comparing efficiency vis-a-vis Simulink/Stateflow® is provided in Section 6. Comparison with the current state-of-the-art adaptive step-size simulation algorithms for stochastic processes is carried out in Section 7. Finally, we conclude in Section 8.

2. Preliminaries

In this section we give the preliminaries required to understand the rest of the paper. Sections 2.1–2.4 describe the basics of Wiener process, followed by Poisson jump process in Section 2.5.

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.

Following standard practice [8,9], 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*, numerical approximation of the 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 Eq. (1). In Eq. (1) and the rest of the paper, symbol $\therefore$ stands for "therefore".

$$W[1] = W[0] + \sqrt{\delta} \times \mathcal{N}(0, 1), \text{ from (2) above}$$

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

$$W[s] = \sqrt{\delta} \times \sum_{i=1}^{j} (\mathcal{N}_i(0, 1)) \text{ from (1) above} \tag{1}$$

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 $W[s + \Delta] - W[s]$, following from Eq. (1), as shown in Eq. (2).

from Eq. (1) we have

$$W[s+\Delta] - W[s] = (\sqrt{\delta}\sum_{i=1}^{M+R}\mathcal{N}_i(0,1)) - (\sqrt{\delta}\sum_{i=1}^{M}\mathcal{N}_i(0,1))$$
$$\therefore W[s+\Delta] - W[s] = \sqrt{\delta}\times(\sum_{i=M}^{M+R}\mathcal{N}_i(0,1)) = \sqrt{\delta}\times(\sum_{i=1}^{R}\mathcal{N}_i(0,1)) \tag{2}$$

2.2. Fixed step Euler–Maruyama solution to SDE

A scalar, autonomous SDE is shown in Eq. (3) and its solution in Eq. (4), 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 Eq. (3) is an Ordinary Differential Equation (ODE).

$$dx(t) = f(x(t))\,dt + g(x(t))\,dW(t) \tag{3}$$
$$\therefore x(t) = x(0) + \int_0^t f(x(s))\,ds + \int_0^t g(x(s))\,dW(s), \forall t \in [0,T] \tag{4}$$

The numerical Euler–Maruyama (EM) solution [6] to Eq. (4), at time T, with $x(0)$ as the initial value, is given in Eq. (5). In Eq. (5), 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 Eq. (5) devolves to standard forward Euler solution of an ODE. The EM solution is unique and convergent if $f(x(t))$ and $g(x(t))$ are Lipschitz continuous and bounded. We *assume* Lipschitz continuous and bounded $f(x(t))$ and $g(x(t))$, respectively.

$$x[T] = x(0) + \sum_{j=1}^{Y}(f(x[j-1])\Delta + g(x[j-1])(W[s+\Delta]-W[s]))$$
from Eq. (2) we have for $\Delta = \delta \times R$
$$\therefore x[T] = x(0) + \sum_{j=1}^{Y}(f(x[j-1])\Delta + g(x[j-1])(\sqrt{\delta}\sum_{i=1}^{R}\mathcal{N}_i(0,1))) \tag{5}$$

2.3. Multi-dimensional stochastic process

A multi-dimensional SDE is characterised by multiple independent Wiener processes and is defined in Definition 2.

Definition 2. Let $W_1(t), \ldots, W_n(t)$, be independent Wiener processes. A multi-dimensional SDE for continuous vector $\boldsymbol{X}(t) = [X_1(t); X_2(t); \ldots; X_n(t)]$ is defined in Eq. (6).

$$dX_i(t) = f_i(\boldsymbol{X}(t))dt + g_i dW_i(t), \forall i \in \{1,\ldots,n\}$$
$$d\boldsymbol{X}(t) = [dX_1(t); \ldots; dX_n(t)] \tag{6}$$

Remark 1. In Eq. (6):

1. The diffusion coefficients are constants, i.e., we have only additive noise.
2. Every stochastic process $X_i(t)$ in Eq. (6) only depends upon its own Wiener process. This requirement can be relaxed such that $X_i(t)$ and $X_j(t)$, $i \neq j$ share Wiener processes as shown in [8].
3. Multiplicative noise process can be translated into additive noise with constant diffusion coefficients using the Lamperti transform [10,11].
4. Every stochastic process $X_i(t)$ defined in Eq. (6) can be integrated from $[0,T]$, using Eq. (5).
5. The diagonal matrix of diffusion coefficients can be possibly degenerate.

2.4. Multi-dimensional Ito's lemma

Ito's lemma [12] relates a function of stochastic processes with their SDEs as shown in Definition 3.

Definition 3. Let $\boldsymbol{X}(t)$ be a vector of stochastic processes as defined in Definition 2. Then given a twice continuously differentiable function $G : \mathbb{R}^n \to \mathbb{R}$, $dG(\boldsymbol{X}(t))$ is also a SDE as shown in Eq. (7).

$$dG(\boldsymbol{X}(t)) = \sum_{i=1}^{n}\mathcal{G}_i dX_i(t) + 1/2\sum_{i=1}^{n}\sum_{m=1}^{n}\mathcal{G}_{im}dX_i(t)dX_m(t),$$

$$\text{where } \mathcal{G}_i = \partial G/\partial X_i(\boldsymbol{X}(T)),\ \mathcal{G}_{im} = \partial^2 G/\partial X_i X_m(\boldsymbol{X}(T))$$
$$\text{and } dX_i(t)dX_m(t) = g_i^2 dt, \text{ if } i = m, \text{ else } 0 \tag{7}$$

In Eq. (7), $dX_i(t)$ are the SDEs of the vector as defined in Eq. (6). Notice that Eq. (7) is equivalent to the multi-variate Taylor expansion of the function $G(\boldsymbol{X}(t))$ around time instant T with a multiplication condition.

2.5. Poisson jump process and generating exponential variates

A Poisson jump process is characterised by time between events occurring from an exponential distribution. Exponentially distributed random variables have a Cumulative Distribution Function (CDF) as shown in Eq. (8). Hence, the probability of an exponentially distributed random variable $\mathcal{Y}$ being strictly greater than y, called the survivor function, is given in Eq. (9). Using the inverse of the survivor function and the inverse sampling technique [13] we can relate the variates from the exponential distribution and the standard uniform distribution ($\mathcal{U}(0, 1)$) as shown in Eq. (10). In Eq. (10), symbol $\sim$ represents equal in distribution.

$$\mathbb{P}\{\mathcal{Y} \le y\} = F(y) = 1 - \exp^{(-cy)}, c > 0 \tag{8}$$
$$\therefore \mathbb{P}\{\mathcal{Y} > y\} = 1 - F(y) = \exp^{(-cy)} \tag{9}$$
$$\therefore cy = -\ln(\mathbb{P}\{\mathcal{Y} > y\}) = -\ln(p)$$
$$\therefore \text{via inverse sampling, } c\mathcal{Y} \sim -\ln(\mathcal{U}(0, 1)) \tag{10}$$

3. Syntax and semantics

In this section we describe the syntax and semantics of a SHA. Our syntax and semantics are in the same vein as in [2,14–16].

Definition 4. Stochastic Hybrid Automaton (SHA) $\mathcal{H}$ is a tuple
$\langle V, E, \boldsymbol{X}, \mathit{flow}, \mathit{init}, \mathit{guard}, \mathit{reset}, \lambda\rangle$.[1]

- V is the set of vertices of a SHA, also called the control modes/locations.
- $E \subseteq V \times V$ is the set of edges connecting the vertices.
- $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ is the set of real valued variables. The derivative is denoted as $\dot{X}_j$, and the change in value of X_j upon a reset is denoted as X_j'. Hence, the set $\dot{\boldsymbol{X}}$ and $\boldsymbol{X}'$ denotes the set of derivatives and updates, respectively. It is important to note that $\dot{X}_j \in \dot{\boldsymbol{X}}$ should be considered as stochastically evolving continuous variables as defined in Eq. (3).
- Each vertex $v \in V$ is associated with SDEs called flow_v, which evolve the continuous variables in $\boldsymbol{X}$ in that mode, i.e., they are SDEs of the form shown in Eq. (6). Moreover, init_v is a predicate on $\boldsymbol{X}$ specifying the admissible initial conditions for a given vertex.
- Each edge $e \in E$ is associated with guard_e — a predicate on $\boldsymbol{X}$, which forces a transition. Furthermore, each edge is also labelled with a reset_e, a formula in $\boldsymbol{X} \cup \boldsymbol{X}'$ specifying the change of the variable' values upon transition.
- Finally, an edge $e \in E$, maybe labelled with a function $\lambda_e : \mathbb{R}^n \to \mathbb{R}^{\ge 0}$, which gives the *hazard rate* of taking the edge.

We can elucidate Definition 4 using Fig. 1(b). The θ-SHA can be captured within the formal syntactic definition as follows: $V \overset{\text{def}}{=} \{\text{CTheta}, \text{NCTheta}\}$. Edges are the set
$E \overset{\text{def}}{=} \{(\text{CTheta}, \text{NCTheta}), \ldots\}$. $\boldsymbol{X} \overset{\text{def}}{=} \{x(t), y(t), \theta(t)\}$ and their derivatives and updates are given by set $\dot{\boldsymbol{X}} \overset{\text{def}}{=} \{dx(t), dy(t), d\theta(t)\}$, and $\boldsymbol{X}' \overset{\text{def}}{=} \{x(t)', y(t)', \theta(t)'\}$, respectively. An example of flow in location CTheta is given as $\mathit{flow}_{\mathit{CTheta}} \overset{\text{def}}{=} [dx(t); dy(t); d\theta(t)] = [0; 0; (2 \times v)dt]$. Note that these derivatives are ODEs, because the diffusion coefficient is zero (degenerate). Example of a SDE with non-degenerate diffusion can be see in any of the locations of the XY-SHA. The init_v predicate on location CTheta can be encoded as $\mathit{init}_{\mathit{CTheta}} \overset{\text{def}}{=} \mathit{abs}(x^2(0) + y^2(0) - v^2) > \epsilon$. An example of the guard and reset formulae is $\mathit{guard}_{(\mathit{CTheta},\mathit{NCTheta})} \overset{\text{def}}{=} \mathit{abs}(x(t)^2 + y(t)^2 - v^2 \le \epsilon)$ and $\mathit{reset}_{(\mathit{CTheta},\mathit{NCTheta})} \overset{\text{def}}{=} [\theta(t)'; x(t)'; y(t)'] = [\arctan(y(t)/x(t)); x(t); y(t)]$, respectively. An example of the Poisson hazard rate is $\lambda_{(\mathit{CTheta},\mathit{CTheta})} \overset{\text{def}}{=} (x^2(t) + y^2(t)) \times \theta^3(t)$. The hazard rate for other edges is zero, by default.

3.1. Semantics of SHA

The semantics of SHA is a stochastic execution over the standard probability space with filtration: $(\Omega, \mathcal{F}, \mathcal{F}_t, \mathbb{P})$, where Ω is a non-empty sample space. $\mathcal{F}$ is the σ-algebra, $\mathcal{F} \subseteq 2^{\Omega}$. The filtration on $(\Omega, \mathcal{F})$ is a family of σ-algebras: $\{\mathcal{F}_t\}_{t\ge 0}$, which are increasing with $\mathcal{F}_{t1} \subset \mathcal{F}_{t2}, t1 < t2$. Finally, $\mathbb{P} : \mathcal{F} \to [0, 1]$ is the probability measure. Given $TS \subseteq \mathbb{R}$, a

[1] We do not have invariants on vertices, and consider them to be always `True`.

stochastic process $x : TS \times \Omega \to \mathbb{R}^n$ is a function where for each $t \in TS, x(t, \cdot) : \Omega \to \mathbb{R}^n$ is a random variable. $\mathbb{R}^n$ is the domain of $\mathbf{X}$. We represent the stochastic process with respect to time as x_t. We give the semantics, in the same vein as [16] and refer the reader to [16] for a complete construction of the stochastic execution over the probability space.

Definition 5. A stochastic process $x_t = (V(t), \boldsymbol{X}(t))$ is called a SHA $\mathcal{H}$' execution if there exists a sequence of stopping times $T_0 = 0 \leq T_1 \leq T_2 \leq T_k, \ldots$ such that, for each $k \in \mathbb{N}$,

- $x_0 = (v(0), \boldsymbol{X}(0))$, satisfies the initial condition $init_{v(0)}$.
- For $t \in [T_k, T_{k+1})$, $v(t) = v(T_k)$ is constant and $\boldsymbol{X}(t)$ is a solution to the SDE defined by $flow_{v(t)}$.
- For *some* outgoing edge of vertex $v(T_k)$, i.e., $\exists e \in E, s.t., e = (v(T_k), \_)$:

  1. Let instant t_* be the very *first* time instant when $\boldsymbol{X}(t_*)$ satisfies the predicate $guard_e$.
  2. $\mathbb{P}\{S^k_{v(t)} > t\} = \mathbb{1}_{(t<t_*)} \exp(-\int_{T_k}^{t} \lambda_e(\boldsymbol{X}(s))ds)$. Following Eq. (10) and the first event Monte Carlo simulation method [17], we have the *first* time instant t_{**}, which satisfies; $\int_{T_k}^{t_{**}} \lambda_e(\boldsymbol{X}(s))ds \sim -\ln(\mathcal{U}(0, 1))$.

  Then, $T_{k+1} = T_k + S^k_{v(t)}$, where $S^k_{v(t)} = \min(t_*, t'_*, t''_*, \ldots, t_{**}, t'_{**}, t''_{**}, \ldots)$. Here, $t_*, t'_*, \ldots$ and $t_{**}, t'_{**}, \ldots$ indicate the first time instants for *all* outgoing edge guards and jump edges for vertex $v(T_k)$.
- Computing stop times t_* and t_{**} requires that the SDEs and hazard rates evolve their respective continuous variables towards the satisfaction of edge predicates, and edge jump conditions, respectively.
- If multiple outgoing edges, from any given vertex $v(T_k)$, can be taken at the same instant, then we choose an outgoing edge randomly.
- The valuation $\boldsymbol{X}(T_{k+1})$ is governed by $reset_e$, where $e \in E = (v(T_k), v(T_{k+1}))$.

If we consider the θ-SHA, and if initially ($T_0 = 0$) we start from location CTheta, then the continuous variables evolve according to $flow_{CTheta}$ until the first stop time T_1, which is the minimum of t_{**}; the very first time instant, which triggers the edge (*CTheta*, *CTheta*)' hazard rate and t_*, which is the very first time instant, which satisfies edge guard: $guard_{(CTheta,NCTheta)}$. Upon taking either of the edges at the resultant stop time T_1, the variables are reset according to edge' reset formulae, and then the process continues until the simulation end time *SIM_END_TIME*. If there are no jump edges, then hazard rates are considered to be zero, which in turn results in $t_{**} = \infty$.

3.2. Composition of concurrent SHAs

Given N SHAs, a pair-wise composition is carried out. Our composition definition is a modified version of the *flux* product defined in [14]. The flux product is both commutative and associative.

Definition 6. Given two SHAs
$\mathcal{H}^1 = \langle V^1, E^1, \boldsymbol{X}^1, flow^1, init^1, guard^1, reset^1, \lambda^1 \rangle$ and
$\mathcal{H}^2 = \langle V^2, E^2, \boldsymbol{X}^2, flow^2, init^2, guard^2, reset^2, \lambda^2 \rangle$, the flux product $\mathcal{H}^1 \otimes \mathcal{H}^2$ is a new SHA $\mathcal{H} = \langle V, E, \boldsymbol{X}, flow, init, guard, reset, \lambda \rangle$, where:

1. The vertices of $\mathcal{H}$ are $V = V^1 \times V^2$.
2. The edges $E \subseteq V \times V$, of $\mathcal{H}$ are of three types:

   (a) $E = ((v_i, w), (v_j, w))$, where $(v_i, v_j) \in E^1$. Furthermore, there exists a projection $\pi^1(e)$ defined for all edges e and is $\pi^1(e) = (v_i, v_j) \in E^1$.

   (b) The symmetric dual $E = ((v, w_i), (v, w_j))$, where $(w_i, w_j) \in E^2$. Furthermore, there exists a projection $\pi^2(e)$ defined for all edges e and is $\pi^2(e) = (w_i, w_j) \in E^2$.

   (c) Finally, there is a *synchronous transition* edge:
   $E = ((v_i, w_i), (v_j, w_j))$, where $(v_i, v_j) \in E^1$ and $(w_i, w_j) \in E^2$; and the projection $\pi^1(e) = (v_i, v_j)$ and $\pi^2(e) = (w_i, w_j)$.
3. $\boldsymbol{X} = \boldsymbol{X}^1 \cup \boldsymbol{X}^2$, $\dot{\boldsymbol{X}} = \dot{\boldsymbol{X}}^1 \cup \dot{\boldsymbol{X}}^2$, and $\boldsymbol{X}' = \boldsymbol{X}'^1 \cup \boldsymbol{X}'^2$.
4. For a variable $X_i \in \boldsymbol{X}^1 \cap \boldsymbol{X}^2$, $flow_{(v,w)}[X_i] = flow^1_v[X_i] + flow^2_w[X_i]$, else if $X_i \in \boldsymbol{X}^1$, $flow_{(v,w)}[X_i] = flow^1_v[X_i]$, else its dual $X_i \in \boldsymbol{X}^2$, $flow_{(v,w)}[X_i] = flow^2_w[X_i]$.
5. $init_{v,w} = init_v \wedge init_w$.
6. For any edge $e \in E$, we have three possibilities for the guards, given the three different types of edges: ① $guard_e = guard^1_e$, if $\pi^1(e) \in E^1$. ② Its dual; $guard_e = guard^2_e$, if $\pi^2(e) \in E^2$. ③ The synchronous transition: $guard_e = guard^1_e \wedge guard^2_e$, if $\pi^1(e) \in E^1$ and $\pi^2(e) \in E^2$.
7. For any edge $e \in E$, we have three possibilities for the reset, given the three different types of edges: ① $reset_e = reset^1_e$, if $\pi^1(e) \in E^1$. ② $reset_e = reset^2_e$, if $\pi^2(e) \in E^2$. ③ The synchronous transition: $reset_e = reset^1_e \cup reset^2_e$, if $\pi^1(e) \in E^1$ and $\pi^2(e) \in E^2$.

8. Similarly, for the rate we have: ① $\lambda_e = \lambda_e^1$, if $\pi^1(e) \in E^1$. ② $\lambda_e = \lambda_e^2$, if $\pi^2(e) \in E^2$. ③ The synchronous transition, $\lambda_e = \lambda_e^1 = \lambda_e^2$ if $\pi^1(e) \in E^1$ and $\pi^2(e) \in E^2$.

Consider the flux product of the first SHA XY-SHA and the second SHA θ-SHA in Fig. 1(b). The vertices of the resultant SHA is a cross product of the vertices of the individual SHAs. Hence, $V \overset{\text{def}}{=} \{(\text{Move}, \text{CTheta}), \ldots\}$. An example of an edge with *just* the second projection is $e1 \overset{\text{def}}{=} ((\text{Inner}, \text{CTheta}), (\text{Inner}, \text{CTheta}))$, with $\pi^2(e1) \overset{\text{def}}{=} (\text{CTheta}, \text{CTheta})$. Note that in this case $\pi^1(e)$ is undefined. An example of the synchronous edge, which forces a transition in both SHAs simultaneously is $e2 \overset{\text{def}}{=} ((\text{Inner}, \text{CTheta}), (\text{Move}, \text{NCTheta}))$. For this edge, both projections $\pi^1(e2) \overset{\text{def}}{=} (\text{Inner}, \text{Move})$ and $\pi^2(e2) \overset{\text{def}}{=} (\text{CTheta}, \text{NCTheta})$ are defined. Following from these two examples edges; we have the $guard_{e1} \overset{\text{def}}{=} \texttt{True}$, $guard_{e2} \overset{\text{def}}{=} guard_{(Inner,Move)} \wedge guard_{(CTheta,NCTheta)} = abs(x^2(t) + y^2(t) - v^2) \le \epsilon$. Similarly, we have $reset_{e1} \overset{\text{def}}{=} [\theta(t)'] = [\theta(t) - U_z]$ and $reset_{e2} \overset{\text{def}}{=} reset_{(Inner,Move)} \cup reset_{(CTHeta,NCTheta)} = [x'(t); y'(t); \theta'(t)] = [x(t); y(t); \arctan(y(t)/x(t))]$. The hazard rate for edge $e1$ is $\lambda_{e1} = \lambda_{(CTheta,CTheta)}$. The hazard rate for edge $e2$, in this example, is zero.

The continuous variables in the resultant SHA and their derivatives/updates are simply the union of the individual continuous variables. Hence, for the resultant SHA we have; $\boldsymbol{X} \overset{\text{def}}{=} \{x(t), y(t), \theta(t)\}$ and their respective derivatives ($\dot{\boldsymbol{X}}$), and updates ($\boldsymbol{X}'$). If the individual SHAs share continuous variables, then in any given location of the resultant SHA the SDEs are an addition of the SDEs of individual SHA. SHAs XY-SHA and θ-SHA share the continuous variable set $\boldsymbol{X}$, hence in any given location the flow would be the sum of the individual SDEs. For example, $flow_{(Move,NCTheta)} \overset{\text{def}}{=} [dx(t); dy(t); d\theta(t)] = [-0.1v\sin(\theta(t))dt + 0; 0.1v\cos(\theta(t))dt + 0; 0 + 0.1dt] = [-0.1v\sin(\theta(t))dt; 0.1v\cos(\theta(t))dt; 0.1dt]$.

The resultant SHA executes according to the semantics defined in Definition 5. The flux product builds a cross product of the locations in the individual SHAs (point (1) in Definition 6). Hence, by definition, one might expect state space explosion. However, we will describe a simulation algorithm, which does *not* require building an explicit cross product in Section 4.

4. Simulation algorithm

We describe the simulation algorithm in a top-down manner. First we describe the overall algorithm for simulating concurrent network of SHAs. Followed by the generation of the simulation/integration step-size for individual SHAs.

```
   Input   : Well-formed SHAs $\mathcal{S}$, Initial values $\mathbf{X}$, $R$, $SIM\_END\_TIME$
   Output  : Trace $\mathcal{X}$
1  $t \leftarrow 0$ ;                                                        /* The timer variable */
2  $V \leftarrow$ compute_init_locs($\mathcal{S}$, X);
3  $\mathcal{X} \leftarrow \emptyset$ ;                                        // Initialise the output trace
4  while $t \le SIM\_END\_TIME$ do
5      $\mathcal{X} \leftarrow \mathcal{X} \cup \{(t, V, X)\}$ ;                 // Adding to the output trace
       /* Wiener sample path of length R for each continuous variable                        */
6      $dWts \leftarrow \{x : \mathcal{N}(0,1) | i \in R | x \in X\}$;
7      $V' \leftarrow \emptyset$;                                      /* Initialise the next location set */
       /* Iterate through each SHA computing the integration-step size                      */
8      for $i, s \in$ enumerate($\mathcal{S}$) do
9          $V'[i], ds[i], zs[i] \leftarrow$ process($h, V[i], X, dWts, t, R$);
           /* Check if the edge may be taken. Here, 0 is considered false, anything else is True */
10         $edges[i] \leftarrow (V[i] == V'[i] \wedge ds[i] == 0)?$ false : $(V[i], V'[i])$
11     if not all $edges$ then                                   /* If any edge is taken --- stop time */
12         $\Delta \leftarrow 0$;                                    /* The integration step-size is 0 */
13         $X \leftarrow reset(\mathcal{S}, edges)$;                            /* Reset by Definition 6 */
14     else
15         $\Delta \leftarrow \min(ds)$;                         /* Minimum of all individual step-sizes */
           /* Evolve continuous variables by Eq. (5) with $Y = 1$, $\Delta = \Delta$, $\delta = \Delta/R$, and Definition 6 */
16         $X \leftarrow evolve(\mathcal{S}, V, X, dWts, \Delta, \Delta/R)$;                // Overridden method
17         $zs \leftarrow evolve\_z(X, zs, \Delta)$;                         /* Evolve rate vars by Eq. (12) */
18     $t \leftarrow t + \Delta$;                                                  /* Increment the timer */
19     $V \leftarrow V'$;                                                         /* Update locations */
20 return $\mathcal{X}$;
```

Algorithm 1: The top-level pseudocode

The top-level algorithm (Algorithm 1) takes as input the set of SHAs, $\mathcal{S}$, to be simulated. The initial values of the continuous variables, $\boldsymbol{X}$. A constant, R, which is used to determine the length of the Wiener sample path for any given step as described in Eq. (2), and the simulation end time SIM_END_TIME. The algorithm outputs the trace $\mathcal{X} = (V(t), \boldsymbol{X}(t))$ as defined in Definition 5. The algorithm starts with declaring a timer variable $t \in \mathbb{R}^{\ge 0}$ (line 1) and initialising the initial locations for each of the SHA in the system (line 2). Considering our running example in Fig. 1(b); given $\boldsymbol{X} \overset{\text{def}}{=} \{x(0), y(0), \theta(0)\} = \{1, 1, \arctan(y(0)/x(0))\}$; we have the initial location set $V = \{\text{Inner}, \text{CTheta}\}$, from line 2. Next, the output trace set $\mathcal{X}$ is declared on line 3.

The algorithm then iterates taking integration steps of size Δ and in the process incrementing the timer, and computing the output trace for each of the simulation step until the simulation end time (lines 4–19). Once the algorithm is past

SIM_END_TIME, the output trace is returned (line 20). In each iteration of the loop, the output trace is updated first (line 5). A sample path of length R is created for each of the continuous variable in the system (line 6).[2] A next location set V' is used to store the next location that each of the SHAs might transition to for any given simulation step.

The algorithm, then, iterates through each of the SHA in the network and computes the *minimum individual* step-size that each of the SHA can take such that: ① the outgoing edge guard(s) are either satisfied, or ② the hazard time, for an outgoing jump edge passes, while ensuring that the step-size is good enough such that evolution of the continuous variables is bounded by an error bound ϵ. The `process` function computes this step-size. In addition to returning the step-size in set ds, the function also returns the next possible location for each of the SHA in set V' and a rate variable set zs, which is used to compute the possibility of taking a (jump) edge with a rate annotation, respectively. Consider the running example, with the initial location in set V and valuation of variables in $\boldsymbol{X}$. The `process` function will compute the step-size for the `XY`-SHA and the θ-SHA from location Inner and CTheta, respectively. In case of the running example, for the `XY`-SHA, the returned step-size is $\min(d1, d2, d3)$, where $d1$ and $d2$ are the step-sizes, such that after taking these two steps the continuous variables satisfy the two outgoing edges from location Inner (c.f. Fig. 1(b)), respectively. Step-size $d3$ bounds the next value of the continuous variables in location Inner to within some error bound ϵ. If the returned step-size is $d1$ or $d2$, then V' for `XY`-SHA will be either Move or Outer, since the edge(s) can be taken. However, if $d3$ is the smallest, then `process` will return the next location for `XY`-SHA as Inner, i.e., there is no location change. The returned zs variable(s) for `XY`-SHA is empty, because there are no rate transitions in this SHA.

If any of the edges are taken from amongst all the SHAs (line 11) then this transition is considered to be instantaneous (line 12) and the continuous variables are reset (line 13) according to Definition 6 point (7). The then-branch (lines 11–13) accounts for all cases of edges and their reset and guard conditions in the flux product. There is no need to build the cross product of the vertices of individual SHAs explicitly. Consider the running example again; if the edge guard $guard_{(\mathrm{CTheta},\mathrm{NCTheta})}$ holds, then so does the edge guard $guard_{(\mathrm{Inner},\mathrm{Move})}$, since they are the same guard and the then branch, in the algorithm, will reset the continuous variables following Definitions 6 (point (7)) and 5. On the other hand, if the hazard time for the jump edge with rate $\lambda_{(\mathrm{CTheta},\mathrm{CTheta})}$ has passed, then only continuous variable $\theta(t)$ will be reset (c.f. Fig. 1(b)) following Definition 6 point (7).

The else-branch (lines 14–17) evolves the continuous variables according to their *flow* vectors (c.f. Definition 6, point (4)). The else-branch computes the next valuation of the continuous variables in *all* the SHAs at a synchronous step-size Δ, which is the minimum from amongst all the step-sizes for each of the SHA (line 16). The continuous variables are evolved according to the single step of the Euler–Maruyama solution of SDE (c.f. Eq. (5)). In the same spirit, the jump rate variables are evolved (line 17) (expounded further in Section 4.1.1). Finally, the timer variable is incremented and the current location set V is updated, before the algorithm reiterates.

4.1. Computing individual SHA' simulation step-size

The `process` function (Algorithm 2) computing the step-size for each individual SHA forms the core of the proposed simulation technique. The `process` function computes the integration/simulation step-size such that the *very first* stop-times for the jump and guard edges for any given location are found, while maintaining error bounds when evolving continuous variables. The `process` function is divided into two parts: ① computing the integration step-size considering the outgoing jump edges and ② computing the integration step-size considering the guards on outgoing edges, respectively. We will use the θ-SHA with current location v = CTheta, $\boldsymbol{X} \overset{\mathrm{def}}{=} \{x(0), y(0), \theta(0)\} = \{1, 1, \arctan(y(0)/x(0))\}$, and current time $T = 0$ to show the working of the function `process` in Algorithm 2.

The `process` function first identifies the outgoing jump and guard edges from the current location (lines 2–4). For the aforementioned set-up; this results in $jumpedges \overset{\mathrm{def}}{=} \{(\mathrm{CTheta}, \mathrm{CTheta})\}$, and $guardedges \overset{\mathrm{def}}{=} \{(\mathrm{CTheta}, \mathrm{NCTheta})\}$ (c.f. Fig. 1(b)). Next, the algorithm checks if any of the outgoing edge guards already hold and inserts them into the set *trueedges* (lines 5–11). Checking the guard condition in the *guardedges* set requires simple evaluation of the guard (line 8). For the current set-up, we have $guard_{(\mathrm{CTheta},\mathrm{NCTheta})} \overset{\mathrm{def}}{=} abs(x^2(T) + y^2(T) - v^2) \le \epsilon = abs(x^2(0) + y^2(0) - v^2) \le \epsilon$. If the guard holds, at time $T = 0$, then edge (CTheta, NCTheta) can be taken, and hence, it is added to the *trueedges* set. For computing the possibility of taking a jump edge, consider Definition 5; we know that the jump edge can be taken at time t_{**}, such that $\int_{T_k}^{t_{**}} \lambda_e(\boldsymbol{X}(s))ds \sim -\ln(\mathcal{U}(0,1))$. Let $\frac{dz_e(t)}{dt} = \lambda(\boldsymbol{X}[t])$ denote the jump rate and $Uz_e = -\ln(\mathcal{U}(0,1))$. Then from Fig. 2, for current simulation time $T \in [T_k, t_{**})$ we have $abs(z_e[T] - Uz_e)$ is the distance to the level crossing. Hence, checking if the jump edge is enabled is simply a matter of checking if $abs(z_e[T] - Uz_e) \le \epsilon$ (lines 5–7). Finally, if *trueedges* has a cardinality greater than one, a random outgoing edge is chosen and the resultant location is returned back, along with resetting the rate and uniform variables, so that they can be reinitialised for the next location computation (lines 9–11).

In the aforementioned set-up (at $T = 0$); the else-branch will be taken, for θ-SHA, since the outgoing edge guards/jump conditions will *not* hold at time $T = 0$. This else-branch (lines 12–15) will compute the next integration step-size by considering the jump and guard conditions along with error bounds. We describe each of the step-size computations in the next subsections.

[2] We use the set comprehension notation on line 6 similar to one used in Python.

```
1  Function process(SHA h, loc v, vars X, Wiener paths dWts, timer T, constant R):
2      $outedges \leftarrow \{e | e \in h(E) \wedge e[0] == v\}$;                    // All out edges of location v
       /* All out edges that have non zero rates                                   */
3      $jumpedges \leftarrow \{e | e \in outedges \wedge h(\lambda)(e)$ is not $0\}$;
       /* All out edges that have an expression guard defined                      */
4      $guardedges \leftarrow \{e | e \in outedges \wedge h(guard)(e)$ is $expr\}$;
       /* Initialise static uniform variables for rates                            */
5      $Uzs \leftarrow Uzs$ is $\emptyset ? \{e : -\ln(\mathcal{U}(0,1)) | e \in jumpedges\} : Uzs$;
       /* Initialise static global variables z for each jump edge                  */
6      $zs \leftarrow zs$ is $\emptyset ? \{e : 0 | e \in jumpedges\} : zs$;
       /* Check all outgoing edges that are already true                           */
7      $trueedges \leftarrow \{abs(zs[e] - Uzs[e]) \le \epsilon | e \in jumpedges\}$;
8      $trueedges \leftarrow trueedges \cup \{h(guard)(e)(X) \le \epsilon | e \in guardedges\}$;
9      if $trueedges$ is not $\emptyset$ then                                      /* If some out edge holds */
10         $trueedge \leftarrow$ random($trueedges$);                              /* Randomly pick one edge */
11         $zs \leftarrow \emptyset$; $Uzs \leftarrow \emptyset$; return $(trueedge[1], 0, zs)$
12     else
           /* Compute the step-size for the jump edges                             */
13         $dzs \leftarrow \{$rate_compute$(h(\lambda)(e), X, t, zs[e], Uzs[e], dWts, h(flow)(v), R) | e \in jumpedges\}$;
           /* Compute the step-size for the guard edges                            */
14         $dgs \leftarrow \{$guard_compute$(h(guard)(e), X, t, dWts, h(flow)(v), R) | e \in guardedges\}$;
15         return $(v, \min(dgs, dzs), zs)$
```

Algorithm 2: The `process` function computing the step size for each SHA individually.

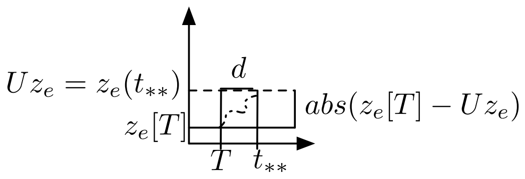


Fig. 2. Level crossing for rate variables.

4.1.1. *Computing the step-size considering jump edges*

From Fig. 2 we see that from the current simulation time T, an integration step d should exist such that $z_e[T+d] = z_e(t_{**}) = Uz_e$, where $z_e(t)$, Uz_e, for each edge $e \in jumpedges$ is as defined in Algorithm 3. Expressing Fig. 2 formally we have Eq. (11). Computing the integral of $\frac{dz_e(t)}{dt}$ using the forward Euler solution as shown in Eq. (13) we can compute the integration step-size d using only linear equations (Algorithm 3, lines 3–7).

$$\frac{dz_e(t)}{dt} = \lambda_e(\boldsymbol{X}(t)), \text{ with } z_e(T_k) = 0, \text{ and } z_e(t_{**}) = Uz_e,\ \ Uz_e = -\ln(\mathcal{U}(0,1)) \tag{11}$$

$$\forall T \in [T_k, t_{**}), z_e[T+d] = z_e[T] + \frac{dz_e(t)}{dt} d = Uz_e$$

$$\therefore z_e[T+d] = \lambda_e(\boldsymbol{X}[T])d = abs(z_e[T] - Uz_e) \tag{12}$$

$$\therefore d = \frac{z_e[T] - Uz_e}{\lambda_e(\boldsymbol{X}[T])} \vee d = \frac{-(z_e[T] - Uz_e)}{\lambda_e(\boldsymbol{X}[T])} \tag{13}$$

$$\texttt{error}(\boldsymbol{X}[T+d], \boldsymbol{X}'[T+d]) = \sum_{\forall x \in \boldsymbol{X}} |(x'[T+d] - x[T+d])/x'[T+d]| \tag{14}$$

Eq. (13) will give a step-size d, such that from any time $T \in [T_k, t_{**})$, $T+d = t_{**}$. However, d might be too large, in that the value of continuous variables $\boldsymbol{X}[T+d]$ have large error. In order to bound the step-size d in Eq. (13), the algorithm gets the value for *all* continuous variables ($\boldsymbol{X}[T+d]$) for the time step d by applying EM numerical integration (Eq. (5) and Algorithm 3, lines 8 and 9). Next, two half steps are taken; first from T to $T+d/2$ computing $\boldsymbol{X}'[T+d/2]$ and then from $T+d/2$ to $T+d$ computing the vector $\boldsymbol{X}'[T+d]$. Finally, the `error` function (Eq. (14) and Algorithm 3, lines 10–12) checks if the difference between the two vectors, $\boldsymbol{X}[T+d]$ and $\boldsymbol{X}'[T+d]$, is bounded by some user specified $\epsilon \in \mathbb{R}^{>0}$. If the difference is bounded, then the step d is accepted and returned. Else, $abs(z_e[T] - Uz_e)$ is halved and the process is iterated (line 12). The next iteration will give a new, but smaller, time step d, which satisfies the error bound. These iterations keep on happening until the error bound is satisfied.

4.1.2. *Computing the step-size considering edge guards*

Computing the step-size considering the outgoing edge guards is rather involved. Hence, we first set-up the appropriate equations, before describing the step-size computation.

```
Function rate_compute(rate expr ex, vars X, timer T, rate variable z, Uniform variable Uz, Wiener paths dWts, location flow flow, constant R):
    $d \leftarrow 0$;                                                    /* The returning step-size variable */
    $L \leftarrow abs(z - Uz)$;                                          /* Level crossing */
    while true do
        $d_1 \leftarrow \frac{L}{ex(X)}$;                                /* Compute step-1 by Eq. (13) */
        $d_2 \leftarrow \frac{-L}{ex(X)}$;                               /* Compute step-2 by Eq. (13) */
        $d \leftarrow d1 > 0?d1 : d2$;                                   /* Get the real positive step */
        /* Compute the new value of continuous variables for step d using Eq. (5), with $Y = 1$, $\Delta = d$, and $\delta = d/R$ */
        $Xs \leftarrow \{evolve(x, dWts, flow[x], d, (d/R)) | x \in X\}$;
        /* Evolve in two half steps */
        $Xs' \leftarrow \{evolve2(x, dWts, flow[x], d, (d/R)) | x \in X\}$;
        if all error$(Xs, Xs') \leq \epsilon$ then                         // Check d satisfies Eq. (14)
            break
        L /= 2;                                                          /* Half the level crossing */
    return d
```

Algorithm 3: The rate_compute function pseudocode

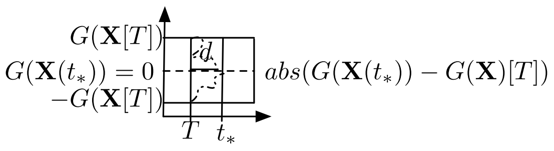


Fig. 3. Level crossing for guards.

The outgoing edge guard is a function of continuous variables $\boldsymbol{X}$. e.g., $guard_{(\text{CTheta},\text{NCTheta})} \overset{\text{def}}{=} abs(x^2(t) + y^2(t) - v^2) \leq \epsilon$. Hence, we first use the multi-dimensional Ito' lemma (Eq. (7)) to express the guard as a function of continuous variable' *flow* vectors. From Eq. (7), at any simulation time T, we have Eq. (15). From the EM approximation in Eq. (5), we have Eq. (17). Finally, applying Eqs. (16) and (17) to Eq. (15), we get the change in the guard with a step-size of d as shown in Eq. (18). In Eq. (18), $c_i = \partial G/\partial X_i(\boldsymbol{X}[T])$, $c_i' = \partial^2 G/\partial^2 X_i(\boldsymbol{X}[T])$, and $\gamma_i = \sum_{j=1}^{R} \mathcal{N}(0, 1)$, $\forall T \in [T_k, t_*)$.

$$G(\boldsymbol{X}[T+d]) - G(\boldsymbol{X}[T]) = \sum_{i=1}^{n} \mathcal{G}_i dX_i(t) + 1/2 \sum_{i=1}^{n}\sum_{m=1}^{n} \mathcal{G}_{im} dX_i(t) dX_m(t),$$

$$\text{where } \mathcal{G}_i = \partial G/\partial X_i(\boldsymbol{X}[T]),\ \mathcal{G}_{im} = \partial^2 G/\partial X_i X_m(\boldsymbol{X}[T]), \forall T \in [T_k, t_*) \tag{15}$$

$$dX_i(t) dX_m(t) = g_i^2 dt, \text{ if } i = m, \text{ else } 0 \tag{16}$$

$$dX_i(t) = X_i[T+d] - X_i[T] = f_i(\boldsymbol{X}[T])d + g_i\sqrt{d/R}\sum_{j=1}^{R} \mathcal{N}(0, 1) \tag{17}$$

$$G(\boldsymbol{X}[T+d]) = G(\boldsymbol{X}[T]) + \sum_{i=1}^{n} c_i(f_i d + \sqrt{d/R}\gamma_i) + 1/2\sum_{i=1}^{n} c_i' g_i^2 d \tag{18}$$

Now we are ready to describe the computation of the step-size d, considering the edge guards. From Definition 5, the outgoing edge guard holds at time t_*. This means at this instant $G(\boldsymbol{X}(t_*)) = 0$, i.e., the level crossing happens.[3] Hence, for all $T \in [T_k, t_*)$, we can relate the step-size d from T and time instant t_* as shown in Fig. 3. The guard value approaches the level $G(\boldsymbol{X}(t_*)) = 0$ from above or below. Hence, the current distance, at the time T, to the level crossing, denoted L, is $L = abs(G(\boldsymbol{X}(t_*)) - G(\boldsymbol{X}[T])) = abs(G(\boldsymbol{X}[T]))$. Now, we need to find an integration step d, such that $T + d = t_*$. This can be done using Eqs. (18) and (19) to get Eq. (20), where A and B are constants. Finally, Eq. (20) can be expanded into two quadratics indicating the drift of the continuous variables towards and away from the outgoing edge guards as shown in Eqs. (21) and (22).

The minimum of the solutions of these quadratic equations gives the step-size d that satisfies the outgoing edge guard. The pseudocode in Algorithm 4 computes the solution to the quadratics to find the step-size. However, just like in case of computing the step-size for the jump edge; this step-size might be too large. The continuous variable values at the returned step-size are bounded within the error bounds using Eq. (14) similar to computing the step-size for the jump edge (Algorithm 4, lines 11–15).

$$abs(G(\boldsymbol{X}(t_*)) - G(\boldsymbol{X}[T])) = abs(0 - G(\boldsymbol{X}[T])) = G(\boldsymbol{X}[T]) \tag{19}$$

$$\therefore G(\boldsymbol{X}(t_*)) - G(\boldsymbol{X}[T]) = \pm G(\boldsymbol{X}[T])$$

[3] In case of the running example, $abs(x^2(t_*) + y^2(t_*) - v^2) - \epsilon = 0$.

$\therefore$ Expressing $G(\boldsymbol{X}(t_*))$ using Eq. (18)

$$G(\boldsymbol{X}[T]) + (\sum_{i=1}^{n} c_i(f_i d + \sqrt{d/R}\gamma_i) + 1/2\sum_{i=1}^{n} c_i' g_i^2 d) - G(\boldsymbol{X}[T]) = \pm G(\boldsymbol{X}[T])$$

$$\therefore (K_1 d + K_2\sqrt{d}) + (K_3 d) + L = 0 \vee (K_1 d + K_2\sqrt{d}) + (K_3 d) - L = 0$$

$$\therefore Bd + A\sqrt{d} + L = 0 \vee Bd + A\sqrt{d} - L = 0 \tag{20}$$

$$B^2d^2 + (2BL - A^2)d + L^2 = 0 \tag{21}$$

$$B^2d^2 - (2BL + A^2)d + L^2 = 0 \tag{22}$$

```
Function guard_compute(guard expr ex, current variables X, current time T, Wiener paths dWts, location flow flow, constant R):
    $d \leftarrow 0$;                                                        /* The returning time step */
    $L \leftarrow abs(ex(X))$;                                                /* The level crossing */
    while true do
        /* Compute quadratic/linear formulations according to Equations (??) and (??). D and H are gradient and Hessian */
        /* Time step from $q_1$ */
        $d_1 \leftarrow compute\_quadratic\_soln(X, D, H, t, dWts, ex(X), d_1)$;
        /* Time step from $q_2$ */
        $d_2 \leftarrow compute\_quadratic\_soln(X, D, H, t, dWts, -ex(X), d_2)$;
        /* Get the minimum time-step from amongst the two */
        if $d_1 > 0 \wedge d_2 > 0$ then
            $d \leftarrow \min(d_1, d_2)$;
        else
            $d \leftarrow d_1 > 0?d_1 : d_2$;
        /* Compute the new value of continuous variables for step d using Eq. (5), with $\Upsilon = 1$, $\Delta = d$, and $\delta = d/R$ */
        $Xs \leftarrow \{\text{evolve}(x, dWts, flow[x], d, (d/R)) | x \in X\}$;
        /* Compute the continuous variables in half steps */
        $Xs' \leftarrow \{\text{evolve2}(x, dWts, flow[x], d, (d/R)) | x \in X\}$;
        if all error$(Xs, Xs') \leq \epsilon$ then                            // Check d satisfies Eq. (14)
            break
        L /= 2;                                                               /* Half the level crossing */
    return d
```

Algorithm 4: The `guard_compute` function pseudocode

In the next section we will prove two important properties of the step-size computation algorithms: ① existence of a real-positive step-size for both Algorithms 3 and 4; and ② convergence to the *first* stop time t_{**} and t_* without missing the level crossing.

5. Analysis of the algorithm

In this section we show that the proposed algorithms always result in a real positive step-size, which guarantees simulation progress, and the proposed simulation technique always detects the *first* instant when the discontinuities hold from any given location.

5.1. Simulation progress

Lemma 1. *Algorithm 3, and hence Eq. (13) always results in a real-positive integration/simulation step-size d.*

Proof. Eq. (13) computes the step-size for an outgoing jump edge. In this equation, $Uz_e \in \mathbb{R}^{>0}$ by Eq. (11). Similarly, $z_e[T] \in \mathbb{R}^{\geq 0}$ from Eqs. (11) and (12), $\forall T \in [T_k, t_{**})$. If, $z_e[T] - Uz_e$ is negative, while $-(z_e[T] - Uz_e)$ is positive; given that the value of $\lambda_e(\boldsymbol{X}[T])$ maybe positive or negative at instant T, d is always positive, by the disjunction in Eq. (13). Same for the case with $z_e[T] - Uz_e$ being positive and $-(z_e[T] - Uz_e)$ being negative. ■

Lemma 2. *Algorithm 4, and hence Eqs. (21)* and *(22) always give a real-positive integration/simulation step-size d.*

Proof. First we show that the discriminants $D1$ and $D2$ for Eqs. (21) and (22) are always non-negative, resulting in a real solution. $D1 \geq 0 \therefore (2BL - A^2)^2 - 4B^2L^2 \geq 0 \therefore A^2 - 4BL \geq 0$. Similarly, $D2 \geq 0 \therefore (2BL + A^2)^2 - 4B^2L^2 \geq 0 \therefore A^2 + 4BL \geq 0$. In either of these cases, $A^2 \in \mathbb{R}^{\geq 0}$ by definition of square. $L \in \mathbb{R}^{>0}$ by Algorithm 4 line 3. If B is positive then $A^2 + 4BL \geq 0$ holds, else $A^2 - 4BL \geq 0$ holds. Now, we show that we can always get a positive root for each of the non-negative discriminant cases. If B is positive, then Eq. (22) will give a real-root. It has two roots given by the standard solution to quadratic equation. For Eq. (22), the two solutions are: $\frac{(2BL+A^2)\pm\sqrt{D2}}{2B^2}$, with $B > 0$. Since $D2$, A^2, and L are all non-negative, we have a real-positive solution. Similar reasoning holds for Eq. (21), with $B < 0$. Finally, note that with $B = 0$, Eq. (21) will give a real-positive solution for a simple linear equation $A^2d = L^2$. ■

5.2. Convergence to the discontinuity

In Theorem 1 we prove that the simulation technique converges to the *first* stop time t_{**} (Definition 5) induced by a jump edge without overshooting it, within floating point bounds. To be precise, we prove that there exists a step Δ during numerical simulation from any point in time $T \in [T_{k-1}, t_{**})$, such that $T + \Delta = t_{**}$.

Theorem 1. *For a given SHA $\mathcal{H}$, following Definitions 4 and 5, there exists a stop time $t_{**} \in \mathbb{R}^{>0}$, such that, $\forall T \in [T_{k-1}, t_{**})$, $\int_{T_{k-1}}^{T} \lambda_{(v(T_{k-1}),v(t_{**}))}(\boldsymbol{X}(s))ds \nsim -\ln(\mathcal{U}(0,1))$. Then, there exists a step Δ, such that*
$\int_{T}^{T+\Delta} \lambda_{(v(T_{k-1}),v(t_{**}))}(\boldsymbol{X}(s))ds \sim -\ln(\mathcal{U}(0,1))$, and $T + \Delta = t_{**}$.

Proof. The proof is via induction. Base case: the step-size Δ is given by Eq. (13). If Algorithm 3 takes a single iteration, i.e., line 10 is satisfied then $T + \Delta = t_{**}$, by Eq. (12) and we have converged to the stop time. Inductive case: if the else branch (line 12) is taken, then the resultant solution of Eq. (13)$\Delta' < \Delta$, because from line 12, the numerator in Eq. (13) is reduced, but the denominator is constant for any given instant T. Moreover, at instant $T + \Delta'$, the jump condition is not satisfied. Finally, we set $T = T + \Delta' \in [T_{k-1}, t_{**})$, and then we need to show that $\int_{T}^{T+\Delta} \lambda_{(v(T_{k-1}),v(t_{**}))}(\boldsymbol{X}(s))ds \sim -\ln(\mathcal{U}(0,1))$, with $T + \Delta = t_{**}$, which is true by the inductive hypothesis. ■

In Theorem 2 we prove that the simulation technique converges to the stop time t_* (Definition 5) induced by a guard edge without overshooting it, within floating point bounds. To be precise, we prove that there exists a step Δ during numerical simulation from any point in time $T \in [T_{k-1}, t_*)$, such that $T + \Delta = t_*$.

Theorem 2. *For a given SHA $\mathcal{H}$, following Definitions 4 and 5, there exists a stop time $t_* \in \mathbb{R}^{>0}$, such that $\forall T \in [T_{k-1}, t_*)$, $\boldsymbol{X}[T]$ does not satisfy $guard_{(v(T_{k-1}),v(t_*))}$. Then, $\int_{T}^{T+\Delta} flow_{v(T_{k-1})}$ satisfies $guard_{(v(T_{k-1}),v(t_*))}$, i.e., our simulation technique can take an integration step Δ, such that the level crossing is satisfied and $T + \Delta = t_*$.*

Proof. This proof is via induction. Base case: the step-size Δ is given by the roots $\frac{(2BL+A^2)\pm\sqrt{D2}}{2B^2}$, $\frac{(2BL-A^2)\pm\sqrt{D1}}{2B^2}$ or $d = \frac{L^2}{A^2}$ for Eqs. (22), Eq. (21), respectively from Lemma 2. If Algorithm 4 takes a single iteration, i.e., line 13 is satisfied the very first time, then returned roots satisfy the guard condition, from Eq. (19). Inductive case: if the else branch (Algorithm 4 line 15) is taken then the resultant root $\Delta' < \Delta$, because in the aforementioned solutions, denominator is independent of L and B, A are constants for a given instant T. Moreover, at instant $T + \Delta'$, the guard condition is not satisfied (Eq. (19) is not satisfied). Hence, we now let $T = T + \Delta' \in [T_{k-1}, t_*)$, and we need to show that there exists a step Δ, such that $\int_{T}^{T+\Delta} flow_{v(T_{k-1})} \in guard_{(v(T_{k-1}),v(t_*))}$ with $T + \Delta = t_*$. This is true by the inductive hypothesis. ■

Theorems 1 and 2 can be trivially extended to one or more SHAs, because the simulation step-size is the minimum from amongst all the individual SHA step-sizes (Algorithm 1, line 15).

Remark 2. Given the convergence theorems above:

1. The expected stop times $\mathbb{E}(t_*)$ and $\mathbb{E}(t_{**})$ can be computed using the so called Monte-carlo inversion method [17].
2. For any hazard rate, the expected value of $\int_{T}^{t_{**}} \lambda_{(v(T_{k-1}),v(t_{**}))}(\boldsymbol{X}(s))ds$ is one (c.f. Proposition 4.1 [15]).
3. For a fixed simulation time $SIM\_END\_TIME > 0$, let $X_i(s)$ and $X_i[s]$ be the exact and numerical solution (obtained using the proposed technique) to a SDE defined in Eq. (6) for any $s \in [0, SIM\_END\_TIME]$. Then, the strong convergence $(\mathbb{E}[sup_{s\in[0,SIM\_END\_TIME]} \|(X_i(s) - X_i[s])\|^2])^{(1/2)}$ is bounded by $\sqrt{\Delta/R}$, where constant R is the length of the Wiener path for this step as defined in Section 4. This can be shown by taking Δ/R as the step size in Equation 3 and Theorem 4.1 in [18].

6. Benchmarking

In this section we present the experimental results showcasing the overall execution efficiency and the correctness of the proposed technique vis-a-vis Simulink/Stateflow®.

6.1. Benchmark description

In addition to the running example (Fig. 1) of the point UAV, we also include three distinct examples from published literature. The second example, after the UAV example, is the SHS modelling a SRGN from [15]. The SHA for SRGN is shown in Eqs. (23) and (24). There are two SHAs, corresponding to the two aforementioned equations, respectively. The first SHA has two locations, while the second has a single location. Variable $x(t)$ is shared between the two SHAs. The SRGN example is a jump process without continuous stochasticity, i.e., a zero Wiener process matrix. Moreover, variables $U1_z = -\ln(\mathcal{U}(0,1))$ and $U2_z = -\ln(\mathcal{U}(0,1))$ are drawn from the uniform distribution and $z(t)$ captures the Poisson jump

Table 1
Simulation set-up and speed-up results.

Benchmark	*SIM_END_TIME* (s)	Δ_s (s)	Simulink steps	Proposed steps	Speed-up steps	Simulink run-time (ms)	Proposed run-time (ms)	Run-time speed-up (%)
UAV	1.1	10^{-5}	110K	645	≈170×	2000	1146	42.7
SRGN	2000	10^{-1}	20K	2406	≈8.3×	1000	857	14.3
ARS	10	10^{-2}	2K	21	≈95×	1000	31	96.9
CF	20	10^{-2}	2K	36	≈55.5×	460	71	84.5

rate (λ). Finally, upon jumping from the first to the second location in Eq. (23), variable $x'(t) = x(t) - 1$, and the rate variable $z(t)$ is reset to zero upon changing any location.

$$[dx(t); dz(t)] = \begin{cases} [dt; 0.01x(t)dt], & z(t) \geq U1_z \\ [0dt; 0.001dt], & z(t) \geq U2_z \wedge x(t) \geq 1 \end{cases} \tag{23}$$

$$dx(t) = -0.01x(t)dt \tag{24}$$

The third example is the modified version of an ARS from [8]. This example has a single SHA with transcendental edge guards as shown in Eq. (25). This example models a non-chattering implementation of a sliding mode controller for the steering wheel position ($x(t)$) being at $\pi/2$ or $-\pi/2$. There are three locations in this SHA. Upon making a location change, reset $x'(t) = x(t)$. Chattering Zeno effects where infinite location switches are made in finite amount of time, can be introduced in *incorrect* SHS. In order to overcome these chattering effects, a practical solution is to introduce relaxing bounds (via ϵ) on edge guards forcing location switches and then introducing a new location that stops evolving the continuous variables when the switch happens. An example of non-chattering implementation is given in Eq. (25). The third location stops evolution of the continuous variable once the stable point is reached.

$$dx(t) = \begin{cases} -0.01dt + dW(t), & \cos(x(t)) < -\epsilon \\ 0.01dt - dW(t), & \cos(x(t)) > \epsilon \\ 0dt, & -\epsilon \leq \cos(x(t)) \leq \epsilon \end{cases} \tag{25}$$

The fourth and final example is a two CF with the second car (with position $x2(t)$ and velocity $v2(t)$) following the first car (with position $x1(t)$) under uncertainty, modelled as a Wiener stochastic process without jump, obtained from [19]. The SHA, with three locations, is shown in Eq. (26). The reset formulae carry over the values of the continuous variables.

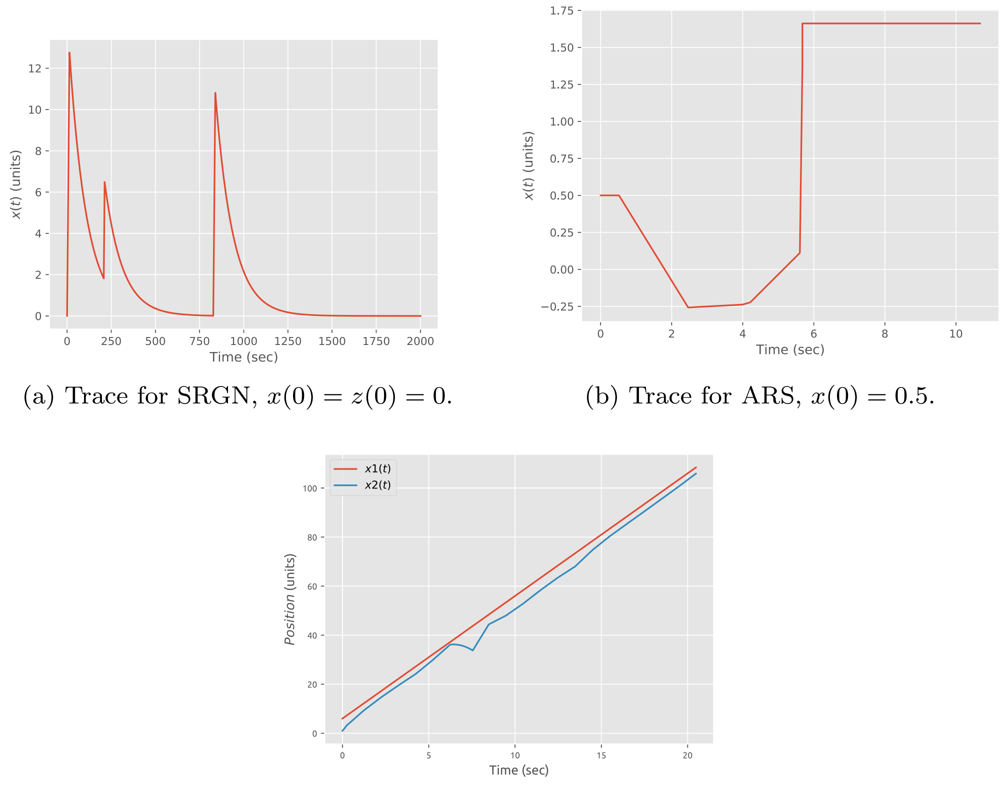
$$[dx1(t); dx2(t); dv2(t)] = \begin{cases} [5dt; 10dt + dW(t); 0], & |(x2(t) - x1(t) - 7)| \leq \epsilon \\ [5dt; 5dt + dW(t); 0], & |(x2(t) - x1(t) - 4)| \leq \epsilon \\ [5dt; v2(t)dt; -4dt], & |(x2(t) - x1(t) - 1)| \leq \epsilon \end{cases} \tag{26}$$

6.2. Experimental set-up and results

We run 1000 Monte Carlo runs for our benchmarks with varying Wiener/jump paths. We model the benchmarks in Simulink/Stateflow® version 2020a, with the help of the Mathworks® financial toolbox, and the proposed simulation engine is implemented in C++. For all examples, we set $\epsilon = 10^{-1}$ and $R = 2^3$, in Eq. (5), which indicates the length of the Wiener path for *each* integration/simulation step. All experiments are executed on OSX version 10.15.6 on a 2.9 GHz Intel® processor with 8 GB RAM. All code was compiled to binary using the GNU gcc/g++ compiler version 10.2 with the -O3 optimisation enabled. The simulation tool and benchmarks are available from [20] under an MIT license. The Simulink/Stateflow® implementation uses a fixed step EM solution with the *largest* step-size (Δ_s) that results in the correct output trace, without missing the discontinuities. The Simulink/Stateflow® benchmarks are also compiled into C/binary with optimisations enabled before execution.

6.2.1. Efficiency results

The output traces for a single Monte Carlo run and the speed-up obtained, over 1000 Monte Carlo runs, by the proposed technique are shown in Fig. 4 and Table 1, respectively. In Table 1, the column Δ_s gives the fixed step-size used for Simulink/Stateflow®. The proposed technique does not have a fixed step-size, rather it is a dynamic step-size integrator. Column 'Simulink steps' gives the number of simulation steps taken by Simulink/Stateflow®. Column 'Proposed steps' gives the average number of simulation steps taken by the proposed technique, over 1000 Monte Carlo runs, for each benchmark. The 'Speed-up steps' column indicates the average reduction in the number of simulation steps using the proposed technique. We see orders of magnitude speed-up from the proposed technique compared to Simulink/Stateflow®. Moreover, the proposed solution does not require a numerical solver, since there exist well-known solutions for both linear and quadratic equations (c.f. Section 5).



(a) Trace for SRGN, $x(0) = z(0) = 0$.

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

(c) Trace for CF, $x1(0) = 6, x2(0) = 1, v2(0) = 0.5$.

Fig. 4. Output traces for the SRGN, ARS, and CF benchmarks for single Monte Carlo run.

The efficiency of the proposed technique can also be seen in absolute run-time speed-up obtained compared to Simulink/Stateflow® as shown in columns 'Simulink rum-time (msec)' through to 'Run-time speed-up (%) '. On average, the proposed technique ≈ 48% faster in absolute execution time compared to Simulink/Stateflow® in our benchmarks. Overall, the proposed technique is well-suited for efficient simulation of complex SHS with non-linear and linear SDEs and discontinuities.

6.2.2. *Accuracy results*

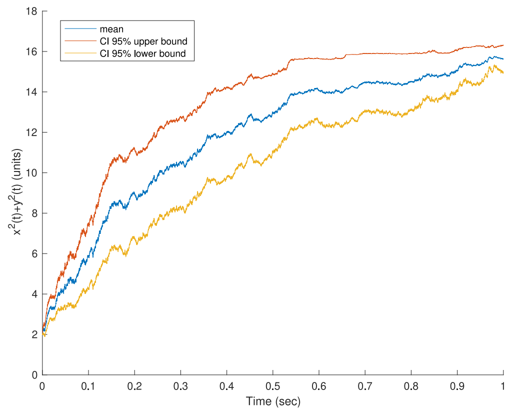
We use 95% Confidence Interval (CI) over 1000 Monte Carlo runs for the benchmarks to show the accuracy of the proposed technique. Here again, the Simulink/Stateflow® benchmarks were run using a fixed step-size EM solver with the same step-size (Δ_s) as given in Table 1. The confidence interval traces for the benchmarks are given in Figs. 5 and 6.

The CI for all benchmarks are pretty similar. The difference between the CIs from Simulink/Stateflow® and the proposed technique are due to the random number generator. The Simulink/Stateflow® uses the current time as the seed for the random number generator, whereas the proposed techniques uses the random device on hardware for the seed. Finally, note that for the CF benchmark we only give the CI for the second car, because the first car has no randomness (Eq. (26)) and hence the mean and the upper and lower 95% CIs are the same.

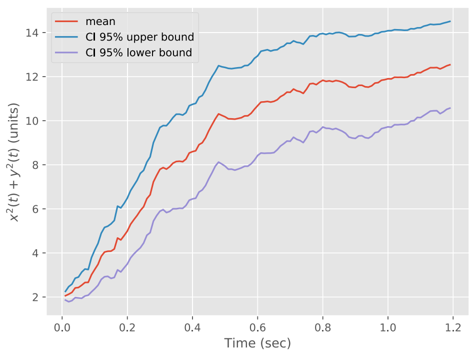
6.3. *Scalability of concurrent composition of SHAs*

This section presents the scalability results for the runtime composition of SHAs as described in Algorithm 1. We extend the CF example in Eq. (26) to a platoon of vehicles. The platoon of vehicles consists of the first vehicle moving without randomness (Eq. (27)). The first vehicle is captured in its own SHA. The other vehicles in the platoon, each modelled in its own SHA, follow their *leader* vehicle with uncertainty in the same vein as CF. The SHA model of each of the following vehicle is shown in Eq. (28), where *N* is the number of vehicles in the platoon. The number of vehicles in the platoon are increased from 2 all the way to 64, in powers of 2.

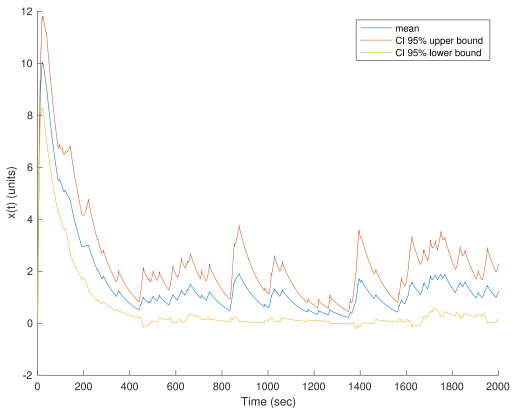
The first SHA has a single location with no outgoing edges (Eq. (27)). The rest of the SHAs have three locations each (Eq. (28)) with four edges. The first location has two outgoing edges to either locations one or three. There is an edge from



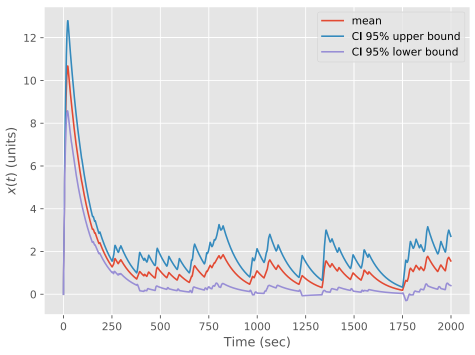
(a) Simulink/Stateflow®mean and 95% CI for UAV.



(b) Proposed technique mean and 95% CI for UAV.



(c) Simulink/Stateflow®mean and 95% CI for SRGN.



(d) Proposed technique mean and 95% CI for SRGN.

Fig. 5. Mean and confidence intervals for the benchmarks-1.

the second location to the first, and finally, there is an edge from the third location to the second. Moreover, there are 64 continuous variables ($x_i(t)$) representing the position of all the vehicles and 63 continuous variables ($v_i(t)$) representing the velocity of the followers.

$$dx_1(t) = 5dt \tag{27}$$

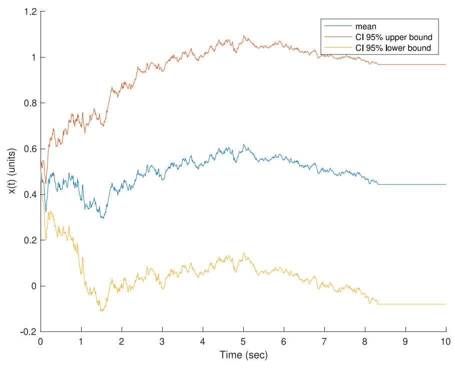
$$[dx_{i+1}(t), dv_{i+1}(t)] = \begin{cases} [10dt + dW_{i+1}(t); 0], & |(x_{i+1}(t) - x_i(t) - 7)| \leq \epsilon \\ [5dt + dW_{i+1}(t); 0], & |(x_{i+1}(t) - x_i(t) - 4)| \leq \epsilon \\ [v_{i+1}(t)dt; -4dt], & |(x_{i+1}(t) - x_i(t) - 1)| \leq \epsilon \end{cases}$$

$$\forall i \in \{1, N\} \tag{28}$$

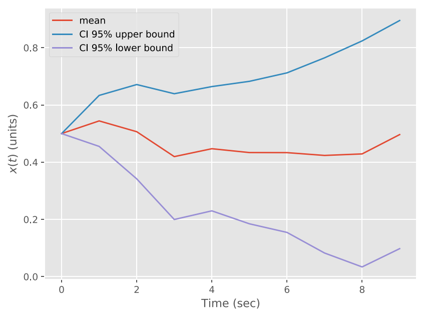
The platoon benchmark SHAs execute the model via the proposed technique. The runtime results with growing number of vehicles in the platoon in shown in Fig. 7. We can see that the execution time does **not** increase exponentially. This is in stark contrast to the well known state space explosion problem for standard flux/concurrent composition of SHAs and deterministic hybrid systems. There are two primary reasons for this scalability: ① Algorithm 1 does not build an explicit cross product of locations of all SHAs in the model. ② Given each SHA is in its own location; the algorithm needs to check level crossing for *only* the outgoing edges from those locations at runtime. There is no need to build a cross product (and the resultant conjunction) of outgoing edges from all the locations and check their level crossing conditions.

7. Related work

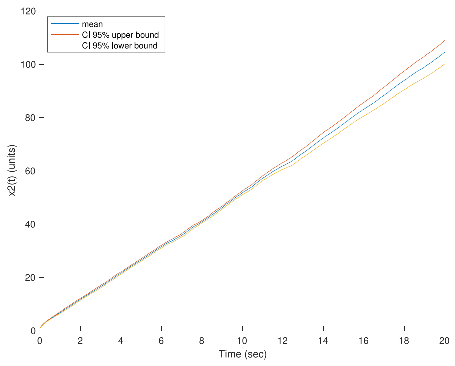
A number of adaptive step-size integration techniques have been proposed in the numerical analysis literature for simulating SDEs *without* discontinuous drift and diffusion [21–24]. These adaptive step-size integration techniques have



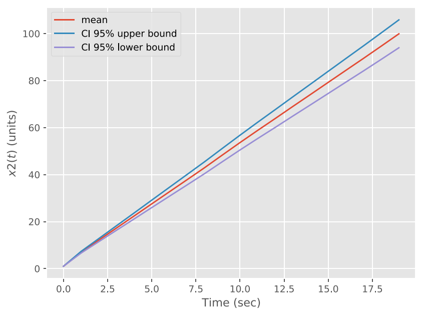
(e) Simulink/Stateflow®mean and 95% CI for ARS.



(f) Proposed technique mean and 95% CI for ARS.



(g) Simulink/Stateflow®mean and 95% CI for CF.



(h) Proposed technique mean and 95% CI for CF.

Fig. 6. Mean and confidence intervals for the benchmarks-2.

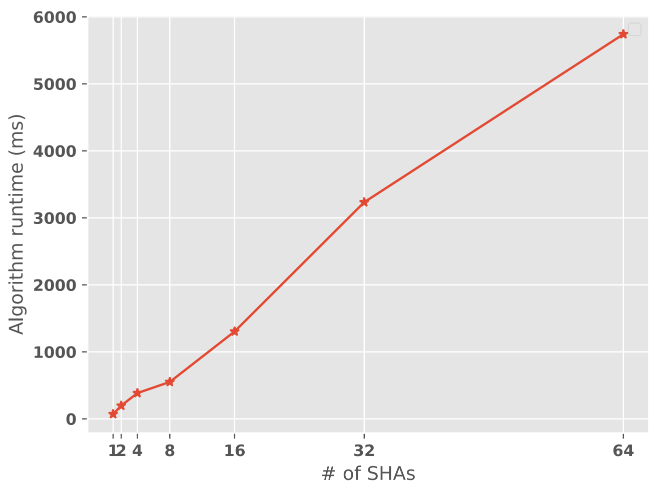


Fig. 7. Execution time of platoon model with increasing number of vehicles.

shown greatly improved simulation efficiency. Recently two different techniques have been proposed for adaptive step-size simulation of SDEs with discontinuous drift and diffusion [8,18]. The work in [18] proposes an integration function and its strong convergence given a user specified integration step. This is in contrast to the proposed work, where the integration step-size is *not* specified by the user. Rather, the integration step-size is computed using the SDEs and edge guards for a given network of SHAs. The proposal in [8] only considers linear time invariant SDEs with discontinuities for a single SHA. However, it does not address a network of SHAs, or non-linear and transcendental SDEs and edge guards. Similarly, the work in [8] also does not address Poisson jump edges/rates.

The verification and validation community has proposed a number of formal semantics and validation techniques for network of SHAs [2,16,19,25–28]. A multitude of these validation/verification techniques depend upon the EM solution for SDEs and Ito' lemma (Definition 3). The proposed work is complimentary to the work done by the validation/CPS community, in that, the proposed simulation techniques can be used for testing SHS and can form the basis for validation of SHA in the future. Finally, in terms of semantics; unlike current proposals, which perform a syntactic flux product [14], in this proposal we present a synchronous runtime flux semantics (Definition 6), which avoids state space explosion.

8. Conclusions and future work

In this paper we have proposed an efficient adaptive step-size simulation technique for a network of Stochastic Hybrid Automatons (SHAs). Given a network of SHAs, we propose to execute them concurrently and synchronously at the minimum from amongst their individual integration step-sizes. The individual integration step-sizes are computed by making the Stochastic Differential Equations and Poisson Jump rate integrators dependent upon the discontinuities. The proposed integration technique does not require a numerical solver. Ito' multi-dimensional theorem and the inverse sampling techniques are leveraged to compute the integration step-size using linear and quadratic equations only. In addition to existence and convergence analysis, experimental results also show orders of magnitude performance improvement compared to industry standard Simulink/Stateflow$^{®}$ for simulation of general Stochastic Hybrid Systems.

In the future we plan to improve the proposed algorithm to handle logical edge guards, e.g., guards with conjunction and disjunction.

CRediT authorship contribution statement

Avinash Malik: Conceptualization, Experimentation, Proofs, Drafting, Revising the manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. Alur, Principles of Cyber-Physical Systems, MIT Press, 2015.
[2] C.G. Cassandras, J. Lygeros, Stochastic Hybrid Systems, CRC Press, 2018.
[3] J.C. Jensen, D.H. Chang, E.A. Lee, A model-based design methodology for cyber-physical systems, in: 2011 7th International Wireless Communications and Mobile Computing Conference, IEEE, 2011, pp. 1666–1671.
[4] A. Malik, P. Roop, A dynamic quantized state system execution framework for hybrid automata, Nonlinear Anal. Hybrid Syst. 36 (2020) 100870.
[5] F. Zhang, M. Yeddanapudi, P.J. Mosterman, Zero-crossing location and detection algorithms for hybrid system simulation, IFAC Proc. Vol. 41 (2) (2008) 7967–7972.
[6] P.E. Kloeden, E. Platen, Numerical Solution of Stochastic Differential Equations, vol. 23, Springer Science & Business Media, 2013.
[7] J.-P. Laumond, S. Sekhavat, F. Lamiraux, Guidelines in nonholonomic motion planning for mobile robots, in: Robot Motion Planning and Control, Springer, 1998, pp. 1–53.
[8] A. Malik, Adaptive step size numerical integration for stochastic differential equations with discontinuous drift and diffusion, Numer. Algorithms (2020) 1–24.
[9] Z. Zhang, G. Karniadakis, Numerical Methods for Stochastic Partial Differential Equations with White Noise, vol. 196, Springer, 2017.
[10] J. Mø ller, H. Madsen, from State Dependent Diffusion To Constant Diffusion in Stochastic Differential Equations By the Lamperti Transform, in: IMM-Technical Report-2010-16, Technical University of Denmark, DTU Informatics, Building 321, 2010.
[11] J. Lamperti, A simple construction of certain diffusion processes, J. Math. Kyoto Univ. 4 (1) (1964) 161–170.
[12] K. Itô, 109. Stochastic integral, Proc. Imp. Acad. 20 (8) (1944) 519–524.
[13] J.M. Steele, Non-uniform random variate generation (Luc Devroye), 1987.
[14] L. Bortolussi, A. Policriti, Hybrid approximation of stochastic concurrent constraint programming, in: Proceedings of IFAC, vol. 2008, 2008.
[15] L. Bortolussi, A. Policriti, Hybrid dynamics of stochastic programs, Theoret. Comput. Sci. 411 (20) (2010) 2052–2077.
[16] M.L. Bujorianu, J. Lygeros, Theoretical foundations of stochastic hybrid systems, in: Mathematical Theory of Networks and Systems Conference, Leuwen, BG, 2004.
[17] D.J. Wilkinson, Stochastic Modelling for Systems Biology, Chapman and Hall/CRC, 2006.
[18] A. Neuenkirch, M. Szolgyenyi, L. Szpruch, An adaptive Euler–Maruyama scheme for stochastic differential equations with discontinuous drift and its convergence analysis, SIAM J. Numer. Anal. 57 (1) (2019) 378–403.
[19] J. Hu, J. Lygeros, S. Sastry, Towards a theory of stochastic hybrid systems, in: International Workshop on Hybrid Systems: Computation and Control, Springer, 2000, pp. 160–173.
[20] SHS benchmarks and simulation tool, 2021, https://github.com/amal029/eha/tree/working/cpp. (Accessed 18 July 2021).
[21] P.M. Burrage, K. Burrage, A variable stepsize implementation for stochastic differential equations, SIAM J. Sci. Comput. 24 (3) (2003) 848–864.
[22] J.G. Gaines, T.J. Lyons, Variable step size control in the numerical solution of stochastic differential equations, SIAM J. Appl. Math. 57 (5) (1997) 1455–1484.
[23] H. Lamba, An adaptive timestepping algorithm for stochastic differential equations, J. Comput. Appl. Math. 161 (2) (2003) 417–430.

[24] S. Ilie, K.R. Jackson, W.H. Enright, Adaptive time-stepping for the strong numerical solution of stochastic differential equations, Numer. Algorithms 68 (4) (2015) 791–812.
[25] J. Lygeros, M. Prandini, Stochastic hybrid systems: a powerful framework for complex, large scale applications, Eur. J. Control 16 (6) (2010) 583–594.
[26] A. Abate, J.-P. Katoen, J. Lygeros, M. Prandini, Approximate model checking of stochastic hybrid systems, Eur. J. Control 16 (6) (2010) 624–641.
[27] M. Fränzle, E.M. Hahn, H. Hermanns, N. Wolovick, L. Zhang, Measurability and safety verification for stochastic hybrid systems, in: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control, 2011, pp. 43–52.
[28] A. David, D. Du, K.G. Larsen, A. Legay, M. Mikučionis, D.B. Poulsen, S. Sedwards, Statistical model checking for stochastic hybrid systems, 2012, arXiv preprint arXiv:1208.3856.