Push-forward method for piecewise deterministic biochemical simulations

Guilherme C. P. Innocentini, Arran Hodgkinson, Fernando Antoneli, Arnaud Debussche, Ovidiu Radulescu

HAL Id: hal-03137189

https://hal.science/hal-03137189

Submitted on 15 Feb 2021

Push-forward method for piecewise deterministic biochemical simulations

Guilherme C.P. Innocentini

Universidade Federal do ABC, Santo André, Brazil

Arran Hodgkinson

Quantitative Biology and Medicine, University of Exeter, United Kingdom
Hodgkinson Laboratories Limited, Sheffield, United Kingdom

Fernando Antoneli

Escola Paulista de Medicina, Universidade Federal de São Paulo, São Paulo, Brazil

Arnaud Debussche

University of Rennes, CNRS, IRMAR - UMR 6625, F- 35000 Rennes, France

Ovidiu Radulescu

University of Montpellier, CNRS, LPHI - UMR CNRS 5235, Montpellier, France

Abstract

A biochemical network can be simulated by a set of ordinary differential equations (ODE) under well stirred reactor conditions, for large numbers of molecules, and frequent reactions. This is no longer a robust representation when some molecular species are in small numbers and reactions changing them are infrequent. In this case, discrete stochastic events trigger changes of the smooth deterministic dynamics of the biochemical network. Piecewise-deterministic Markov processes (PDMP) are well adapted for describing such situations. Although PDMP models are now well established in biology, these models remain computationally challenging. Previously we have introduced the push-forward method to compute how the probability measure is spread by the deterministic ODE flow of PDMPs, through the use of analytic expressions of the corresponding semigroup. In this paper we provide a more general simulation algorithm that works also for non-integrable systems. The method can be used for biochemical simulations with applications in fundamental biology, biotechnology and biocomputing. This work is an extended version of the work presented at the conference CMSB2019.

1. Introduction

Stochastic simulation is a powerful tool in biology. Moreover, stochasticity is a general property of biochemical networks, having multiple origins. Noise is generated intrinsically by these molecular systems when neither the law of large numbers nor the averaging theorem can be applied, for instance when some of the constituent molecular species are present in small numbers and trigger relatively slow reactions [5]. There are also extrinsic sources of noise, resulting from a fluctuating cellular environment. The extrinsic noise paradigm also applies to stochasticity resulting from manipulating biochemical networks in an artificial environment such as lab on a chip or simply in a titration device [1].

Stochastic simulation is used in single cell experimental studies, where the amounts of mRNA [26, 20, 4, 25] and protein [9, 10] products of a gene can be determined for each cell. By double- or multiple-fluorophore fluorescence techniques, products from several genes can be quantified simultaneously and one can have access to multivariate probability distributions of mRNA or proteins. The stochastic dynamics of promoters and gene networks can have important consequences for fundamental biology [8] but also for HIV [21] and cancer research [11]. Predicting the probability distributions of biochemical networks' products is also instrumental for lab-on-a-chip applications, when one wants to optimize and control the functioning of these networks. For these reasons we aim to develop effective methods for computing time-dependent distributions for stochastic models. Our main objective is the reduction of computation time which is prerequisite for parameter scans and machine learning applications [12].

The traditional stochastic simulation protocol uses the chemical master equation and the Gillespie algorithm. In such a protocol all the chemical reactions are simulated individually as discrete stochastic events. Simulations on relevant experimental times have to cope with $10^6 - 10^9$ such events and generate $10^2 - 10^4$ samples in order to get statistically significant estimates of molecular species distributions. Altogether, these simulations are extremely costly in terms of execution time.

An important reduction of the simulation time is obtained by noticing that the sources of noise can be localized in small parts of the system that behave discretely, or in the discrete environmental variables in the case of intrinsic or extrinsic noise, respectively. The remaining, large part of the system consists of molecular species in large numbers that evolve continuously. This leads to piecewise-deterministic Markov process (PDMP) approximations of the biochemical dynamics, coupling discrete state Markov chain dynamics with continuous state ordinary differential equations (ODE) dynamics. The justification of the PDMP approximations of the chemical master equation can be found in [5, 6]. Although simpler than the chemical master equation, direct simulation of the PDMP remains time consuming because the Markov chains can still have a very large number of states [5].

In the CMSB2019 proceedings paper we have introduced new methods for simulating PDMPs for gene networks [16]. A gene network PDMP model can be

simulated by numerical integration of ODEs satisfied by the mRNA and the protein species, coupled to the Markov chain describing the successive transitions of the gene promoters [28, 5, 22, 18]. The simulation becomes particularly effective when analytic solutions of the ODEs are available [14].

Probability distributions of PDMP are solutions of the Liouville-master partial differential equations (PDEs). Numerical integration of these PDEs is an interesting alternative to direct simulation, combining precision and speed for small models. Finite difference methods, however, are of limited use in this context as they can not cope with high dimensional models (for instance, extant gene networks applications are restricted to the dimension 2, corresponding to a single promoter, with or without self-regulation see [14, 17]).

Another interesting method for computing time dependent distributions is the push-forward method. For gene networks, this method has been first introduced in [13] and further adapted for continuous mRNA variables in [14]. It is based on the idea to compute the probability distribution of gene products as the push-forward measure of the semigroup defined by the ODEs. This method is an approximation, as one has to consider that the discrete PDMP variables are piecewise constant on a deterministic time partition. The transition rates between promoter states were computed using a mean field approximation in [14]. In the CMSB2019 proceeding paper, the mean field approximation was replaced by the next order approximation taking into account the second moment of the protein distribution [16]. In this paper we extend the method to general PDMP models, thus covering all biochemical networks, with both intrinsic and extrinsic noise simulation protocols.

2. Models

2.1. ODE models of biochemical networks

A biochemical network is defined by a set of chemical reactions and a set of chemical species. The amounts of different chemical species form a vector $x = (x_1, x_2, \ldots, x_N) \in \mathbb{R}_+^N$, where x_i is the concentration of the species i. Each reaction is characterized by a stoichiometric vector $\nu_i \in \mathbb{Z}^N$ and a reaction rate function $R_i : \mathbb{R}_+^N \to \mathbb{R}$. When all chemical species are present in large numbers, the biochemical network is well described by a systems of ODEs

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sum_{i=1}^{r} \nu_i R_i(x), \tag{1}$$

For example, a simple gene transcription model can be obtained when a large number, let's say G, of a specific gene is present in the cell and, moreover, the gene promoter has just two states: active and inactive. In this simple model the gene product is the mRNA concentration. The vector of species concentration is given by: $x_1 = P$ (inactive promoter concentration), $x_2 = P^*$ (active promoter concentration) and $x_3 = X$ (mRNA concentration). The concentration of inactive promoters is measured by the ratio between the number

3

of inactive promoters (G_I) and the cell volume V_{cell}, so $P = G_I/V_{\text{cell}}$; the same holds for the concentrations of active promoters and mRNA, $P = G_A/V_{\text{cell}}$, where G_A is the number of active promoters. The reactions are $P \to P^*$, $P^* \to P$, $P^* \to P^* + X$, $X \to \varnothing$. Thus, the stoichiometric vectors are $\nu_1 = (-1, 1, 0)$, $\nu_2 = (1, -1, 0)$, $\nu_3 = (0, 0, 1)$, $\nu_4 = (0, 0, -1)$ and reaction functions $R_1 = k_1 P$, $R_2 = k_2 P^*$, $R_3 = k_3 P^*$, $R_4 = k_4 X$, where the reaction rate constants k_i, $1 \le i \le 4$, have dimension [1/time]. For large copy numbers of P, P^* and X the simple transcription model reads:

$$\frac{\mathrm{d}P}{\mathrm{d}t} = -k_1 P + k_2 P^*, \ \frac{\mathrm{d}P^*}{\mathrm{d}t} = k_1 P - k_2 P^*, \ \frac{\mathrm{d}X}{\mathrm{d}t} = k_3 P^* - k_4 X . \tag{2}$$

Note that $\frac{\mathrm{d}P}{\mathrm{d}t} + \frac{\mathrm{d}P^*}{\mathrm{d}t} = 0$ and so one of the first two equations can be discarded.

We must emphasize that gene transcription is only one of the many possible examples of our formalism. The formalism defined by (1) covers all biochemical network models and have large applicability in cell physiology.

2.2. PDMP models

When some, but not all, species are present in low numbers and there are also slow reactions, a stochastic representation, such as a piecewise deterministic Markov process (PDMP), is more appropriate than the deterministic equations (1).

For instance, in the simple transcription model (2), assume that there is only one copy of the gene, meaning $G = G_A + G_I = 1$. In this case, it does not make sense to describe the time evolution of promoter variables by differential equations anymore. In this scenario, the promoter variables are discrete numbers. As the copy number of the gene is one, is more suitable to replace the concentration P by the discrete variable G_A assuming values $G_A = 1$ or $G_A = 0$. The same holds for P^*, that is replaced by $G_I \in \{0, 1\}$. Moreover, since $G_A + G_I = 1$ one of these variables can be discarded.

The switching between the two discrete states of G_A can be described by a continuous time Markov chain with transition rates k_1 and k_2. That is, if $G_A(0) = 1$, then $G_A(t) = 1$ for $0 \le t < T_1$ where T_1 is a random time such that $\mathbb{P}[T_1 > t] = \exp(-k_2 t)$ and $G_A(t) = 0$ for $T_1 \le t < T_1 + T_2$ where T_2 is a random time such that $\mathbb{P}[T_2 > t] = \exp(-k_1 t)$.

Moreover, suppose that the switching constants, k_1 and k_2, are small compared to k_4, and also that $k_3/k_4 \gg 1$. Thus, the variable X has a switching behavior alternating accumulation and degradation periods when $G_A = 1$ and $G_A = 0$, respectively, each one described by ODEs:

$$\frac{\mathrm{d}X}{\mathrm{d}t} = \begin{cases} k_3/V_{\text{cell}} - k_4 X & \text{when} \quad G_A = 1 \\ -k_4 X & \text{when} \quad G_A = 0 \end{cases}, \tag{3}$$

where V_{cell} is the cell volume.

Using methods from [6] it is possible to show that the "hybrid system" $(G_A/V_{\text{cell}}, X)$, with X given by (3), converges to (2), as $G \to \infty$.

The PDMP formalism can be extended to a rather general class of biochemical network models as follows:

A PDMP biochemical model is a stochastic process ζ_t having states in a set of the form $E = \mathbb{R}_+^N \times \mathcal{S}$, where $\mathcal{S}$ is a finite set encoding discrete states of the biochemical model, and $\zeta_t = (\mathbf{x}_t, s_t)$, where $\mathbf{x}_t$ is a vector in $\mathbb{R}_+^N$ whose components x_t^i ($1 \leq i \leq N$) encode the dynamics of i continuous biochemical species, and s_t describes the jump Markov process between the discrete states. The PDMP $\zeta_t = (\mathbf{x}_t, s_t)$ is determined by three characteristics:

1) For all fixed $s \in \mathcal{S}$, a vector field $\mathbf{V}_s : \mathbb{R}_+^N \to \mathbb{R}^N$ determining a unique global flow $\mathbf{\Phi}_s(t, \mathbf{x})$ in $\mathbb{R}_+^N$, such that, for $t > 0$,

$$\frac{\mathrm{d}\mathbf{\Phi}_s(t, \mathbf{x})}{\mathrm{d}t} = \mathbf{V}_s(\mathbf{\Phi}_s(t, \mathbf{x})), \ \mathbf{\Phi}_s(0, \mathbf{x}) = \mathbf{x}. \tag{4}$$

The flow $\mathbf{\Phi}_s(t, \mathbf{x})$ represents a one parameter semigroup fulfilling the properties

(i) $\mathbf{\Phi}_s(0, \mathbf{x}_0) = \mathbf{x}_0$,

(ii) $\mathbf{\Phi}_s(t + t', \mathbf{x}_0) = \mathbf{\Phi}_s(t', \mathbf{\Phi}_s(t, \mathbf{x}_0))$.

2) A transition rate matrix $\mathbf{H} : \mathbb{R}_+^N \to M_{N_s \times N_s}(\mathbb{R})$, such that $H_{r,s}(\mathbf{x})$ is the (r, s) element of the matrix $\mathbf{H}$, $N_s = \#\mathcal{S}$ is the number of states. If ($s \neq r$), $H_{r,s}(\mathbf{x})$ is the rate of probability to jump to the state r from the state s. Furthermore, $H_{s,s}(\mathbf{x}) = -\sum_{r \neq s} H_{r,s}(\mathbf{x})$ for all $s \in \mathcal{S}$ and for all $\mathbf{x} \in R^N$.

3) A jump rate $\lambda : E \to \mathbb{R}_+$. The jump rate can be obtained from the transition rate matrix

$$\lambda(\mathbf{x}, s) = \sum_{r \neq s} H_{r,s}(\mathbf{x}) = -H_{s,s}(\mathbf{x}). \tag{5}$$

From these characteristics, right-continuous sample paths $\{\mathbf{x}_t : t > 0\}$ starting at $\zeta_0 = (\mathbf{x}_0, s_0) \in E$ can be constructed as follows. Define

$$\mathbf{x}_t(\omega) := \mathbf{\Phi}_{s_0}(t, \mathbf{x}_0) \text{ for } 0 \leq t \leq T_1(\omega), \tag{6}$$

where $T_1(\omega)$ is a realisation of the first jump time of s_t, with the distribution

$$F(t) = \mathbb{P}[T_1 > t] = \exp\left(-\int_0^t \lambda(\mathbf{\Phi}_{s_0}(u, \mathbf{x}_0), s_0)\, du\right), \ t > 0, \tag{7}$$

and ω is the element of the probability space for which the particular realisation of the process is given. The pre-jump state is $\zeta_{T_1^-(\omega)}(\omega) = (\mathbf{\Phi}_{s_0}(T_1(\omega), \mathbf{x}_0), s_0)$ and the post-jump state is $\zeta_{T_1(\omega)}(\omega) = (\mathbf{\Phi}_{s_0}(T_1(\omega), \mathbf{x}_0), s)$, where s has the distribution

$$\mathbb{P}[s = r] = \frac{H_{r,s_0}(\mathbf{\Phi}_{s_0}(T_1(\omega), \mathbf{x}_0), s_0)}{\lambda(\mathbf{\Phi}_{s_0}(T_1(\omega), \mathbf{x}_0), s_0)}, \text{ for all } r \neq s_0. \tag{8}$$

We then restart the process $\zeta_{T_1(\omega)}$ and recursively apply the same procedure at jump times $T_2(\omega)$, etc..

Note that between each two consecutive jumps $\mathbf{x}_t$ follow deterministic ODE dynamics defined by the vector field $\mathbf{V}_s$. At the jumps, the values $\mathbf{x}_t$ are continuous. More general definitions of PDMPs include jumps in the continuous variables but will not be discussed here.

We define multivariate probability density functions $p_s(t, \mathbf{x})$. These functions satisfy the Liouville-master equation which is a system of partial differential equations:

$$\frac{\partial p_s(t, \mathbf{x})}{\partial t} = -\nabla_{\mathbf{x}}.(\mathbf{V}_s(\mathbf{x})p_s(t, \mathbf{x})) + \sum_{s'} H_{s,s'}(\mathbf{x})p_{s'}(t, \mathbf{x}). \tag{9}$$

This general formalism covers all biochemical models that have discrete variables. As salient examples we can cite gene networks [14], ion channels dynamics for neuron networks [2], lab on chip biochemical devices [1].

2.3. PDMP models of ON/OFF gene networks

A particular example of PDMP biochemical model is represented by the gene networks. Because these models are extensively used in systems biology, we provide their details in this subsection.

Two state ON/OFF gene networks generalize the simple two state single gene transcription model, by considering more interacting genes and by using separate variables for the two gene products: mRNA and protein.

To each gene we associate a discrete variable σ_i with two possible values $\sigma_i = 0$ for a non-productive state (OFF) and $\sigma_i = 1$ for a productive state (ON). Furthermore, a gene i produces proteins and mRNAs in the amounts y_i and r_i, respectively. A gene network state is described by the $N-$dimensional vector $x_t = (r_1(t), y_1(t), r_2(t), y_2(t), \ldots, r_{N/2}(t), y_{N/2}(t))$. The gene products amounts satisfy ODEs:

$$\begin{aligned}
\frac{\mathrm{d}y_i}{\mathrm{d}t} &= b_i r_i - a_i y_i, \\
\frac{\mathrm{d}r_i}{\mathrm{d}t} &= k_i(\sigma_i) - \rho_i r_i
\end{aligned} \tag{10}$$

The coupling between genes is modeled at the level of discrete state transitions. The elements of the matrix $\mathbf{H}$ are functions of products from various genes. For instance, if a gene i inhibits a gene j the transition rate from ON to OFF of the gene j is an increasing function of the protein concentration y_i.

As a first example that we denote as model M_1, let us consider a two genes network; the expression of the first gene being constitutive and the expression of the second gene being activated by the first. We consider that the transcription activation rate of the second gene is proportional to the concentration of the first protein $f_2 y_1$. All the other rates are constant f_1, h_1, h_2, representing the transcription activation rate of the first gene, and the transcription inactivation rates of gene one and gene two, respectively. For simplicity, we consider that

6

the two genes have identical protein and mRNA parameters $b_1 = b_2 = b$, $a_1 = a_2 = a$, $\rho_1 = \rho_2 = \rho$. We further consider that $k_i = k_0$ if the gene i is OFF and $k_i = k_1 > k_0$ if the gene i is ON.

The gene network has four discrete states, in order $(0,0)$, $(1,0)$, $(0,1)$, and $(1,1)$. Then, the transition rate matrix for the model M_1 is

$$\mathbf{H}(y_1, y_2) = \begin{bmatrix} -(f_1 + f_2 y_1) & h_1 & h_2 & 0 \\ f_1 & -(h_1 + f_2 y_1) & 0 & h_2 \\ f_2 y_1 & 0 & -(f_1 + h_2) & h_1 \\ 0 & f_2 y_1 & f_1 & -(h_1 + h_2) \end{bmatrix}. \quad (11)$$

The Liouville-master equation for the model M_1 reads

$$\frac{\partial p_1}{\partial t} = -\frac{\partial[(br_1 - ay_1)p_1]}{\partial y_1} - \frac{\partial[(k_0 - \rho r_1)p_1]}{\partial r_1} - \frac{\partial[(br_2 - ay_2)p_1]}{\partial y_2} - \frac{\partial[(k_0 - \rho r_2)p_1]}{\partial r_2}$$
$$+ h_2 p_3 + h_1 p_2 - (f_1 + f_2 y_1)p_1,$$

$$\frac{\partial p_2}{\partial t} = -\frac{\partial[(br_1 - ay_1)p_2]}{\partial y_1} - \frac{\partial[(k_1 - \rho r_1)p_2]}{\partial r_1} - \frac{\partial[(br_2 - ay_2)p_2]}{\partial y_2} - \frac{\partial[(k_0 - \rho r_2)p_2]}{\partial r_2}$$
$$+ f_1 p_1 + h_2 p_4 - (h_1 + f_2 y_1)p_2,$$

$$\frac{\partial p_3}{\partial t} = -\frac{\partial[(br_1 - ay_1)p_3]}{\partial y_1} - \frac{\partial[(k_0 - \rho r_1)p_3]}{\partial r_1} - \frac{\partial[(br_2 - ay_2)p_3]}{\partial y_2} - \frac{\partial[(k_1 - \rho r_2)p_3]}{\partial r_2}$$
$$+ h_1 p_4 + f_2 y_1 p_1 - (h_2 + f_1)p_3,$$

$$\frac{\partial p_4}{\partial t} = -\frac{\partial[(br_1 - ay_1)p_4]}{\partial y_1} - \frac{\partial[(k_1 - \rho r_1)p_4]}{\partial r_1} - \frac{\partial[(br_2 - ay_2)p_4]}{\partial y_2} - \frac{\partial[(k_1 - \rho r_2)p_4]}{\partial r_2}$$
$$+ f_1 p_3 + f_2 y_1 p_2 - (h_1 + h_2)p_4.$$
$$(12)$$

The model M_2 differs from the model M_1 by the form of the activation function. Instead of a linear transcription rate $f_2 y_1$ we use a Michaelis-Menten model $f_2 y_1 / (K_1 + y_1)$. This model is more realistic as it takes into account that the protein p^1 has to attach to specific promoter sites which become saturated when the concentration of this protein is high.

The transition rate matrix for the model M_2 is

$$\mathbf{H}(y_1, y_2) = \begin{bmatrix} -(f_1 + \frac{f_2 y_1}{K_1 + y_1}) & h_1 & h_2 & 0 \\ f_1 & -(h_1 + \frac{f_2 y_1}{K_1 + y_1}) & 0 & h_2 \\ \frac{f_2 y_1}{K_1 + y_1} & 0 & -(f_1 + h_2) & h_1 \\ 0 & \frac{f_2 y_1}{(K_1 + y_1)} & f_1 & -(h_1 + h_2) \end{bmatrix}. \quad (13)$$

The Liouville-master equation for the model M_2 reads

$$\frac{\partial p_1}{\partial t} = -\frac{\partial[(br_1 - ay_1)p_1]}{\partial y_1} - \frac{\partial[(k_0 - \rho r_1)p_1]}{\partial r_1} - \frac{\partial[(br_2 - ay_2)p_1]}{\partial y_2} - \frac{\partial[(k_0 - \rho r_2)p_1]}{\partial r_2}$$
$$+ h_2 p_3 + h_1 p_2 - (f_1 + f_2 y_1/(K_1 + y_1))p_1,$$

$$\frac{\partial p_2}{\partial t} = -\frac{\partial[(br_1 - ay_1)p_2]}{\partial y_1} - \frac{\partial[(k_1 - \rho r_1)p_2]}{\partial r_1} - \frac{\partial[(br_2 - ay_2)p_2]}{\partial y_2} - \frac{\partial[(k_0 - \rho r_2)p_2]}{\partial r_2}$$
$$+ f_1 p_1 + h_2 p_4 - (h_1 + f_2 y_1/(K_1 + y_1))p_2,$$

$$\frac{\partial p_3}{\partial t} = -\frac{\partial[(br_1 - ay_1)p_3]}{\partial y_1} - \frac{\partial[(k_0 - \rho r_1)p_3]}{\partial r_1} - \frac{\partial[(br_2 - ay_2)p_3]}{\partial y_2} - \frac{\partial[(k_1 - \rho r_2)p_3]}{\partial r_2}$$
$$+ h_1 p_4 + f_2 y_1/(K_1 + y_1)p_1 - (h_2 + f_1)p_3,$$

$$\frac{\partial p_4}{\partial t} = -\frac{\partial[(br_1 - ay_1)p_4]}{\partial y_1} - \frac{\partial[(k_1 - \rho r_1)p_4]}{\partial r_1} - \frac{\partial[(br_2 - ay_2)p_4]}{\partial y_2} - \frac{\partial[(k_1 - \rho r_2)p_4]}{\partial r_2}$$
$$+ f_1 p_3 + f_2 y_1/(K_1 + y_1)p_2 - (h_1 + h_2)p_4.$$

3. Simulation methods

3.1. Monte-Carlo method

The Monte-Carlo method utilizes the direct simulation of the PDMP based on the iteration of the following equations:

$$\frac{d\mathbf{x}}{dt} = \mathbf{V}_{s_0}(\mathbf{x}(t)),$$
$$\frac{dF}{dt} = -\lambda(\mathbf{x}, s_0)F, \tag{14}$$

for $t \in [0, T_1)$ with initial conditions $\mathbf{x}(0) = \mathbf{x}_0$, $F(0) = 0$, and stopping condition $F(T_1) = U$, where U is random variable, uniform in the range $[0, 1)$, followed by the choice of the next discrete state s_1 by using the matrix $\mathbf{H}(\mathbf{x}(T_1))$.

A large number N_{mc} of sample traces is generated and the values of $\mathbf{x}_t$ are stored at selected times. Multivariate time dependent probability distributions are then estimated from this data.

The complete algorithm is presented in the Algorithms 1 and 2.

The direct simulation of PDMPs needs the solutions of Eqs. (14) which can be obtained by numerical integration. This is not always computationally easy. Problems may arise for fast switching promoters when the ODEs have to be integrated many times on small intervals between successive jumps. Alternatively, the numerical integration of the ODEs can be replaced by analytic solutions or quadrature. Analytic expressions are always available for the gene network flow

8

Algorithm 1 NextState

Input: 1. x initial value continuous variable, 2. s initial discrete state, 3. t_0 initial time, 4. t_{max} maximal time

Output: 1. $(x_0, x_1, \ldots, x_n)$, 2. $(t_0, t_1, \ldots, t_n)$ 3. s new discrete state

1: $x_0 := x$; $s_0 := s$
2: $U := randunif([0, 1))$
3: solve $\frac{\mathrm{d}x}{\mathrm{d}t} = V_{s_0}(x)$, $\frac{\mathrm{d}F}{\mathrm{d}t} = -\lambda(x, s_0)F$ with initial data $x(t_0) = x_0$, $F(t_0) = 1$, until $F(t) = U$ or $t = t_{max}$, return $x_0 = x(t_0), \ldots, x_n = x(t_n)$, where $t_0 < t_1 < \ldots < t_n$, $t_n = \min(F^{-1}(U), t_{max})$.
4: $V := randunif([0, 1))$
5: $CUM := 0$; $I := 0$
6: **while** $CUM < V$ **do**
7: $\quad I := I + 1$
8: $\quad CUM := CUM + H_{I,s_0}(x) / \sum_{J \neq s_0} H_{J,s_0}(x)$
9: **end while**
10: **return** $((x_0, x_1, \ldots, x_n), (t_0, t_1, \ldots, t_n), I)$

---

(10) and read

$$\Phi_i^y(t, y_0, r_0) = y_0 \exp(-a_i t)$$
$$+ b_i \left[\left(r_0 - \frac{k_i(\sigma_i)}{\rho_i} \right) \frac{\exp(-\rho_i t) - 1}{a_i - \rho_i} + \frac{k_i(\sigma_i)}{\rho_i} \frac{1 - \exp(-a_i t)}{a_i} \right],$$

$$\Phi_i^r(t, y_0, r_0) = (r_0 - k_i/\rho_i) \exp(-\rho_i t) + k_i/\rho_i.$$

Let us consider the following general expression of the jump intensity function

$$\lambda(\mathbf{x}, s) = c_0(s) + \sum_i^N c_i(s) y_i + \sum_i^N d_i(s) f_i(y_i),$$

where f_i are non-linear functions, for instance Michaelis-Menten function

$$f_i(p^i) = \frac{p^i}{K_i + p^i}$$

or Hill functions

$$f_i(y_i) = \frac{(y_i)^{n_i}}{K_i^{n_i} + (y_i)^{n_i}}.$$

If $d_i = 0$ for all $1 \leq i \leq N$, the cumulative distribution function of the waiting time T_1 can be solved analytically [14], otherwise it can be obtained by quadrature. For example, for the model M_2 one has

$$\lambda(\mathbf{x}, s) = \left(f_1 + f_2 \frac{y_1}{K_1 + y_1} \right) \delta_{s,1} + \left(h_1 + f_2 \frac{y_1}{K_1 + y_1} \right) \delta_{s,2}$$
$$+ (h_2 + f_1) \delta_{s,3} + (h_2 + h_1) \delta_{s,4},$$

Algorithm 2 PDMPmontecarlo

Input: 1. $p_0(x, s)$, initial distribution of (x_0, s_0). 2. τ time step for computing the time-dependent distribution. 3. δx bin size in one direction.

Output: n_t vectors $H_0, \ldots, H_{nt}$,
    representing the distribution at times $0, \tau, \ldots, n_t\tau$

1: **for** $i := 1$ **to** n_t **do**
2:    $H_i := n_x^M$ dimensional null vector
3: **end for**
4: **for** $m := 1$ **to** N_{mc} **do**
5:    Draw (x_0, s_0) from the distribution $p_0(x, s)$
6:    $x := x_0, s := s_0$
7:    $t := t_0; i := 0$
8:    **while** $t < n_t\tau$ **do**
9:       $((x_0, \ldots, x_n), (t_0, \ldots, t_n), s_{next}) := \text{NextState}(x, s, t, t_M)$
10:      **for** $j := 1$ **to** n **do**
11:         **if** $t_j \leq i\tau < t_{j+1}$ **then**
12:           $X_i := (x_j + x_{j+1})/2$
13:           increment by $1/((\delta x)^N N_{mc})$,
              the bin corresponding to X_i in the histogram H_i
14:           $i := i + 1$
15:         **end if**
16:      **end for**
17:      $x := x_n$
18:      $s := s_{next}$
19:      $t := t_n$
20:      $nr := nr + 1$
21:    **end while**
22: **end for**
23: **return** $(H_0, \ldots, H_{n_t})$

---

where $\delta_{i,j}$ is Kronecker's delta. In this case, the waiting time T_1 is obtained as the unique solution of the equation

$$
\begin{aligned}
-\log(U) = & \left[(f_1 + f_2)T_1 + f_2 \int_0^{T_1} \frac{1}{K_1 + \Phi_1^y(t', y_0, r_0)} \, dt' \right] \delta_{s_0,1} \\
& + \left[(h_1 + f_2)T_1 + f_2 \int_0^{T_1} \frac{1}{K_1 + \Phi_1^y(t', y_0, r_0)} \, dt' \right] \delta_{s_0,2} \\
& + (h_2 + f_1)T_1 \delta_{s_0,3} + (h_2 + h_1)T_1 \delta_{s_0,4},
\end{aligned}
\tag{15}
$$

where U is a random variable, uniformly distributed in the range $[0, 1)$. In our implementation of the algorithm we solve (15) numerically, using the bisection method.

3.2. Finite difference Liouville master equation method

The finite difference Liouville master equation method for a given number of genes, N_g, uses a discrete approximation of the domain to compute the numerical solution across time for a given higher-dimensional system (12), with initial conditions given by $p_1(0, y_1, r_1, \ldots) := \delta(y_1, r_1, \ldots)$ and $p_2(0, r_1, y_1, \ldots) := \ldots := p_{2N_g}(0, y_1, r_1, \ldots) := 0$ where $\delta(\cdot)$ is the Dirac delta function. We compute the solution for $2N_g$ distributions, since each gene has both an ON and OFF state, respectively.

In order to achieve the simulation of the equations given by system (12), we begin by discretising each of the domains into n_η intervals, whose centres are given by $\{\eta_1, \ldots, \eta_{n_\eta}\} \in \mathbf{y}$ meeting the condition that $\eta_1 < \eta_2 < \ldots < \eta_{n_\eta}$ and where, likewise, an arbitrary discrete value in y_i would be denoted $\eta_{i,j}$. Each $\eta i, j$ then represents a unique, discrete, j^{th} position in the protein abundance domain spanned by y_i. Likewise, mRNA would have an analogous discretisation given by all $\rho_{i,j}, \forall j \in \{1, \ldots, n_\rho\}$. Time is then similarly discretised by a time-step τ into $n_t + 1$ temporal locations given by $\{0, \tau, \ldots, \tau n_t\}$. The task then becomes the computation of the solution at each of these discrete locations in the domain, such that the sink and source terms may be trivially calculated but where the derivative terms warrant further explanation.

The solution to the advection equation under a uniform coefficient, χ, is given by a translation in the relevant domain. To achieve this, whilst also maintaining the stability of the system, as a whole, we implement a simultaneous forwards- and backwards-difference discrete operator scheme. This means that for an arbitrary probability density function, $p_k(t, y_i, \ldots)$, we write the discrete partial derivative operator as

$$\chi \frac{\partial}{\partial y_i} p_k(t, \eta_{i,j}, \ldots) = \begin{cases} |\chi| \dfrac{p_k(t, \eta_{i,j-1}, \ldots) - p_k(t, \eta_{i,j}, \ldots)}{\eta_{i,j} - \eta_{i,j-1}} & \text{if } \chi > 0 \\ |\chi| \dfrac{p_k(t, \eta_{i,j+1}, \ldots) - p_k(t, \eta_{i,j}, \ldots)}{\eta_{i,j+1} - \eta_{i,j}} & \text{otherwise.} \end{cases}$$

$$(16)$$

This may be evaluated as such for each term within the system of equations, given by (12), and guarantees the probability balance of the system, as a whole.

Beyond the computation of the equation's solutions at a single time-step the solutions must be computed robustly across time. We therefore employ a McCormack predictor-corrector scheme [19], given explicitly for any i^{th} time-step by

$$p'_{k,i+1} = p_{k,i} + \tau F_k(p_{1,i}, \ldots, p_{N_g,i})$$
$$p_{k,i+1} = \frac{1}{2}(p_{k,i} + p'_{k,i+1}) + \frac{1}{2}\tau F_k(p'_{1,i+1}, \ldots, p'_{N_g,i+1})$$
$$\forall k \in \{1, \ldots, 2N_g\},$$
$$(17)$$

where $p_{k,i} = p_k(t_i, y_1, r_1, \ldots)$, $F_k(p_1, \ldots, p_{2N_g}) = \partial p_k/\partial t$ and $p'_{k,i}$ is the discrete nomenclature for a prediction of the solution for p_k at time t_i.

11

The method for solving the problem, in totality, is then given by Algorithm 3 where, again, the partial derivative terms on the right-hand side of each equation are evaluated using (16). Algorithm 3 is a direct implementation of (17) as an algorithm with the concurrent calculation of the distributions P_t, $\forall t \in \{0, \tau, \ldots, n_t\tau\}$ from the individual distributions $p_{k,i}$, $\forall k \in \{1, \ldots, 2N_g\}$, $i \in \{0, \ldots, n_t\}$. For each time-step, we solve the right-hand side of the equation using the predictor corrector scheme, update the value of each distribution, and calculate the total distribution, P_t.

Algorithm 3 PDMPLiouvillemaster

Input: 1. $p_{1,0}, p_{2,0}, p_{3,0}, p_{4,0}$ initial distributions in $(y_1, r_1, \ldots, y_{N_g}, r_{N_g})$, where $p_{i,t} := p_i(t, y_1, r_1, \ldots, y_{N_g}, r_{N_g})$, $\forall i \in \{1, \ldots, 2N_g\}$. 2. τ time step for computing the time-dependent distribution. 3. δy, δr bin size for the protein and mRNA distributions, respectively.

Output: $n_t + 1$ distributions $P_0, \ldots, P_{n_t}$ representing the sums of the individual distributions in (y_1, r_1, y_2, r_2) at times $0, \tau, \ldots, n_t\tau$, respectively

1: **for** $i := 1$ **to** n_t **do**
2:    **for** $j := 1$ **to** $2N_g$ **do**
3:       Use $p_{k,i-1}$ and discretised right hand side of PDE in (12) to compute the temporal gradient, $\partial_t p_{j,i-1}$
4:       Compute predicted solution at next time step as $p'_{j,i} = p_{j,i-1} + \tau\partial_t p_{j,i-1}$
5:    **end for**
6:    Set $P_i = 0$
7:    **for** $j := 1$ **to** $2N_g$ **do**
8:       Use prediction $p'_{k,i}$ and discretised right hand side of PDE in 12 to compute the temporal gradient, $\partial_t p'_{j,i}$
9:       Compute corrected solution at next time step as
        $p_{j,i} = \frac{1}{2}(p_{j,i-1} + p'_{j,i}) + \frac{1}{2}\tau\partial_t p'_{j,i}$
10:      Set $P_i = P_i + p_{j,i}$
11:   **end for**
12: **end for**
13: **return** $(P_0, \ldots, P_{n_t})$

3.3. Push-forward method

3.3.1. General algorithm

This method allows one to compute the multivariate probability distribution of the continuous variable $\mathbf{x}$ at a time τ given the probability distribution of $(\mathbf{x}, s)$ at time 0.

In order to achieve this we consider a deterministic partition $\tau_0 = 0 < \tau_1 < \ldots < \tau_M = \tau$ of the interval $[0, \tau]$ such that $\Delta_M = \max_{j\in[1,M]}(\tau_j - \tau_{j-1})$ is small. The main approximation of this method is to assume that s_t, for $t \in [0, \tau]$, is piecewise constant on this partition, more precisely, that $s_t = s_j := s_{\tau_j}$, for $t \in [\tau_j, \tau_{j+1})$, $0 \leq j \leq M - 1$. This is rigorously true for intervals

Algorithm 4 PDMPpushforward

Input: 1. $p_0(x, s)$, initial distribution of (x_0, s_0). 2. τ time step for computing the time-dependent distribution

Output: n_t vectors $H_0, \ldots, H_{nt}$,
representing the distribution at times $0, \tau, \ldots, n_t\tau$

1: compute $H_0, P0$ from initial distribution
2: **for** $i := 1$ **to** n_t **do**
3:   **for** $j := 1$ **to** n_x^N **do**
4:     **for** $s0 := 1$ **to** N_s **do**
5:       solve $\frac{d\mathbf{x}0}{dt} = V_{s_0}(\mathbf{x}0), \frac{d\mathbf{\Pi}0}{dt} = H(\mathbf{x}0)\mathbf{\Pi}0$ with initial conditions $\mathbf{x}0(0) = C_j, \Pi0(0) = \mathbf{I}$ from $t = 0$ to $t = \tau_1$
6:       **for** $s1 := 1$ **to** N_s **do**
7:         solve $\frac{d\mathbf{x}1}{dt} = V_{s_1}(\mathbf{x}1), \frac{d\mathbf{\Pi}1}{dt} = H(\mathbf{x}1)\mathbf{\Pi}1$ with initial conditions $x1(\tau_1) = x0(\tau_1), \Pi1(0) = \mathbf{I}$ from $t = \tau_1$ to $t = \tau_2$
8:         **for** $s2 := 1$ **to** N_s **do**
9:           solve $\frac{d\mathbf{x}2}{dt} = V_{s_2}(\mathbf{x}2), \frac{d\mathbf{\Pi}2}{dt} = H(\mathbf{x}2)\mathbf{\Pi}2$ with initial conditions $x2(\tau_2) = x1(\tau_2), \Pi2(0) = \mathbf{I}$ from $t = \tau_2$ to $t = \tau_3$
10:           **for** $s3 := 1$ **to** N_s **do**
11:             solve $\frac{d\mathbf{x}3}{dt} = V_{s_3}(\mathbf{x}3)$ with initial conditions $x3(\tau_3) = x2(\tau_3)$ from $t = \tau_3$ to $t = \tau_4$
12:             $P_M = \Pi2_{s3,s2}\Pi1_{s2,s1}\Pi0_{s1,s0}P0_{s0}$
13:             $\mathbf{x} = \mathbf{x}3(\tau_4)$
14:             $H_i(BIN(\mathbf{x})) = H_i(BIN(\mathbf{x})) + P_M H_{i-1}(j)$ $BIN(\mathbf{x}) : bin$ $containing$ $\mathbf{x}$
15:           **end for**
16:         **end for**
17:       **end for**
18:     **end for**
19:   **end for**
20: **end for**

$[\tau_j, \tau_{j+1})$ completely contained between two successive random jump times of s_t. This situation becomes very frequent for a very fine partition (large M). Thus, the error generated by the approximation vanishes in the limit $M \to \infty$ (the rigorous result is Theorem 1 given in the Results section).

For each path realization $S_M := (s_0, s_1, \ldots, s_{M-1}) \in \Omega := \mathcal{S}^M$ of the discrete states, we can compute $\mathbf{x}(t)$, $t \in [0, \tau]$ as the continuous solution of the following ODE with piecewise-defined r.h.s:

$$\frac{d\mathbf{x}}{dt} = \mathbf{V}_{s_j}(\mathbf{x}), \text{for } t \in [\tau_j, \tau_{j+1}), 0 \le j \le M - 1 \tag{18}$$

and with the initial condition $\mathbf{x}(0) = \mathbf{x}_0$.

In order to compute the probability $\mathbb{P}[S_M]$ of a path realization we can use
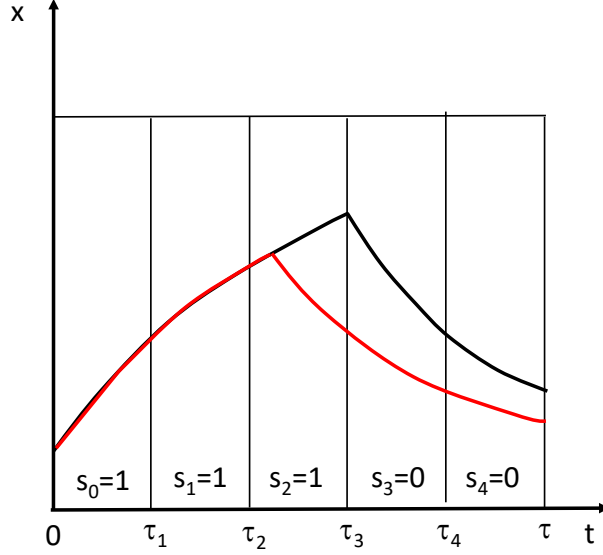
Figure 1: Deterministic partition of time in the push forward method. Trajectories of the PDMP model (red line) having a jump of the discrete state inside $[\tau_2, \tau_3)$ are replaced by a trajectory (black line) having a jump of the discrete state at τ_3.

the fact that, given $\mathbf{x}_t$, s_t is a finite state Markov process. Therefore,

$$\mathbb{P}[S_M] = \Pi_{s_{M-1},s_{M-2}}(\tau_{M-2}, \tau_{M-1}) \ldots \Pi_{s_1,s_0}(\tau_0, \tau_1) P_0^S(s_0), \qquad (19)$$

where $P_0^S : \mathcal{S} \to [0,1]$ is the initial distribution of the discrete state, and $\mathbf{\Pi}(\tau_j, \tau_{j+1})$ is the solution, at $t = \tau_{j+1}$, of

$$\frac{\mathrm{d}\mathbf{\Pi}(\tau_j, t)}{\mathrm{d}t} = H(\mathbf{x}_t)\mathbf{\Pi}(\tau_j, t) \qquad (20)$$

with $\mathbf{\Pi}(\tau_j, \tau_j) = \mathbb{I}$ and $\mathbf{x}_t$ is given by (18).

In order to compute the probability distribution of $\mathbf{x}$ at time τ one has to sum the contributions of all solutions of (18), obtained for the N_s^M realisations of promoter state paths with weights given by the probabilities of the paths.

Suppose that we want to estimate the distribution of $\mathbf{x}(\tau)$, using a multivariate histogram with bin centers $\mathbf{c}^{(l_1,\ldots,l_N)} = (c_1^{l_1}, \ldots, c_N^{l_N})$, $1 \leq l_i \leq n_x, 1 \leq i \leq N$ where n_x is the number of bins in each direction x_i, $1 \leq i \leq N$. The initial distribution of $\mathbf{x}$ at time $t = 0$ is given by the bin probabilities $p_0^{(l_1,\ldots,l_N)}$, $1 \leq i \leq N, 1 \leq l_i \leq n_x$ and the distribution at time τ are given by the probabilities $p_\tau^{(l_1,\ldots,l_N)}$, $1 \leq i \leq N, 1 \leq l_i \leq n_x$.

Let $\mathbf{x}^{(l_1,\ldots,l_N)}(t)$ be the solution of (18) with $\mathbf{x}^{(l_1,\ldots,l_N)}(0) = \mathbf{c}^{(l_1,\ldots,l_N)}$ and let $(l_1', \ldots, l_N')$ be the histogram bin containing $\mathbf{x}^{(l_1,\ldots,l_N)}(\tau)$. The application $(l_1, \ldots, l_N) \to (l_1', \ldots, l_N') := \psi(l_1, \ldots, l_N)$ is in general many to one. Given the

14

probability $\mathbb{P}[S_M]$ of a path $S_M \in \Omega$, the push forward distribution of $\mathbf{x}(\tau)$ is computed as

$$p_\tau^{(l_1',\ldots,l_N')} = \sum_{S_M \in \Omega} \sum_{\psi(l_1,\ldots,l_N)=(l_1',\ldots,l_N')} p_0^{(l_1,\ldots,l_N)} \mathbb{P}[S_M], \qquad (21)$$

The push-forward method can be applied recursively to compute the distribution at times $\tau, 2\tau, \ldots, n_t\tau$. The complete algorithm is schematized in Algorithm 4 for the choice $M = 4$.

Remark 1. The Algorithm 4 can be used for computing the full multivariate probability distribution of the vector $\mathbf{x}(t)$, but also for computing marginal distributions. For gene network the full multivariate distribution implies products from all genes, whereas a marginal distribution can select only one, or a small number of genes. For marginals, the dimension N is replaced by $N_m < N$ where N_m is the number of components of interest of the vector $\mathbf{x}(t)$.

Remark 2. For gene networks, the numerical integration of the ODEs (steps 7,9,11 in the Algorithm 4) can be replaced by symbolic solutions. For each path realization $S_M := (\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{M-1}) \in \Omega := \{0, 1, \ldots, 2^N - 1\}^M$ of promoter states, we can compute the protein and mRNA levels, $\mathbf{y}_t$ and $\mathbf{r}_t$, respectively, of all genes $i \in \{1, N\}$, at $t = \tau$:

$$r_\tau^i = r_0^i e^{-\rho\tau} + \frac{k_0}{\rho}(1 - e^{-\rho\tau}) + \frac{k_1 - k_0}{\rho} \sum_{j=1}^{M-1} e^{-\rho\tau}(e^{-\rho\tau_{j+1}} - e^{-\rho\tau_j})s_j^i \quad (22)$$

$$y_\tau^i = y_0^i e^{-a\tau} + \frac{br_0^i}{a-\rho}(e^{-\rho\tau} - e^{-a\tau}) + \frac{bk_0}{\rho}\left(\frac{1 - e^{-a\tau}}{a} + \frac{e^{-a\tau} - e^{-\rho\tau}}{a-\rho}\right)$$

$$+ \frac{b(k_1 - k_0)}{\rho}e^{-a\tau}\sum_{j=1}^{M} s_{j-1}^i w_j, \qquad (23)$$

for $i \in \{1, \ldots, N\}$. Here,

$$w_j = \frac{e^{(a-\rho)\tau} - e^{(a-\rho)\tau_j}}{a-\rho}(e^{\rho\tau_j} - e^{\rho\tau_{j-1}})$$

$$- \frac{e^{(a-\rho)\tau_j} - e^{(a-\rho)\tau_{j-1}}}{a-\rho}e^{\rho\tau_{j-1}} + \frac{e^{a\tau_j} - e^{a\tau_{j-1}}}{a}$$

with $s_j^i := 0$ if promoter i is OFF for $t \in [\tau_j, \tau_{j+1})$ and $s_j^i := 1$ if promoter i is ON for $t \in [\tau_j, \tau_{j+1})$.

3.3.2. Complexity issues

The complexity of the push forward algorithm scales as $n_t n_x^N N_s^M (N + N_s^2)$, because there are N_s^M possible paths S_M, n_x^N histogram bins, and n_t time complexity for solving (18), (20) is $O(N + N_s^2)$. The complexity of computing marginal distributions of $N_m < N$ variables is lower and scales as $n_t n_x^{N_m} N_s^M (N + N_s^2)$.

3.3.3. Mean field push-forward method

A way to mitigate the computational burden of the Algorithm 4 is to use the mean field approximation. In the mean field approximation, the probabilities $\mathbb{P}[S_M]$ are computed from averaged equations that are identical for each histogram bin.

More precisely, equation (20) is replaced by

$$\frac{\mathrm{d}\mathbf{\Pi}(t',t)}{\mathrm{d}t} = \mathbb{E}[H(\mathbf{x}_t)]\,\mathbf{\Pi}(t',t) \qquad (24)$$

where $t' \leq t$, $\mathbf{\Pi}(t',t') = \mathbb{I}$.

Using a Taylor expansion of $H(\mathbf{x}_t)$ around the expectation $\mathbb{E}[\mathbf{x}_t]$, one gets

$$\frac{\mathrm{d}\mathbf{\Pi}(t',t)}{\mathrm{d}t} = H(\mathbb{E}[\mathbf{x}_t]) + \frac{1}{2}\big(H''(\mathbb{E}[\mathbf{x}_t]) : \mathrm{Var}(\mathbf{x}_t)\big)\mathbf{\Pi}(t',t), \qquad (25)$$

where $\mathrm{Var}(\mathbf{x}_t)$ is the variance/covariance matrix of $\mathbf{x}_t$, H'' is the element-wise second derivative matrix of H and : stands for the double dot product.

In (25) the moments $\mathbb{E}[\mathbf{x}_t]$ and $\mathrm{Var}(\mathbf{x}_t)$ are either available analytically or are solutions of ODEs obtained using moment closures such as in [23].

The main advantage of the mean-field approximation is that the transition matrix elements can be computed outside the discrete variables loop. The complete algorithm is presented in the Algorithm 5. The time complexity in this case is $n_t n_x^N N_s^M N$, because in the innermost loop one has to solve only (18), whose time complexity is $O(N)$.

For gene networks, the mean field push-forward procedure can be applied also to ODE dynamics of the individual genes. This is possible because during ON or OFF periods, the ODE dynamics of one gene is uncoupled from that of another gene. Furthermore, the $N_s \times N_s$ transition matrix $\mathbf{H}$ can be replaced by $N/2$ 2×2 transition matrices of one gene with elements averaged over the values of the other genes. This approximation reduces the complexity of the calculations to $2n_t(n_x)^2 N 2^M$, which is linear in the number of genes. The mean field push-forward algorithm for gene networks is presented in the Algorithm 6.

This method has already been used for particular models. In [14] we have replaced the regulation term $f_2 y_1(t)$ occurring in the transition matrix of the gene network model M_1 by its mean $f_2\mathbb{E}[y_1(t)]$. This means that the gene 2 switches between its ON and OFF states with rates given by the mean of the regulatory protein y_1. In this case both $\mathbf{H}$ and $\mathbf{\Pi}$ can be computed analytically, which leads to a drastic reduction in the execution time. This simple mean field approach is suitable for the model M_1, which contains only linear regulation terms. For non-linear regulation terms, $\mathbf{\Pi}$ can not generally be computed analytically. Moreover, the naive mean field approach introduces biases. For instance, in the case of the model M_2, the approximation $f_2 y_1(t)/(K_1 + y_1(t)) \approx f_2\mathbb{E}[y_1(t)]/(K_1 + \mathbb{E}[y_1(t)])$ is poor. In the CMSB2019 paper we proposed a better approximation [16], in which we replace $f_2 y_1(t)/(K_1 + y_1(t))$ by its mean value and use

$$\mathbb{E}\left[\frac{f_2 y_1(t)}{K_1 + y_1(t)}\right] \approx \frac{f_2\mathbb{E}[y_1(t)]}{(K_1 + \mathbb{E}[y_1(t)])} - \frac{f_2}{(K_1 + \mathbb{E}[y_1(t)])^3}\,\mathrm{Var}(y_1(t)), \qquad (26)$$

in order to correct the bias. This approach is generalized by (25).

Algorithm 5 PDMPpushforwardmeanfield
___

Input: 1. $p_0^g(x, s)$, initial distribution of (x_0, s_0). 2. τ time step for computing the time-dependent distribution

Output: n_t vectors $H_0, \dots, H_{nt}$ representing the distribution of x at times $0, \tau, \dots, n_t\tau$

1: compute $H_0, P0$ from initial distribution
2: **for** $i := 1$ **to** n_t **do**
3:    **for** $s0 := 1$ **to** N_s **do**
4:       **for** $s1 := 1$ **to** N_s **do**
5:          **for** $s2 := 1$ **to** N_s **do**
6:             **for** $s3 := 1$ **to** N_s **do**
7:                Compute
               $P_M(s_0, s_1, s_2, s_3) = \Pi_{s3,s2}(\tau_2, \tau_3)\Pi_{s2,s1}(\tau_1, \tau_2)\Pi_{s1,s0}(\tau_0, \tau_1)P0_{s0}$
               where $\Pi(t', t)$ is the solution of (25).
8:             **end for**
9:          **end for**
10:       **end for**
11:    **end for**
12:    **for** $s0 := 0$ **to** 1 **do**
13:       **for** $j := 1$ **to** n_x^N **do**
14:          solve $\frac{d\mathbf{x0}}{dt} = V_{s_0}(\mathbf{x0})$, with initial conditions $\mathbf{x0}(0) = C_j$, from $t = 0$ to $t = \tau_1$
15:          **for** $s1 := 0$ **to** 1 **do**
16:             solve $\frac{d\mathbf{x1}}{dt} = V_{s_1}(\mathbf{x1})$ with initial conditions $x1(\tau_1) = x0(\tau_1)$ from $t = \tau_1$ to $t = \tau_2$
17:             **for** $s2 := 0$ **to** 1 **do**
18:                solve $\frac{d\mathbf{x2}}{dt} = V_{s_2}(\mathbf{x2})$ with initial conditions $x2(\tau_2) = x1(\tau_2)$ from $t = \tau_2$ to $t = \tau_3$
19:                **for** $s3 := 0$ **to** 1 **do**
20:                   solve $\frac{d\mathbf{x3}}{dt} = V_{s_3}(\mathbf{x3})$ with initial conditions $x3(\tau_3) = x2(\tau_3)$ from $t = \tau_3$ to $t = \tau_4$
21:                   $\mathbf{x} = \mathbf{x3}(\tau_4)$
22:                   $H_i(BIN(\mathbf{x})) = H_i(BIN(\mathbf{x})) + P_M(s_0, s_1, s_2, s_3)H_{i-1}(j)$ where $BIN(\mathbf{x})$ is the bin containing $\mathbf{x}$
23:                **end for**
24:             **end for**
25:          **end for**
26:       **end for**
27:    **end for**
28: **end for**

Algorithm 6 PDMPpushforwardmeanfieldgenenetwork

Input: 1. for each gene $g \in \{1, 2, \ldots, N/2\}$, $p_0(x, s)$, initial distribution of (x_0, s_0), where $x = (r, p)$. 2. τ time step for computing the time-dependent distribution

Output: for each gene $g \in \{1, 2, \ldots, N/2\}$, n_t vectors $H_0, \ldots, H_{nt}$ representing the distribution at times $0, \tau, \ldots, n_t \tau$

1: **for** $g := 1$ **to** $N/2$ **do**
2:    compute $H_0, P0$ from initial distribution for gene g
3:    **for** $i := 1$ **to** n_t **do**
4:      **for** $s0 := 0$ **to** 1 **do**
5:        **for** $s1 := 0$ **to** 1 **do**
6:          **for** $s2 := 0$ **to** 1 **do**
7:            **for** $s3 := 0$ **to** 1 **do**
8:              Compute
$$P_M(s_0, s_1, s_2, s_3) = \Pi_{s3,s2}(\tau_2, \tau_3)\Pi_{s2,s1}(\tau_1, \tau_2)\Pi_{s1,s0}(\tau_0, \tau_1)P0_{s0}$$
             where $\Pi(t', t)$ is the solution of (25) for gene g.
9:            **end for**
10:          **end for**
11:        **end for**
12:      **end for**
13:      **for** $s0 := 1$ **to** 2 **do**
14:        **for** $j := 1$ **to** n_x^2 **do**
15:          solve $\frac{\mathrm{d}\mathbf{x}0}{\mathrm{d}t} = V_{s_0}(\mathbf{x}0)$, with initial conditions $\mathbf{x}0(0) = C_j$, from $t = 0$ to $t = \tau_1$, and V_s is defined by (10) for gene g
16:          **for** $s1 := 1$ **to** N_s **do**
17:            solve $\frac{\mathrm{d}\mathbf{x}1}{\mathrm{d}t} = V_{s_1}(\mathbf{x}1)$ with initial conditions $x1(\tau_1) = x0(\tau_1)$ from $t = \tau_1$ to $t = \tau_2$
18:            **for** $s2 := 1$ **to** N_s **do**
19:              solve $\frac{\mathrm{d}\mathbf{x}2}{\mathrm{d}t} = V_{s_2}(\mathbf{x}2)$ with initial conditions $x2(\tau_2) = x1(\tau_2)$ from $t = \tau_2$ to $t = \tau_3$
20:              **for** $s3 := 1$ **to** N_s **do**
21:                solve $\frac{\mathrm{d}\mathbf{x}3}{\mathrm{d}t} = V_{s_3}(\mathbf{x}3)$ with initial conditions $x3(\tau_3) = x2(\tau_3)$ from $t = \tau_3$ to $t = \tau_4$
22:                $\mathbf{x} = \mathbf{x}3(\tau_4)$
23:                $H_i(BIN(\mathbf{x})) = H_i(BIN(\mathbf{x})) + P_M(s_0, s_1, s_2, s_3)H_{i-1}(j)$ where $BIN(\mathbf{x})$ is the bin containing $\mathbf{x}$
24:              **end for**
25:            **end for**
26:          **end for**
27:        **end for**
28:      **end for**
29:    **end for**
30: **end for**

As in [14] we can use analytic expressions for $\mathbb{E}[y_1(t)]$, but also for $\mathrm{Var}(y_1(t))$. These expressions can be found in the Appendix A. Although the elements of matrix $\mathbf{H}$ have analytic expressions, the elements of the matrix $\mathbf{\Pi}$ contain integrals that must be computed numerically. For the model M_2, we have

$$\mathbf{\Pi}_1(\tau, \tau') = \begin{bmatrix} (1 - p_{1,on}) + p_{1,on}e^{-\epsilon_1(\tau' - \tau)} & (1 - p_{1,on})(1 - e^{-\epsilon_1(\tau' - \tau)}) \\ p_{1,on}(1 - e^{-\epsilon_1(\tau' - \tau)}) & p_{1,on} + (1 - p_{1,on})e^{-\epsilon_1(\tau' - \tau)} \end{bmatrix}, \quad (27)$$

for the transition rates of the first gene, where $p_{1,on} = f_1/(f_1 + h_1)$, $\epsilon_1 = (f_1 + h_1)/\rho$, and

$$\mathbf{\Pi}_2(\tau, \tau') = \begin{bmatrix} K(\tau, \tau') + h_2 \int_\tau^{\tau'} K(t, \tau')\, dt & h_2 \int_\tau^{\tau'} K(t, \tau')\, dt \\ 1 - K(\tau, \tau') - h_2 \int_\tau^{\tau'} K(t, \tau')\, dt & 1 - h_2 \int_\tau^{\tau'} K(t, \tau')\, dt \end{bmatrix}, \quad (28)$$

for the transitions of the second gene, where $K(\tau, \tau') = e^{-\int_\tau^{\tau'}(h_2 + F_2(t))\, dt}$ and $F_2(t) = f_2 \mathbb{E}\left[\frac{y_1(t)}{K_1 + y_1(t)}\right]$.

4. Results

4.1. Convergence of the push-forward method

The probability distribution obtained with the push-forward method converges to the exact PDMP distribution in the limit $M \to \infty$. This is a consequence of the following theorem:

Theorem 1. *Let $\Phi_{S_M}(t, \mathbf{x})$ be the flow defined by the formulas (18), such that $\mathbf{x}_t = \Phi_{S_M}(t, \mathbf{x}(0))$ for $t \in [0, \tau]$. Let $\mu_t^M : \mathcal{B}(\mathbb{R}^N) \to \mathbb{R}_+$ be the probability measure defined as $\mu_t^M(A) = \sum_{S_M \in \Omega} \mathbb{P}[S_M] \mu_0(\Phi_{S_M}^{-1}(t, A))$, where $\mu_0 : \mathcal{B}(\mathbb{R}^N) \to \mathbb{R}_+$ is the probability distribution of $\mathbf{x}$ at $t = 0$, $\mathbb{P}[S_M]$ is given by (19), and $\mathcal{B}(\mathbb{R}^N)$ is the σ-algebra of Borel sets on $\mathbb{R}^N$. Let μ_t, the exact distribution of $\mathbf{x}_t$ for the PDMP defined in Section 2.2, with initial values $(\mathbf{x}_0, s_0)$ distributed according to $\mu_0 \times P_0^S$. Assume that the vector fields $\mathbf{V}_s$, $s \in S$, and the transition matrix $\mathbf{H}$ are C^1 functions and that there exists a bounded set of $\mathbb{R}^N$ such that all flows Φ_s, $s \in S$, leave B invariant. Assume that $|\tau_i - \tau_{i-1}| < C/M$ for all $i \in [1, M]$, where C is a positive constant. Then, for all $t \in [0, \tau]$, μ_t^M converges in distribution to μ_t, when $M \to \infty$. More precisely, for all Lipschitz functions φ on $\mathbb{R}^N$, there exists a constant κ depending on the data of the PDMP, τ and the Lipschitz constant of φ such that:*

$$\left| \int_B \varphi(x) \mu_t^M(dx) - \int_B \varphi(x) \mu_t(dx) \right| = |\mathbb{E}(\varphi(\Phi_{S_M}(t, \mathbf{x}))) - \mathbb{E}(\varphi(\mathbf{x}_t))| \leq \kappa/M.$$

Also for all Borel set A in $\mathcal{B}(\mathbb{R}^N)$ such that $\mu_t(\partial A) = 0$ we have $\mu_t^M(A) \to \mu_t(A)$ when $M \to \infty$. Here, ∂A denotes the boundary of A.

The proof of this theorem is given in Appendix B. It is inspired by the classical proof of weak order of convergence for the Euler scheme for a stochastic differential equation (see [24]).

Remark 3. If the flows Φ_s are not known explicitly, a numerical scheme can be used. This introduces another source of error. Our proof easily extends and provide a similar result of convergence. If one wants to investigate the convergence of $p_\tau^{(l_1,\ldots,l_N)}$ given by (21), this follows from the above theorem only under the assumption that the probability that the PDMP is on the boundary of the bins at time t is 0. Note this is not restrictive and happens only in pathological situations. Also, if the initial distribution has a smooth density, we can prove that the error estimate above holds for borelian bounded functions φ, thus we can choose φ to be an indicator function and obtain error bounds for these probabilities without this restriction. More precisely, we have for any Borel set A:

$$\left| \mu_t^M(A) - \mu_t(A) \right| \le \kappa/M,$$

where now κ depends on the Lipschitz constant of the initial density, see Remark 4 in Appendix B.

4.2. Testing the performance of various methods

4.2.1. Testing accuracy and speed of push-forward method compared to the Monte-Carlo method

In order to test the push-forward method, we compared the resulting probability distributions with the ones obtained by the Monte-Carlo method using the direct simulation of the PDMP. We considered the models M_1 and M_2 with the following parameters: $\rho = 1$, $p_1 = \frac{f}{f+h} = 1/2$, $a = 1/5$, $b = 4$, $k_0 = 4$, $k_1 = 40$ for the two genes. For the parameter $\epsilon = \frac{f+h}{\rho}$ we used two values: $\epsilon = 0.5$ for slow genes and $\epsilon = 5.5$ for fast genes. We tested the slow-slow and the fast-fast combinations of parameters.

The initial distribution of the promoters state was $P_0^S((0,0)) = 1$ where the state $(0,0)$ means that both promoters are OFF. The initial probability measure μ_0 was a delta Dirac distribution centered at $r^1 = r^2 = 0$ and $y^1 = y^2 = 0$. This initial condition is obtained by always setting the direct simulation of the PDMP to start from $r_1(0) = r_2(0) = 0$, $y_1(0) = y_2(0) = 0$, and $s_0^1 = s_0^2 = 0$. The simulations were performed between $t_0 = 0$ and $t_{max} = 20$ for fast genes and between $t_0 = 0$ and $t_{max} = 90$ for slow genes. In order to estimate the distributions we have used $MC = 50000$ samples for the highest sampling. The push-forward method was implemented with $M = 10$ equal length sub-intervals of $[0, \tau]$. The time step τ was chosen to be $\tau = 2$ for fast genes and $\tau = 15$ for slow genes. The procedure was iterated 10 times for fast genes (up to $t_{max} = 20$) and 6 times for slow genes (up to $t_{max} = 90$).

The execution times are shown in the Table 1. The comparison of the probability distributions are illustrated in the Figures 2 and 3. In order to quantify the relative difference between methods we use the L^1 distance between distributions. More precisely, if $p(x)$ and $\tilde{p}(x)$ are probability density functions to be compared, the distance between them is

$$d = \int |p(x) - \tilde{p}(x)|\, dx. \tag{29}$$

Model	Monte-Carlo high sampling [min]	Push-forward [s]
M_1 slow-slow	45	20
M_1 fast-fast	74	30
M_2 slow-slow	447	20
M_2 fast-fast	758	30

Table 1: Execution times for different methods. All the methods were implemented in Matlab R2013b running on a single core (multi-threading inactivated) of a Intel i5-700u 2.5 GHz processor. The Monte-Carlo method computed the next jump waiting time (Algoritm 1) using the analytical solution of (15) for M_1 and the numerical solution of (15) for M_2. The push-forward method used Algorithm 6 and analytic solutions for mRNA and protein trajectories from (23), (18) and numerical computation of the integrals in (28), for both models.
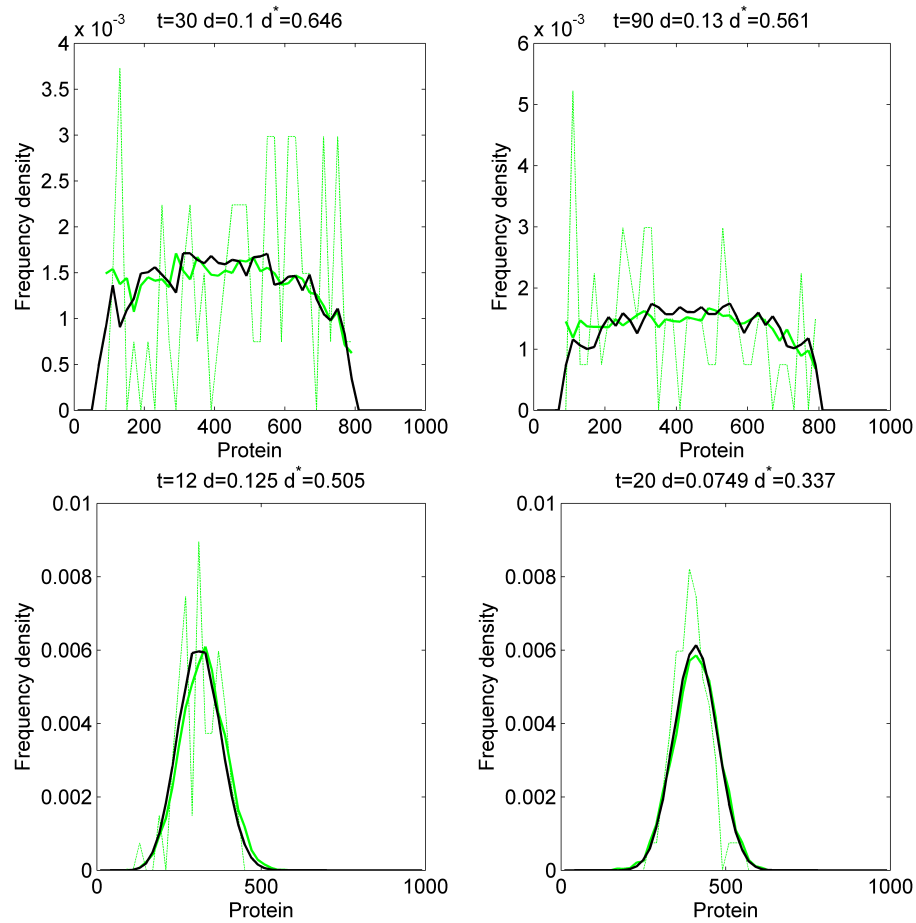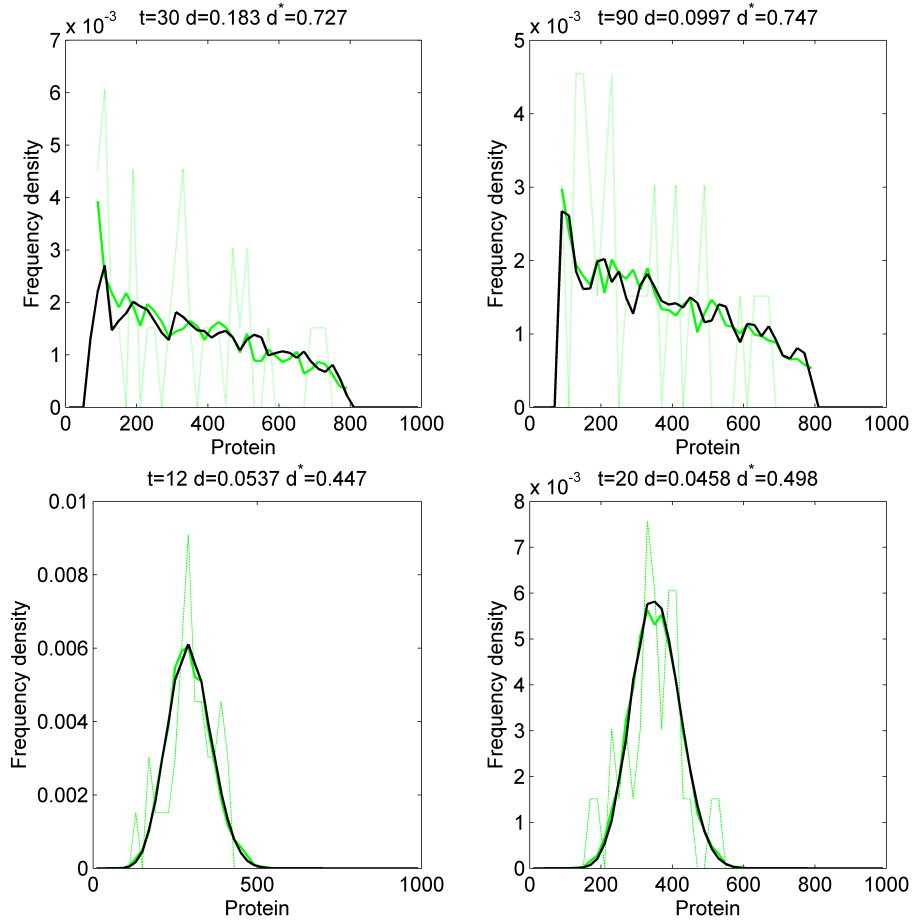

Figure 2: Histograms of protein for the second gene, produced by the Monte-Carlo method (green lines) and by the Push-forward method (black lines) for the model M_1. The green dotted line results from low sampling Monte-Carlo with similar execution time as the push-forward method, whereas the solid green line results from high sampling Monte-Carlo. The distances, defined by (29), are between low sampling and high sampling Monte-Carlo (d^*) and between push-forward and high sampling Monte-Carlo (d).

Figure 3: Histograms of protein for the second gene, produced by the Monte-Carlo method (green lines) and by the Push-forward method (black lines) for the model M_2. The green dotted line results from low sampling Monte-Carlo with similar execution time as the push-forward method, whereas the solid green line results from high sampling Monte-Carlo. The distances, defined by (29), are between low sampling and high sampling Monte-Carlo (d^*) and between push-forward and high sampling Monte-Carlo (d).

This distance was computed for distributions resulting from the push-forward method and the Monte-Carlo method with the highest sampling. We have also used a reduced sampling Monte-Carlo scheme whose execution time is similar to the one of the push-forward method. The distributions resulting from low sampling and high sampling Monte-Carlo were compared using the same distance. Figures 2 and 3 clearly show that, for the same execution time, the push-forward method outperforms the Monte-Carlo method.

4.2.2. Comparing the Monte Carlo, push-forward and Liouville-master methods

Using Python, we have implemented Algorithm 4 for the one gene model, to compute mRNA $(r(t))$ and protein $(y(t))$ probability densities $P(r,t)$ and $P(y,t)$, respectively. For the one gene model, we have considered two switching regimes: slow ($\epsilon = 0.5$) and fast ($\epsilon = 5.5$). Also in Python, we have implemented Algorithm 5 for the two genes model, to compute the probability densities for mRNA and protein associated to gene one ($P(r_1,t)$ and $P(y_1,t)$) and for mRNA and protein associated to gene two ($P(r_2,t)$ and $P(y_2,t)$). For the two genes model, we have considered four different switch configurations: slow-slow, fast-fast, slow-fast and fast-slow. For all implementations each time interval $[0, \tau]$ has been partitioned into four sub-intervals of equal sizes ($M = 4$) resulting in the sequence $\{s_0, s_1, s_2, s_3\}$, representing the state of the switch inside each sub-interval ($s = s_j := s_{t_j}$ for $t \in [t_j, t_{j+1}), j = 0, ..., M - 1$), leading to 2^4 path realizations. For all slow genes we have set $t_0 = 0$, $t_4 = 9$ and evaluated the solution up to $t_{max} = 90$, using the composition rule between successive time intervals. For all fast genes we have set $t_0 = 0$, $t_4 = 1$ and evaluated the solution up to $t_{max} = 20$, using the composition rule between successive time intervals, as well.

Model	Monte-Carlo (high sampling)	PF	LME
One gene slow	-	2.55	97.39
One gene fast	-	5.12	21.64
Two genes slow-slow	45	6.86	97.39
Two genes fast-fast	74	10.85	21.64
Two genes slow-fast	243	9.22	21.64
Two genes fast-slow	249	8.83	97.39

Table 2: Execution times (in minutes for all cases) for different methods to compute the probability distributions for one gene model and two genes model. Methods: PF = Push-forward, LME = Liouville-master equation

We have compared the results obtained by the implementation of the push forward method (PF), as described above, with Monte-Carlo simulation (MC) (Algorithms 1 and 2) and with numerical solution of the Liouville-master equation (LME) (Algorithm 3). The comparison between execution times for each model and each method can be found in Table 2 (all the execution times are expressed in minutes).

In order to illustrate our results we have produced Figures 4, 5, 6 and 7, for selected models and for methods: Push forward (PF), Monte-Carlo simulation (MC) and Liouville-master equation (LME).

The one gene model (Figures 4 and 5) has as initial conditions $p_0(0) = 1$, where $p_0(0)$ is the probability to find the switch in state 0 at time $t = 0$, and $r(0) = y(0) = 0$, for both switch regimes: slow and fast. The two genes model (Figures 6 and 7) has as initial conditions $p_0^1(0) = p_0^2(0) = 1$ (meaning that at time $t = 0$ both genes are in state 0 with probability one), $r_1(0) = y_1(0) =$
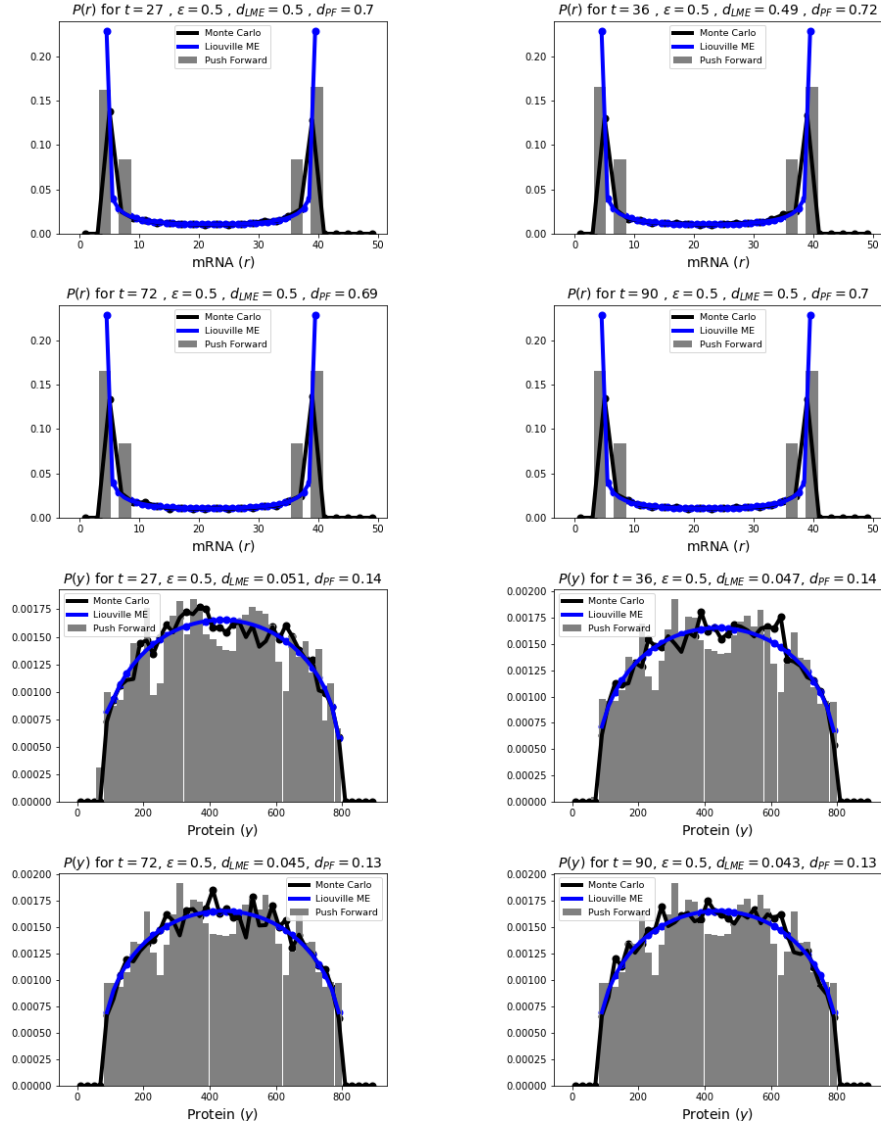
Figure 4: This set of plots shows the comparison between the computed probability distribution for mRNA ($P(r,t)$) and protein ($P(y,t)$), using the push forward method (Algorithm 4), direct Monte-Carlo simulation (Algorithms 1 and 2) and numerical solution of the Liouville-master Equation (Algorithm 3). The model is for one gene in the slow switch regime ($\epsilon = 0.5$). The execution times of PF is 2.55 minutes in Python and 97.39 minutes for LME in MATLAB. The values d_{LME} and d_{PF} are shown in each plot and measure the distances between the LME equation and MC simulation, PF and MC simulation, respectively, as given in (29). The value of parameters are: $p_0 = 0.5$, $k_0 = 4$, $k_1 = 40$, $a = 1/5$ and $b = 4$. For initial conditions we set: $r(0) = y(0) = 0$ and $p_0(0) = 1$.
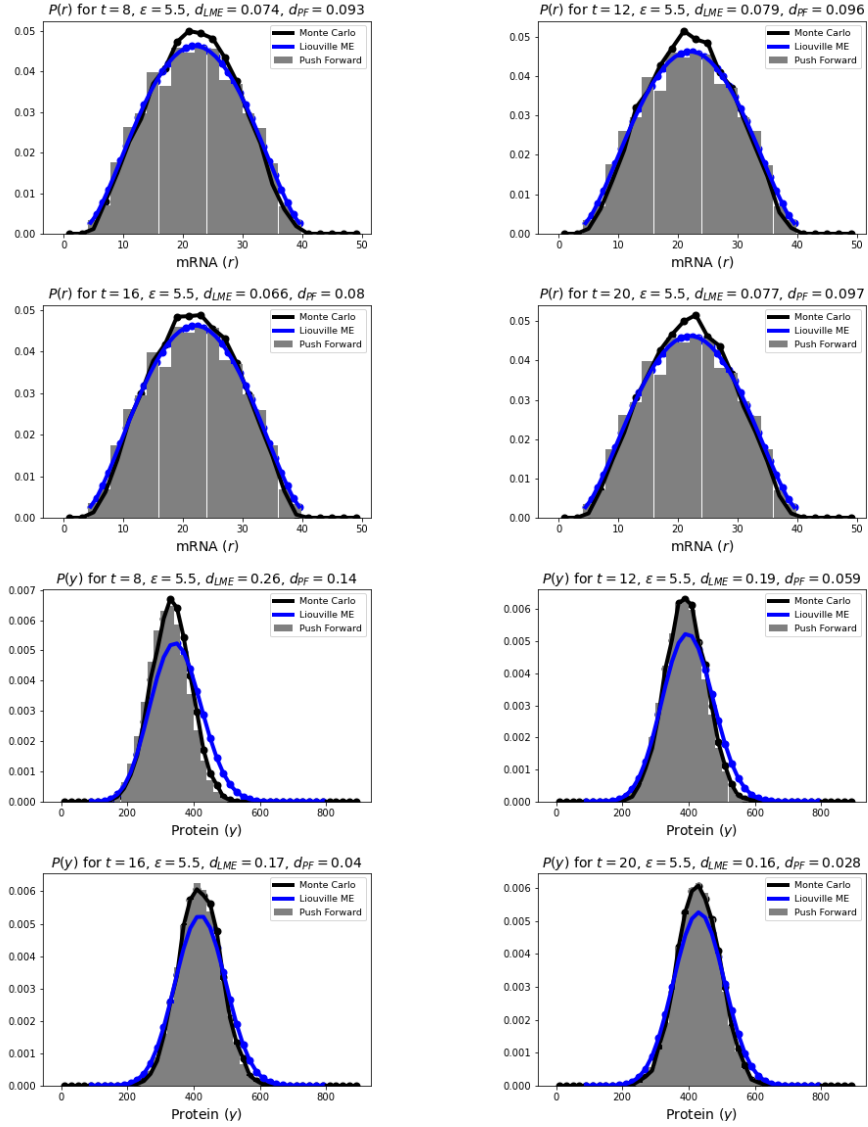
Figure 5: This set of plots shows the comparison between the computed probability distribution for mRNA $(P(r,t))$ and protein $(P(y,t))$, using PF (Algorithm 4), MC simulation (Algorithms 1 and 2) and numerical solution LME (Algorithm 3). The model is for one gene producing in the fast switch regime ($\epsilon = 5.5$). The execution time of the PF is 5.12 minutes in Pyhton, and 21.64 minutes for LME in MATLAB. The values d_{LME} and d_{PF} are shown in each plot and measure the distances between the LME equation and MC simulation, and PF and MC simulation, respectively, as given in (29). The value of parameters are: $p_0 = 0.5$, $k_0 = 4$, $k_1 = 40$, $a = 1/5$ and $b = 4$. For initial conditions we set: $r(0) = y(0) = 0$ and $p_0(0) = 1$.
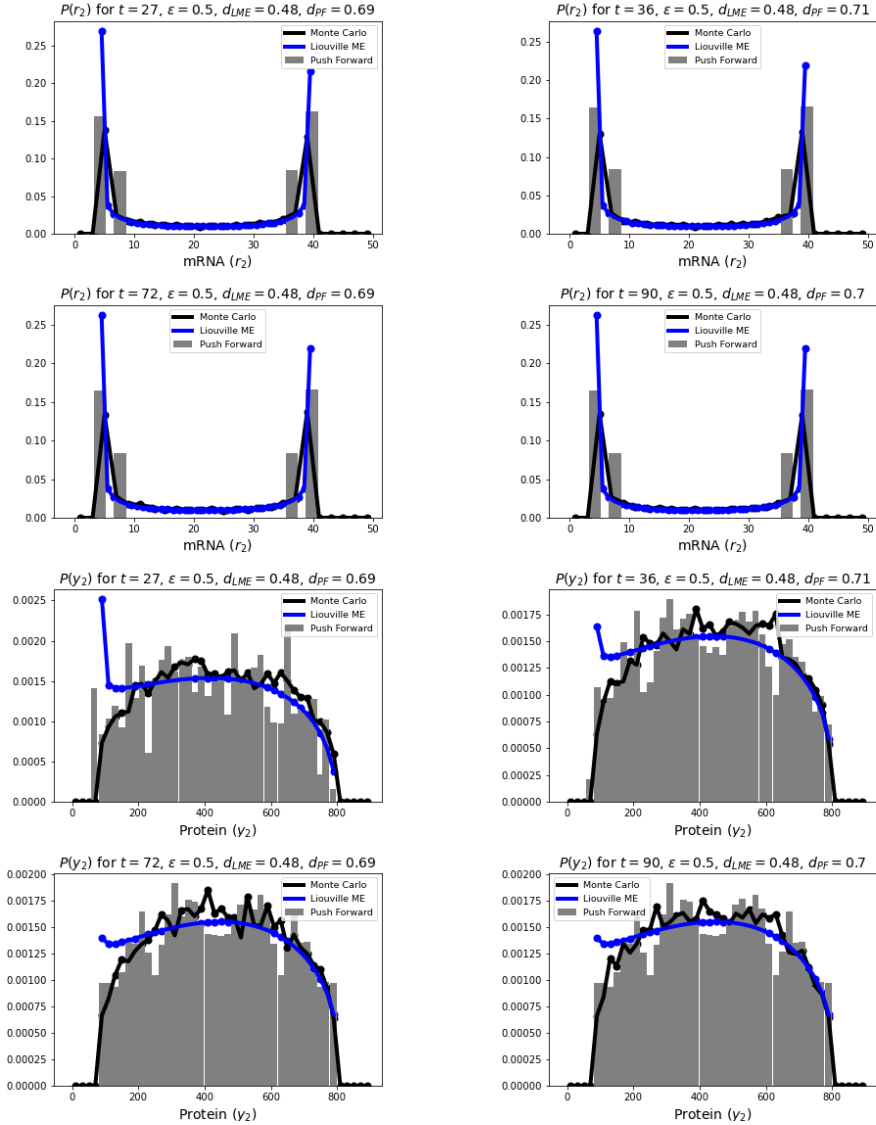
Figure 6: This set of plots shows the comparison between the computed probability density for mRNA ($P(r_2, t)$) and protein ($P(y_2, t)$) associated to gene two, using PF (Algorithm 5), MC simulation (Algorithms 1 and 2) and numerical solution LME (Algorithm 3). The model is M_1 in the slow-slow regime ($\epsilon = 0.5$). The execution time of PF is 6.86 minutes in Python, and 97.39 minutes for LME in MATLAB. The values d_{LME} and d_{PF} are shown in each plot and measure the distances between the LME equation and MC simulation, and PF and MC simulation, respectively, as given in (29). The value of parameters, for both genes are: $p_0 = 0.5$, $k_0 = 4$, $k_1 = 40$, $a = 1/5$ and $b = 4$. For initial conditions we set: $r_1(0) = y_1(0) = r_2(0) = y_2(0) = 0$ and $p_0(0) = 1$.

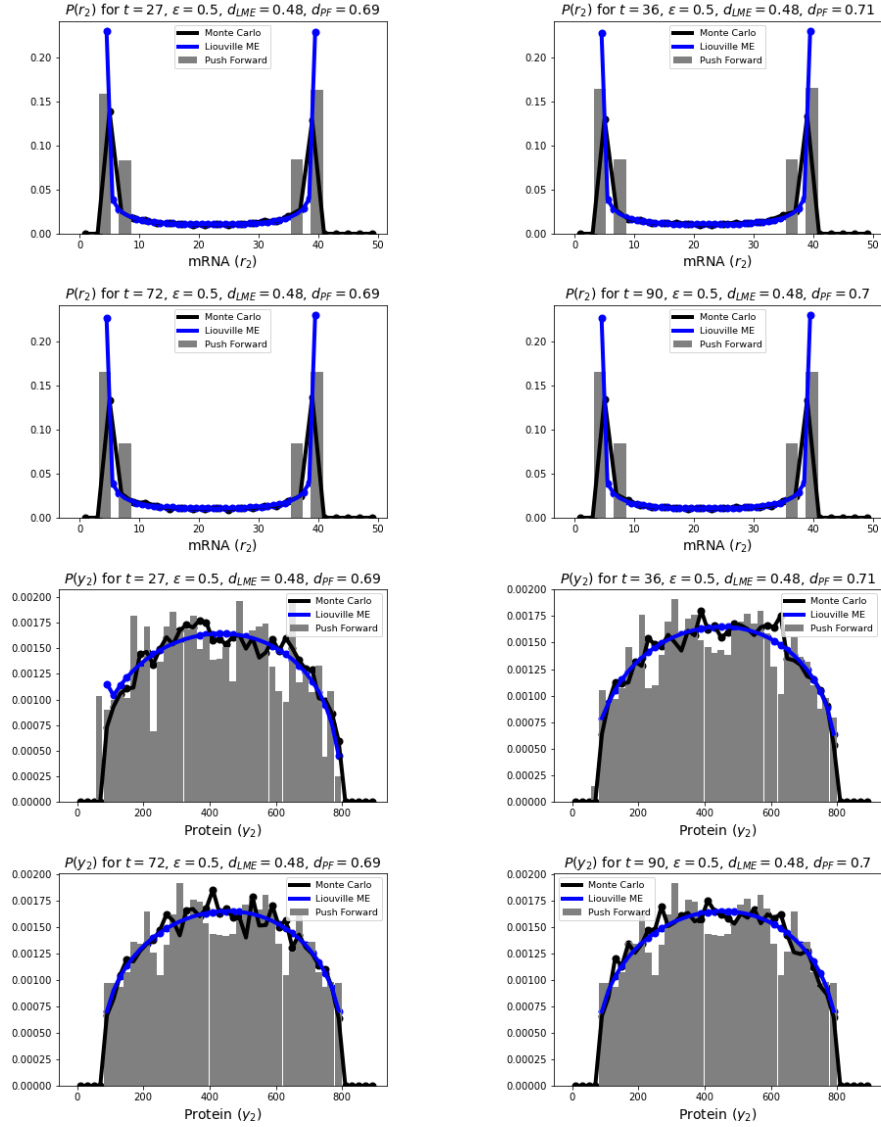Figure 7: This set of plots shows the comparison between the computed probability density for mRNA $(P(r_2, t))$ and protein $(P(y_2, t))$ associated to gene two, using PF (described in Algorithm 5), MC simulation (Algorithms 1 and 2) and numerical solution LME (Algorithm 3). The model is M_1 in the fast-slow regime ($\epsilon_1 = 5.5$ and $\epsilon_2 = 0.5$). The execution time of PF is 8.83 minutes in Python and 97.39 minutes for LME in MATLAB. The values d_{LME} and d_{PF} are shown in each plot and measure the distances between the LME equation and MC simulation, and PF and MC simulation, respectively, as given in (29). The value of parameters, for both genes are: $p_0 = 0.5$, $k_0 = 4$, $k_1 = 40$, $a = 1/5$ and $b = 4$. For initial conditions we set: $r_1(0) = y_1(0) = r_2(0) = y_2(0) = 0$ and $p_0(0) = 1$.

$r_2(0) = y_2(0) = 0$, for both switch configurations: slow-slow and slow-fast. We used the same parameters for all the models and for all the genes: $\epsilon_1 = \epsilon_2 = 0.5$ (slow switch), $\epsilon_1 = \epsilon_2 = 5.5$ (fast switch), $p_0^1 = p_0^2 = 0.5$, $\rho_1 = \rho_2 = 1$, $k_1(\sigma_1 = 0) = k_2(\sigma_2 = 0) = 4$, $k_1(\sigma_1 = 1) = k_2(\sigma_2 = 1) = 40$, $a_1 = a_2 = 1/5$ and $b_1 = b_2 = 4$. We have quantified the relative difference between the distributions obtained by two distinct methods using the L^1 distance (29). The distance between the distributions obtained by the Push Forward method and the Monte-Carlo method (high sampling) is indicated in the plots by d_{PF}, and the distance between the distributions obtained by the Liouville-master equation and Monte-Carlo method (high sampling) is indicated by d_{LME}.

5. Discussion and conclusion

Combining direct simulation of PDMP gene network models and analytic formulae for the ODE flow provides an effective, easy to implement method for computing time dependent, multivariate probability distributions of these models. However, the precision of the Monte-Carlo estimates of the distributions increases with $\sqrt{MC}$, where MC is the number of Monte-Carlo samples. For this reason, the execution time of the Monte-Carlo method, although smaller when compared to PDMP simulation methods, which implement numerical resolution of the ODEs, such as reported in [18] (data not shown), is large when compared to the push-forward method. The push-forward method represents an effective alternative to Monte-Carlo and Liouville-master equation methods, ensuring reduced execution time.

With respect to an earlier implementation of this method for gene networks in [13] we used promoter states instead of mRNA copy numbers as discrete variables of the PDMP. As a consequence, the number of discrete states is lower and we can afford to increase the number M of temporal subdivisions. Compared to the similar work done in [14] we used second moments of the protein distribution, which took into account the correlation of the promoter states and lead to increased accuracy in the case of nonlinear regulation. Although the protein moments and the exponential transition rate matrix $\mathbf{\Pi}$ can be computed numerically, the effectiveness of the push-forward method is increased when analytic expressions are available for these quantities. In this paper, these expressions were computed for particular cases.

The push-forward method is an approximate method, and its accuracy relies on the careful choice of the temporal and spatial step densities, namely of the integers M, n_t, n_x. We have rigorously proved that the convergence of the push-forward method is of order one, with errors that scale with $1/M$.

We situate our findings in the broader effort of the community to produce new effective simulation algorithms for computational biology. Although in this manuscript we restricted ourselves to examples of modeling gene networks, the push forward method and the algorithms of this paper can be applied to a broader class of phenomena where some continuous quantity is perturbed by a discrete stochastic process. A very interesting and important example is the stochastic representation of ion channel kinetics in neuron models [2].

28

The system of equations describing the kinetics of an ion channel is mathematically equivalent to the system of equations describing our one gene model. More complex neuron models with multiple types of ion channels (stochastic versions of Hodgkin-Huxley or Morris-Lecar deterministic models, for instance) are also covered by our PDMP formalism and algorithms. Thus, the push forward method as presented in Algorithm 4 can be used to obtain the histograms for the membrane voltage for rather general neuron models. A detailed comparison between the push forward method and the one presented in [2] will be the subject of a future work.

Appendices

Appendix A. Expectation and variance of the protein

For the sake of completeness, we give explicit formulas for the expectation and the variance of the protein synthesized by a constitutive promoter (gene 1 of models M_1 and M_2). The details of the calculation can be found in [16].

The expectation is given by

$$\mathbb{E}[y_t] = M_0 + M_1 e^{-at} + M_2 e^{-\rho t} + M_3 e^{-\epsilon t}, \tag{A.1}$$

where

$$
\begin{aligned}
M_0 &= \frac{b(k_0 + (k_1 - k_0)p_1)}{a}, \\
M_1 &= \mathbb{E}[x_0] - \frac{b\mathbb{E}[y_0]}{a - \rho} + \frac{bk_0}{a(a - \rho)} + \frac{b(k_1 - k_0)(p_{10} - p_1)}{(a - \rho\epsilon)(a - \rho)} + \frac{b(k_1 - k_0)p_1}{a(a - \rho)}, \\
M_2 &= \frac{b\mathbb{E}[y_0]}{a - \rho} - \frac{bk_0}{a - \rho} - \frac{b(k_1 - k_0)(p_{10} - p_1)}{\rho(1 - \epsilon)(a - \rho)} - \frac{b(k_1 - k_0)p_1}{a - \rho}, \\
M_3 &= \frac{b(k_1 - k_0)(p_{10} - p_1)}{\rho(1 - \epsilon)(a - \epsilon\rho)}.
\end{aligned}
$$

The variance is given by

$$
\begin{aligned}
\mathrm{Var}(y_t) &= \mathrm{Var}(y_0)e^{-2at} + b^2 \, \mathrm{Var}(r_0)\left(\frac{e^{-\rho t} - e^{-at}}{a - \rho}\right)^2 \\
&\quad - \left[\frac{(p_{10} - p_1)(k_1 - k_0)b}{\rho(1 - \epsilon)}\left(\frac{e^{-\rho\epsilon t} - e^{-at}}{a - \rho\epsilon} - \frac{e^{-\epsilon t} - e^{-at}}{a - \rho\epsilon}\right)\right]^2 \\
&\quad + \frac{p_1(1 - p_1)(k_1 - k_0)^2 b^2}{\rho^2}(V_0 + V_1 e^{-(a+\rho\epsilon)t} + V_2 e^{-\rho(1+\epsilon)t} + V_3 e^{-2at} + V_4 e^{-(a+\rho)t} + V_5 e^{-2\rho t}) \\
&\quad + \frac{(1 - 2p_1)(p10 - p_1)(k_1 - k_0)^2 b^2}{\rho^2}(V_6 e^{-\rho\epsilon t} + V_7 e^{-(a-\rho\epsilon)t} + V_8 e^{-\rho(\epsilon+1)t} + V_9 e^{-(a+\rho\epsilon)t} + V_{10} e^{-\rho t} + V_{11} e^{-2\rho t}),
\end{aligned}
\tag{A.2}
$$

where

$$V_0 = \frac{a + (\epsilon + 1)\rho}{a(a + \rho\epsilon)(a + \rho)(\epsilon + 1)}, \qquad V_1 = -\frac{2}{(a^2 - \rho^2\epsilon^2)(a - \rho)(\epsilon - 1)},$$

$$V_2 = \frac{2}{(a - \rho\epsilon)(a - \rho)(\epsilon^2 - 1)}, \qquad V_3 = \frac{1}{a(a - \rho\epsilon)(a - \rho)^2},$$

$$V_4 = \frac{2(a + (1 - 2\epsilon)\rho)}{(a - \rho\epsilon)(a - \rho)^2(a + \rho)(\epsilon - 1)}, \qquad V_5 = -\frac{1}{(\epsilon - 1)(a - \rho)^2},$$

$$V_6 = -\frac{2(2a + (2 - \epsilon)\rho)}{a(\epsilon - 2)(2a - \rho\epsilon)(a + (1 - \epsilon)\rho)}, \qquad V_7 = \frac{2}{(1 - \epsilon)a(a - \rho)(a - \rho\epsilon)},$$

$$V_8 = \frac{2}{(a - \rho\epsilon)(a - \rho)(\epsilon - 1)}, \qquad V_9 = \frac{2(a + (1 - 2\epsilon)\rho)}{(a - \rho)^2(\epsilon - 1)(a - \rho\epsilon)(a + (1 - \epsilon))},$$

$$V_{10} = \frac{2}{(a - \rho)^2(2 - \epsilon)(1 - \epsilon)}, \qquad V_{11} = \frac{2}{(a - \rho)^2(2a - \rho\epsilon)(a - \rho\epsilon)}.$$

Appendix B. Proof of the Theorem 1

For simplicity, we consider only the case $N = 1$, the proof generalizes immediately to $N > 1$ at the price of cumbersome notations. We also consider the case when the process $\mathbf{x}$ always belongs to $[0, 1]$. Thus, we may assume that V_s, $s \in S$ and H are bounded as well as their derivatives. We first consider the case of a Lipschitz function φ.

We denote by $(\mathbf{x}, s)$ the PDMP starting with the random initial data distributed along μ_0 and by $(\mathbf{x}(t, \zeta_0), s(t, \zeta_0))$ the PDMP starting from the deterministic initial data $\zeta_0 = (x_0, s_0)$.

In this proof, we use the letter κ for a constant which depends on V_s, $s \in S$, H and τ.

The push-forward algorithm can be rewritten as follows: starting from $\tilde{s}(0) = s_0$, $\tilde{x}(0) = \mathbf{x}_0$, we set $\tilde{x}(\tau_{j+1}) = \boldsymbol{\Phi}_{s(\tau_j)}(\tau_{j+1} - \tau_j, \tilde{x}(\tau_j))$ and then choose $s(\tau_{j+1})$ randomly, it is equal to $r \in S$ with probability $\Pi_{s(\tau_j), r}(\tau_j, \tau_{j+1})$, the latter matrix being defined in (20).

In fact, we consider more generally the approximation of $\mathbb{E}(\varphi(\mathbf{x}(\tau), s(\tau)))$, for a function $\varphi : \mathbb{R} \times S \to \mathbb{R}$, by $\mathbb{E}(\varphi(\tilde{x}(\tau), \tilde{s}(\tau)))$. The result is obtained when we choose φ independent on $s \in S$.

We first assume that φ is C^1 with respect to x. Let us denote by L its Lipschitz constant with respect to x:

$$|\varphi(x_1, s) - \varphi(x_2, s)| \leq L|x_1 - x_2|$$

for $x_1, x_2 \in \mathbb{R}$, $s \in S$.

The main tool for the proof is the generator of the process:

$$\mathcal{L}\varphi(x, s) = V_s(x)\partial_x\varphi + \sum_{r \in S}(\varphi(x, r) - \varphi(x, s))H_{r,s}(x),$$

and the forward Kolmogorov equation:

$$\frac{du}{dt}(t,x,s) = \mathcal{L}u(t,x,s), \ x \in \mathbb{R}, \ s \in S, \ t > 0.$$

When $u(0,x,s) = \varphi(x,s)$ for $x \in \mathbb{R}$, $s \in S$, it is well known that $u(t,x_0,s_0) = \mathbb{E}(\varphi(\mathbf{x}(t,\zeta_0), s(t,\zeta_0)))$ (see [7]). Therefore,

$$\mathbb{E}(\varphi(\mathbf{x}(t), s(t)) = \int_{\mathbb{R} \times S} \mathbb{E}(\varphi(\mathbf{x}(t,\zeta_0), s(t,\zeta_0)))d\mu_0(\zeta_0)$$

$$= \int_{\mathbb{R} \times S} u(t,x_0,s_0)d\mu_0(x_0,s_0).$$

It follows from Proposition A.1 in [6] that u satisfies the formula:

$$u(t,x,s) = \varphi(\Phi_s(t,x),s) \exp\left(-\int_0^t \lambda(\Phi_s(\sigma,x))d\sigma\right)$$

$$+ \sum_{s \in S} \int_0^t u(t-\sigma, \Phi_s(\sigma,x), r,) H_{r,s}(\Phi_s(\sigma,x)) \exp\left(-\int_0^\sigma \lambda(\Phi_s(\tau,x))d\tau\right) d\sigma$$

Following [6] we apply Gronwall's lemma to show that, when φ is Lipschitz with respect to x, so is u, with a Lipschitz constant less than $\kappa\|\varphi\|_{0,1}$. Here, κ depends on the characteristics of the PDMP and the time horizon τ; $\|\varphi\|_{0,1}$ is the sum of the supremum of φ with its Lipschitz constant. Moreover, it follows from the forward Kolmogorov equation that its time derivative is also bounded. Hence,

$$|u(t_1,x_1,s) - u(t_2,x_2,s)| \leq \kappa\|\varphi\|_{0,1}(|x_1 - x_2| + |t_1 - t_2|). \tag{B.1}$$

Now we rewrite the error as follows:

$$\mathbb{E}(\varphi(\mathbf{x}(\tau,\zeta_0), s(\tau,\zeta_0))) - \mathbb{E}(\varphi(\tilde{x}(\tau,\zeta_0), \tilde{s}(\tau,\zeta_0)))$$

$$= \mathbb{E}(u(\tau,x_0,s_0) - u(0, \tilde{x}(\tau,\zeta_0), \tilde{s}(\tau,\zeta_0)))$$

$$= \sum_{j=0}^{M-1} \mathbb{E}(u(\tau - \tau_j, \tilde{x}(\tau_j,\zeta_0), \tilde{s}(\tau_j,\zeta_0)) - u(\tau - \tau_{j+1}, \zeta_0, \tilde{x}(\tau_{j+1},\zeta_0), \tilde{s}(\tau_{j+1},\zeta_0))$$

On each interval $[\tau_j, \tau_{j+1}]$ we have, by chain rule and the forward Kolmogorov equation, that

$$\frac{d}{dt}[u(\tau - t, \tilde{x}(t,\zeta_0), \tilde{s}(\tau_j,\zeta_0))]$$

$$= -\frac{du}{dt}(\tau - t, \tilde{x}(t,\zeta_0), \tilde{s}(t,\zeta_0)) + \frac{d\tilde{x}}{dt}(t)\partial_x u(\tau - t, \tilde{x}(t,\zeta_0), \tilde{s}(\tau_j,\zeta_0)$$

$$= -\frac{du}{dt}(\tau - t, \tilde{x}(t,\zeta_0), \tilde{s}(t,\zeta_0)) + V_{\tilde{s}(\tau_j,\zeta_0)}(\tilde{x}(t,\zeta_0))\partial_x u(\tau - t, \tilde{x}(t,\zeta_0), \zeta_0, \tilde{s}(\tau_j,\zeta_0)$$

$$= -\sum_{r \in S}(u(\tau - t, \tilde{x}(t,\zeta_0), r) - u(\tau - t, \tilde{x}(t,\zeta_0), \tilde{s}(\tau_j,\zeta_0)))H_{r,\tilde{s}(\tau_j,\zeta_0)}(\tilde{x}(t,\zeta_0)).$$

Hence,

$$\mathbb{E}\big(u(\tau-\tau_j,\tilde{x}(\tau_j,\zeta_0),\tilde{s}(\tau_j),\zeta_0)-u(\tau-\tau_{j+1},\tilde{x}(\tau_{j+1},\zeta_0),\tilde{s}(\tau_{j+1}),\zeta_0))\big)$$

$$=\mathbb{E}\left(\int_{\tau_j}^{\tau_{j+1}}\sum_{r\in S}(u(\tau-t,\tilde{x}(t,\zeta_0),r)-u(\tau-t,\tilde{x}(t,\zeta_0),\tilde{s}(\tau_j,\zeta_0)))H_{r,\tilde{s}(\tau_j,\zeta_0)}(\tilde{x}(t,\zeta_0))dt\right)$$

$$+\mathbb{E}(u(\tau-\tau_{j+1},\tilde{x}(\tau_{j+1},\zeta_0),\tilde{s}(\tau_j,\zeta_0))-u(\tau-\tau_{j+1},\tilde{x}(\tau_{j+1},\zeta_0),\tilde{s}(\tau_{j+1},\zeta_0)))$$

$$=\mathbb{E}\left(\int_{\tau_j}^{\tau_{j+1}}\sum_{r\in S}(u(\tau-t,\tilde{x}(t,\zeta_0),r)-u(\tau-t,\tilde{x}(t,\zeta_0),\tilde{s}(\tau_j,\zeta_0)))H_{r,\tilde{s}(\tau_j,\zeta_0)}(\tilde{x}(t,\zeta_0))dt\right)$$

$$+\mathbb{E}\left(\sum_{r\in S}(u(\tau-\tau_{j+1},\tilde{x}(\tau_{j+1},\zeta_0)),\tilde{s}(\tau_j,\zeta_0)-u(\tau-\tau_{j+1},\tilde{x}(\tau_{j+1},\zeta_0),r)\right)\Pi_{r,\tilde{s}(\tau,\zeta_0j)}(\tau_j,\tau_{j+1}).$$

By eq. (20), we have that

$$\Pi(\tau_j,\tau_{j+1})=I+\int_{\tau_j}^{\tau_{j+1}}H(\tilde{x}(t),\zeta_0)\Pi(\tau_j,t)dt,$$

where I is the identity matrix.

Since H is bounded, it follows from Gronwall's lemma that Π is bounded and

$$\|\Pi(\tau_j,t)-I\|\le\kappa|t-\tau_j|$$

for some constant κ, where $\|\cdot\|$ is a norm on the space of matrices. For instance we may take the supremum of the modulus of the coefficients of a matrix. Hence,

$$\|\Pi(\tau_j,\tau_{j+1})-I-\int_{\tau_j}^{\tau_{j+1}}H(\tilde{x}(t,\zeta_0))dt\|=\|\int_{\tau_j}^{\tau_{j+1}}H(\tilde{x}(t,\zeta_0))(I-\Pi(\tau_j,t))dt\|$$

$$\le\kappa|\tau_{j+1}-\tau_j|^2,$$

for another constant κ. In particular, for $r\ne\tilde{s}(\tau_j)$:

$$|\Pi_{r,\tilde{s}(\tau_j)}(\tau_j,\tau_{j+1})-\int_{\tau_j}^{\tau_{j+1}}H_{r,\tilde{s}(\tau_j,\zeta_0)}(\tilde{x}(t,\zeta_0))dt\|\le\kappa|\tau_{j+1}-\tau_j|^2,$$

Moreover,

$$|\tilde{x}(\tau_{j+1,\zeta_0})-\tilde{x}(t,\zeta_0)|\le\kappa|\tau_{j+1}-t|,$$

since its derivative is $V_{\tilde{s}_j}(\tilde{x}(t))$, which is bounded. We have shown in (B.1) that u is Lipschitz with respect to x and t, thus

$$|u(\tau-t,\tilde{x}(t,\zeta_0),r)-u(\tau-\tau_{j+1},\tilde{x}(\tau_{j+1},\zeta_0),r)|\le\kappa(\|\varphi\|_{0,1}+1)|\tau_{j+1}-t|. \quad \text{(B.2)}$$

Now we have the following estimate

$$|\mathbb{E}(u(\tau - \tau_j, \tilde{x}(\tau_j, \zeta_0), \tilde{s}(\tau_j, \zeta_0)) - u(\tau - \tau_{j+1}, \tilde{x}(\tau_{j+1}, \zeta_0), \tilde{s}(\tau_{j+1}, \zeta_0)))|$$

$$= \left| \mathbb{E} \left(\int_{\tau_j}^{\tau_{j+1}} \sum_{r \in S} (u(\tau - t, \tilde{x}(t, \zeta_0), r) - u(\tau - t, \tilde{x}(t, \zeta_0), \tilde{s}(\tau_j, \zeta_0))) H_{r, \tilde{s}(\tau_j, \zeta_0)}(\tilde{x}(t, \zeta_0)) dt \right) \right.$$

$$+ \mathbb{E} \left(\sum_{r \in S} (u(\tau - \tau_{j+1}, \tilde{x}(\tau_{j+1}, \zeta_0), \tilde{s}(\tau_j, \zeta_0)) - u(\tau - \tau_{j+1}, \tilde{x}(\tau_{j+1}, \zeta_0), r)) \Pi_{r, \tilde{s}(\tau_j, \zeta_0)}(\tau_j, \tau_{j+1}) \right)$$

$$\leq \left| \mathbb{E} \left(\int_{\tau_j}^{\tau_{j+1}} \sum_{r \in S} (u(\tau - t, \tilde{x}(t, \zeta_0), r) - u(\tau - t, \tilde{x}(t, \zeta_0), \tilde{s}(\tau_j, \zeta_0))) \right. \right.$$

$$+ (u(\tau - \tau_{j+1}, \tilde{x}(\tau_{j+1}, \zeta_0)), \tilde{s}(\tau_j, \zeta_0) - u(\tau - \tau_{j+1}, \tilde{x}(\tau_{j+1}, \zeta_0), r)) H_{r, \tilde{s}(\tau_j), \zeta_0}(\tilde{x}(t, \zeta_0)) dt \left. \right) \right|$$

$$+ \kappa |\tau_{j+1} - \tau_j|^2$$

$$\leq \kappa(\|\varphi\|_{0,1} + 1)|\tau_{j+1} - \tau_j|^2$$

and so

$$|\mathbb{E}(\varphi(\mathbf{x}(\tau, \zeta_0), s(\tau, \zeta_0)) - \mathbb{E}(\varphi(\tilde{x}(\tau, \zeta_0), \tilde{s}(\tau, \zeta_0))| \leq \sum_{j=0}^{M-1} \kappa|(\|\varphi\|_{0,1} + 1)\tau_{j+1} - \tau_j|^2$$

$$\leq C\kappa\tau(\|\varphi\|_{0,1} + 1)/M.$$

The final error estimate is obtained by taking φ depending only on x and integrating with respect to μ_0:

$$\left| \int_{[0,1]} \varphi(x)\mu_t^M(dx) - \int_B \varphi(x)\mu_t(dx) \right|$$

$$= \left| \int_{[0,1]} \mathbb{E}(\varphi(\mathbf{x}(\tau, \zeta_0))) - \mathbb{E}(\varphi(\tilde{x}(\tau, \zeta_0))) d\mu_0(\zeta_0) \right|$$

$$\leq C\kappa\tau(\|\varphi\|_{0,1} + 1)/M$$

When φ is only Lipschitz, we obtain the same result by choosing a sequence of C^1 functions which converges uniformly to φ and whose Lipschitz constants do not exceed L. This finishes the proof of the first statement. It is classical that this implies that the left-handed side converges to zero when

$$\max_{j=0,\dots,M-1} |\tau_{j+1} - \tau_j| \to 0$$

for any uniformly continuous and bounded function φ. We conclude thanks to the Portemanteau theorem [3].

33

Remark 4. When φ is only borelian bounded but μ_0 has a C^1 and Lipschitz density f_0, the proof can be adapted. We observe that the only point where the Lipschitz property of φ is used is to obtain (B.1) which is used to derive (B.2). To replace (B.2), we can instead write the explicit formula for $\mathbb{E}(\varphi(\mathbf{x}(t), s(t))$ and $\mathbb{E}(\varphi(\tilde{x}(t), \tilde{s}(t))$. Since the flows $\mathbf{\Phi}_s$, $s \in S$ are C^1 diffeomorphisms that depend smoothly on time, we see that (B.2) still holds, but on the left-handed side, $\kappa(\|\varphi\|_{0,1} + 1)$ must be replaced by a constant which depends on $\|f_0\|_{0,1}$ and the supremum of φ.

References

[1] A. Abate, L. Cardelli, M. Kwiatkowska, L. Laurenti, and B. Yordanov. Experimental biological protocols with formal semantics, *International Conference on Computational Methods in Systems Biology*, pp. 165–182. Springer, 2018.

[2] D.F. Anderson, B. Ermentrout, and P.J. Thomas. Stochastic Representations of Ion Channel Kinetics and Exact Stochastic Simulations of Neuronal Dynamics, *Journal for Computational Neuroscience*, 38(1):67-82, 2015.

[3] P. Billingsley. *Convergence of Probability Measures, Second Edition*. Wiley series in Probability and Statistics, 1999.

[4] L. Cai, N. Friedman, and X.S. Xie. Stochastic protein expression in individual cells at the single molecule level. *Nature*, 440(7082):358, 2006.

[5] A. Crudu, A. Debussche, and O. Radulescu. Hybrid stochastic simplifications for multiscale gene networks. *BMC Systems Biology*, 3(1):89, 2009.

[6] A. Crudu, A. Debussche, A. Muller, and O. Radulescu. Convergence of stochastic gene networks to hybrid piecewise deterministic processes. *Annals of Applied Probability*, 22:1822–1859, 2012.

[7] M. Davis. *Markov Models and Optimization*. Chapman and Hall, 1993.

[8] A. Eldar and M.B. Elowitz. Functional roles for noise in genetic circuits. *Nature*, 467(7312):167, 2010.

[9] M.B. Elowitz, A.J. Levine, E.D. Siggia, and P. S. Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002.

[10] M.L Ferguson, D. Le Coq, M. Jules, S. Aymerich, O. Radulescu, N. Declerck, and C.A. Royer. Reconciling molecular regulatory mechanisms with noise patterns of bacterial metabolic promoters in induced and repressed states. *Proceedings of the National Academy of Sciences USA*, 109:155, 2012.

[11] P.B. Gupta, C.M. Fillmore, G. Jiang, S.D. Shapira, K. Tao, C. Kuperwasser, and E.S. Lander. Stochastic state transitions give rise to phenotypic equilibrium in populations of cancer cells. *Cell*, 146(4):633–644, 2011.

[12] U. Herbach, A. Bonnaffoux, T. Espinasse, and O. Gandrillon. Inferring gene regulatory networks from single-cell data: a mechanistic approach. *BMC Systems Biology*, 11(1):105, 2017.

[13] G.C.P. Innocentini, M. Forger, O. Radulescu, and F. Antoneli. Protein synthesis driven by dynamical stochastic transcription. *Bulletin of Mathematical Biology*, 78(1):110–131, 2016.

[14] G.C.P. Innocentini, A. Hodgkinson, and O. Radulescu. Time dependent stochastic mRNA and protein synthesis in piecewise-deterministic models of gene networks. *Frontiers in Physics*, 6:46, 2018.

[15] G.C.P. Innocentini, M. Forger, A.F. Ramos, O. Radulescu, and J.E.M. Hornos. Multimodality and flexibility of stochastic gene expression. *Bulletin of Mathematical Biology*, 75(12):2600–2630, 2013.

[16] G.C.P. Innocentini, A. Hodgkinson, F. Antoneli, and O. Radulescu. Effective computational methods for hybrid stochastic gene networks. *Computational Methods in Systems Biology: 17th International Conference, CMSB 2019, Trieste, Italy, September 18–20, 2019, Proceedings*, LNBI 11773: 60, Springer Nature, 2019.

[17] P. Kurasov, A. Lück, D. Mugnolo, and V. Wolf. Stochastic hybrid models of gene regulatory networks – a PDE approach. *Mathematical Biosciences*, 305:170–177, 2018.

[18] Y. Ting Lin and N.E. Buchler. Efficient analysis of stochastic gene dynamics in the non-adiabatic regime using piecewise deterministic markov processes. *Journal of The Royal Society Interface*, 15(138):20170804, 2018.

[19] R.W. MacCormack. The Effect of Viscosity in Hypervelocity Impact Cratering *Frontiers of Computational Fluid Dynamics* 2002, pp. 27-43, 2001.

[20] A. Raj, C.S. Peskin, D. Tranchina, D.Y. Vargas, and S. Tyagi. Stochastic mRNA synthesis in mammalian cells. *PLoS Biology*, 4(10):e309, 2006.

[21] B.S. Razooky, A. Pai, K. Aull, I.M. Rouzine, and L.S. Weinberger. A hardwired HIV latency program. *Cell*, 160(5):990–1001, 2015.

[22] M.G. Riedler. Almost sure convergence of numerical approximations for piecewise deterministic Markov processes. *Journal of Computational and Applied Mathematics*, 239:50–71, 2013.

[23] A. Singh and J.P. Hespanha. Stochastic hybrid systems for studying biochemical processes. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368 : 4995–5011, 2010.

[24] D. Talay. Discrétisation d'une équation différentielle stochastique et calcul approché d'espérances de fonctionnelles de la solution. *RAIRO Modél. Math. Anal. Numér.*, 20:141–179, 1986.

[25] K. Tantale, F. Mueller, A. Kozulic-Pirher, A. Lesne, J.-M. Victor, M.-C. Robert, S. Capozi, R. Chouaib, V. Bäcker, J. Mateos-Langerak, et al. A single-molecule view of transcription reveals convoys of RNA polymerases and multi-scale bursting. *Nature Communications*, 7:12248, 2016.

[26] M. Thattai and A. Oudenaarden. Stochastic gene expression in fluctuating environments. *Genetics*, 167(1):523–530, 2004.

[27] P. Thomas, N. Popović, and R. Grima. Phenotypic switching in gene regulatory networks. *Proceedings of the National Academy of Sciences USA*, 111(19):6994–6999, 2014.

[28] S. Zeiser, U. Franz, O. Wittich, and V. Liebscher. Simulation of genetic networks modelled by piecewise deterministic Markov processes. *IET Systems Biology*, 2(3):113–135, 2008.