Numerical evaluation of ODE solutions by Monte Carlo enumeration of Butcher series

Guillaume Penent[*] Nicolas Privault[†]
Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University
21 Nanyang Link, Singapore 637371

August 25, 2022

Abstract

We present an algorithm for the numerical solution of ordinary differential equations by random enumeration of the Butcher trees used in the implementation of the Runge-Kutta method. Our Monte Carlo scheme allows for the direct numerical evaluation of an ODE solution at any given time within a certain interval, without iteration through multiple time steps. In particular, this approach does not involve a discretization step size, and it does not require the truncation of Taylor series.

Keywords: Ordinary differential equations, Runge-Kutta method, Butcher series, random trees, Monte Carlo method.
Mathematics Subject Classification (2020): 65L06, 34A25, 34-04, 05C05, 65C05.

1 Introduction

Butcher series [But63], [But16] are a powerful tool used to represent the Taylor expansions appearing in the Runge-Kutta methods for the numerical solution of ordinary differential equations (ODEs), see Chapters 4-6 of [DB02], and [MMMKV17] for a recent review starting from the early work of [Cay57]. Those series are making use rooted tree enumeration, which have many applications ranging from geometric numerical integration to stochastic differential equations, see for instance [HLW06] and references therein, and [Gub10], [BHZ19],

[*] pene0001@e.ntu.edu.sg
[†] nprivault@ntu.edu.sg

[Fos21] for the use of decorated trees for stochastic partial differential equations and their connections with the Butcher-Connes-Kreimer Hopf algebra [CK99].

It is known that the solution $y(t)$ of the autonomous d-dimensional ODE system

$$\begin{cases} y'(t) = f(y(t)) \\ y(0) = y_0 \in \mathbb{R}^d, \qquad t \in \mathbb{R}_+, \end{cases} \tag{1.1}$$

where $f(y) = (f_1(y), \ldots, f_d(y))$ is a smooth $\mathbb{R}^d$-valued function of y in a domain of $\mathbb{R}^d$, can be expressed as

$$y(t) = y_0 + tf(y_0) + \frac{t^2}{2} f'(y_0)f(y_0) + \frac{t^3}{6} f'(y_0)f(y_0)'f(y_0) + \frac{t^3}{6} f''(y_0)[f(y_0), f(y_0)] + \cdots \tag{1.2}$$

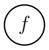
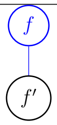
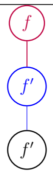
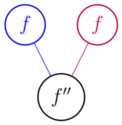
where we use the notation

$$f'(y_0)f(y_0) := \left(\sum_{i_2=1}^d \frac{\partial f_{i_1}}{\partial x_{i_2}}(y_0) f_{i_2}(y_0) \right)_{i_1=1,\ldots,d}, \tag{1.3}$$

$$f'(y_0)f'(y_0)f(y_0) := \left(\sum_{i_2,i_3=1}^d \frac{\partial f_{i_1}}{\partial x_{i_2}}(y_0) \frac{\partial f_{i_2}}{\partial x_{i_3}}(y_0) f_{i_3}(y_0) \right)_{i_1=1,\ldots,d}, \tag{1.4}$$

$$f''(y_0)[f(y_0), f(y_0)] := \left(\sum_{i_2,i_3=1}^d \frac{\partial^2 f_{i_1}}{\partial x_{i_2} \partial x_{i_3}}(y_0) f_{i_2}(y_0) f_{i_3}(y_0) \right)_{i_1=1,\ldots,d}, \tag{1.5}$$

etc. In addition, the expansion (1.2) can be coded and enumerated using the following sequence of Butcher trees, which can also be represented without referring to the derivatives of f, see for example Table 1.1 page 53 of [HLW06].

Order	Coefficient	Butcher tree	Order	Coefficient	Butcher tree	Order	Coefficient	Butcher tree
0	y_0	$\emptyset$	1	$f(y_0)$		2	$f'f(y_0)$	
3	$f'f'f(y_0)$		3	$f''[f,f](y_0)$				

The numerical evaluation of Butcher series involves tree enumeration up to a certain order that determines the level of precision of the algorithm.

In this paper, we propose an alternative method to the numerical evaluation of ODE solutions, based on a random enumeration of Butcher trees by Monte Carlo simulation. Probabilistic methods based on the Feynman-Kac formula provide alternatives to finite difference schemes, and have been successfully applied to the solution of partial differential equations. In particular, stochastic branching mechanisms have been used to represent the solutions of partial differential equations in [Sko64], [INW69], [NS69], [McK75], [LM96], [CLM08]. This branching argument has been recently extended in [HLOT$^+$19] to the treatment of polynomial non-linearities in gradient terms, see also [PP22] for nonlocal and fractional PDEs.

In Theorem 4.2, under suitable integrability conditions we express ODE solutions as the expected value of a functional of random Butcher trees which encode nonlinearities. Then in Proposition 4.3 we provide sufficient conditions ensuring that the representation formula of Theorem 4.2 holds at any time within certain interval. Numerical values of ODE solutions can be computed beyond that initial interval by iterating the method and by piecing together the solutions obtained on adjacent intervals. As noted before Proposition 4.3, the integrability conditions are stronger in higher dimensions.

This approach complements the use of the Feynman-Kac formula for the numerical estimation of the solutions of partial differential equations, see also [SH21] for a different approach to the Feynman-Kac representation of ODE solutions. Other links between Butcher trees and probability theory have been pointed out in [Maz04], see also [Ski92] for the numerical solution of ODEs as an inference problem by Bayesian techniques.

In comparison with integrators in the Runge-Kutta method, our method requires the evaluation of f and its partial derivatives up to any order, whereas the Runge-Kutta method uses only f. The complexity of our algorithm grows linearly with dimension, as d trees are used to solve a d-dimensional ODE system, whereas the complexity of finite difference methods is generally polynomial in the dimension d, depending on the order chosen in the truncation of (1.2). Complexity in time can be estimated via the mean length of binary trees, which grows exponentially in time independently of dimension $d \geq 1$ in the case of exponentially distributed branch lifetimes, as noted in Section 4. On the other hand, our method can be used to exactly approximate the solution at any given time on a (possibly infinite) time interval. Our approach also benefits from the advantages of Monte Carlo estimators whose computation can be paralleled straightforwardly.

In Section 2 we introduce the construction of coding trees that will be used for the numerical solution of ODEs. Section 4 presents the probabilistic representation formula of

ODE solutions obtained by the random generation of coding trees. In Section 5 we consider examples and in Section 6 we describe the correspondance between Butcher trees and coding trees, namely we show how any Butcher tree can be recovered by performing a depth first search on the corresponding coding tree, showing how Butcher series can be rewritten as series of expected values. Section 7 considers numerical applications, and the appendix contains the corresponding computer codes in Maple, Mathematica and Python.

2 Codes and mechanism

This section introduces the random coding trees used for the probabilistic representation of ODE solutions. We consider a multidimensional autonomous system of the form

$$\begin{cases} y_1(t) = y_1(0) + \displaystyle\int_0^t f_1(y_1(s), \ldots, y_d(s))ds \\ y_2(t) = y_2(0) + \displaystyle\int_0^t f_2(y_1(s), \ldots, y_d(s))ds \\ \qquad \vdots \\ y_d(t) = y_d(0) + \displaystyle\int_0^t f_d(y_1(s), \ldots, y_d(s))ds, \end{cases} \tag{2.1}$$

$t \in \mathbb{R}_+$, where f_i is a smooth Lipschitz function defined on a domain of $\mathbb{R}_+ \times \mathbb{R}^d$, $i = 1, \ldots, d$. In order to solve (2.1) iteratively, we can start by expanding $f_i(y_1(s), \ldots, y_d(s))$ as

$$f_i(y_1(s), \ldots, y_d(s)) = f_i(y_1(0), \ldots, y_d(0)) + \sum_{j=1}^d \int_0^s f_j(y_1(u), \ldots, y_d(u)) \partial_j f_i(y_1(u), \ldots, y_d(u)) du \tag{2.2}$$

by differentiating $v(s) := f_i(y_1(s), \ldots, y_d(s))$, where we use the notation $\partial_j f_i = \partial f_i / \partial y_j$, $i, j = 1, \ldots, d$. In the sequel, given g a function from $\mathbb{R}^d$ into $\mathbb{R}$, we let g^* denote the mapping

$$\begin{aligned} g^* : (\mathbb{R}^d)^{\mathbb{R}_+} &\longrightarrow \mathbb{R}^{\mathbb{R}_+} \\ (t \mapsto y(t)) &\longmapsto g^*(y) := (t \mapsto g(y(t))), \end{aligned} \tag{2.3}$$

where $(\mathbb{R}^d)^{\mathbb{R}_+}$ represents the set of functions from $\mathbb{R}_+$ to $\mathbb{R}^d$. In order to formalize and extend the iteration initiated in (2.2), we introduce the following definitions. In the sequel we let Id_i denotes the *i-th* canonical projection from $(\mathbb{R}^d)^{\mathbb{R}_+}$ to $\mathbb{R}^{\mathbb{R}_+}$ with $\mathrm{Id}_i(y_1, \ldots, y_d) = y_i$, $i = 1, \ldots, d$.

Definition 2.1 *We let $\mathcal{C}$ denote the set of functions from $(\mathbb{R}^d)^{\mathbb{R}_+}$ to $\mathbb{R}^{\mathbb{R}_+}$ called codes, defined as*

$$\mathcal{C} := \left\{ \mathrm{Id}_i, \ \left(\partial_1^{i_1} \cdots \partial_d^{i_d} f_i \right)^* \ : \ i_1, \ldots, i_d \geq 0, \ i = 1, \ldots, d \right\}$$

By (2.3), the elements of $\mathcal{C}$ are operators mapping a function $h \in (\mathbb{R}^d)^{\mathbb{R}_+}$ to another function $c(h) \in \mathbb{R}^{\mathbb{R}_+}$. We also consider a mapping $\mathcal{M}$, called the *mechanism*, defined on $\mathcal{C}$ by matching a *code* $c \in \mathcal{C}$ to a set $\mathcal{M}(c)$ of code tuples.

Definition 2.2 *The mechanism $\mathcal{M}$ is defined by $\mathcal{M}(\mathrm{Id}_i) = \{f_i\}$ and*

$$\mathcal{M}(g^*) = \left\{ (f_1^*, (\partial_1 g)^*), (f_2^*, (\partial_2 g)^*), \ldots, (f_d^*, (\partial_d g)^*) \right\}, \tag{2.4}$$

for g a smooth function from $\mathbb{R}^d$ into $\mathbb{R}$.

In the next key lemma we show that $c(y)$ satisfies a system of equations indexed by $c \in \mathcal{C}$.

Lemma 2.3 *For any code $c \in \mathcal{C}$ we have*

$$c(y)(t) = c(y)(0) + \sum_{Z \in \mathcal{M}(c)} \int_0^t \prod_{z \in Z} z(y)(s) ds, \qquad t \in \mathbb{R}_+. \tag{2.5}$$

Proof. When $c = \mathrm{Id}_i$ we have

$$c(y)(t) = y_i(t) = y_i(0) + \int_0^t f_i(y_1(s), \ldots, y_d(s)) ds = y_i(0) + \int_0^t f_i^*(y_1, \ldots, y_d)(s) ds,$$

hence (2.5) holds since $\mathcal{M}(\mathrm{Id}_i) = \{f_i^*\}$, $i = 1, \ldots, d$. When $c = g^* \in \mathcal{C}$ with $c \neq \mathrm{Id}_i$, the equation

$$g(y_1(t), \ldots, y_d(t)) = g(y_1(0), \ldots, y_d(0)) + \sum_{j=1}^d \int_0^t f_j(y_1(s), \ldots, y_d(s)) \partial_j g(y_1(s), \ldots, y_d(s)) ds$$

satisfied by $g^*(y)(t)$ reads

$$g^*(y)(t) = g^*(y)(0) + \sum_{j=1}^d \int_0^t f_j^*(y)(s) (\partial_j g)^*(y)(s) ds,$$

and (2.5) follows by the definition (2.4) of $\mathcal{M}$. □

We note that for any $g^* \in \mathcal{C}$ it is always possible to compute $g^*(y)(0)$ by applying the code g^* to the solution y of the ODE and then evaluating it at time 0 as $g^*(y)(0) = g(y_0)$. In particular, the full knowledge of the function y is not necessary to compute $g^*(y)(0)$.

Example - One-dimensional autonomous ODE

Consider the solution $y(t)$ of the one-dimensional ODE

$$y(t) = y_0 + \int_0^t f(y(s))ds, \qquad t \in \mathbb{R}_+,$$

where $f(y(s))$ is expanded as

$$f(y(s)) = f(y_0) + \int_0^s f(y(u))f'(y(u))du, \qquad s \in \mathbb{R}_+.$$

Here, g^* denotes the mapping

$$\begin{aligned} g^* : \mathbb{R}^{\mathbb{R}_+} &\longrightarrow \mathbb{R}^{\mathbb{R}_+} \\ (t \mapsto y(t)) &\longmapsto g^*(y) := (t \mapsto g(y(t))), \end{aligned}$$

for g a smooth function from $\mathbb{R}$ to $\mathbb{R}$, the set of codes is given by

$$\mathcal{C} := \left\{ \mathrm{Id}, \ \left(f^{(k)}\right)^*, \ \ k \geq 0 \right\},$$

where $\left(f^{(k)}\right)^*$, $k \geq 0$, denotes the operator acting as

$$\left(f^{(k)}\right)^*(y)(s) := f^{(k)}(y(s)), \qquad s \in \mathbb{R}_+,$$

and the mechanism $\mathcal{M}$ is given by $\mathcal{M}(\mathrm{Id}) = \{f^*\}$ and $\mathcal{M}(g^*) = \left\{(f^*, (g')^*)\right\}$.

Example - Non-autonomous ODE

Consider the non-autonomous ODE

$$y(t) = y_0 + \int_0^t f_2(s, y(s))ds, \qquad t \in \mathbb{R}_+, \tag{2.6}$$

where f_2 is a smooth Lipschitz function defined on a domain of $\mathbb{R}_+ \times \mathbb{R}$. This ODE can be rewritten as the system

$$\begin{cases} y_1(t) = t = y_1(0) + \displaystyle\int_0^t f_1(y_1(s), y_2(s))ds \\ y_2(t) = y(t) = y_2(0) + \displaystyle\int_0^t f_2(y_1(s), y_2(s))ds \end{cases} \tag{2.7}$$

by taking $f_1 \equiv 1$ and $y_1(s) = s$, $s \in \mathbb{R}_+$. Here, the set of codes satisfies

$$\mathcal{C} : \ = \ \left\{ \mathrm{Id}_1, \mathrm{Id}_2, \ \left(\partial_1^{i_1}\partial_2^{i_2} f_1\right)^*, \ \left(\partial_1^{j_1}\partial_2^{j_2} f_1\right)^*, \ \ i_1, i_2, j_1, j_2 \geq 0 \right\}$$

$$= \ \big\{ \mathrm{Id}_1, \mathrm{Id}_2, 0, 1, \big(\partial_1^{j_1}\partial_2^{j_2} f_1\big)^*, \quad j_1, j_2 \geq 0 \big\},$$

where 0 and 1 denote constant functions. The mechanism $\mathcal{M}$ is given by $\mathcal{M}(\mathrm{Id}_1) = \{f_1^*\} = \{1\}$, $\mathcal{M}(\mathrm{Id}_2) = \{f_2^*\}$, and $\mathcal{M}(g^*) = \big\{(f_1^*, (\partial_1 g)^*), (f_2^*, (\partial_2 g)^*)\big\} = \big\{(1, (\partial_1 g)^*), (f_2^*, (\partial_2 g)^*)\big\}$ for g a smooth function from $\mathbb{R}^2$ into $\mathbb{R}$.

More generally, any non autonomous system can be transformed into an autonomous system by addition of a dimension. In particular, any higher order ordinary differential equation of the form

$$y^{(d)}(t) = f\big(t, y(t), y'(t), \ldots, y^{(d-1)}(t)\big)$$

can be written as a system of the form (2.1) by taking $f_1(y_1, \ldots, y_d) \equiv 1$, $f_i(y_1, \ldots, y_d) := y_{i+1}$, $i = 2, \ldots, d-1$ and $f_d(y_1, \ldots, y_d) := f(y_1, \ldots, y_d)$, with $y_1(t) = t$ and $y_i(t) = y^{(i-1)}(t)$, $i = 2, \ldots, d$.

3 Coding trees

For each code $c \in \mathcal{C}$ we denote by I_c a uniformly distributed random variable on $\mathcal{M}(c)$. For example, when $c = g^*$, since $\mathcal{M}(g^*) = \big\{(f_1^*, (\partial_1 g)^*), (f_2^*, (\partial_2 g)^*), \ldots, (f_d^*, (\partial_d g)^*)\big\}$, we have

$$\mathbb{P}\big(I_{g^*} = (f_i^*, (\partial_i g)^*)\big) = \frac{1}{d}, \qquad i = 1, \ldots, d.$$

In the sequel we will use the notation

$$q_c(b) := \mathbb{P}(I_c = b), \qquad b \in \mathcal{M}(c), \quad c \in \mathcal{C}.$$

In addition, we consider a probability density function $\rho : \mathbb{R}_+ \to (0, \infty)$ and

- an i.i.d. family $(\tau^{i,j})_{i,j\geq 1}$ of random variables with distribution $\rho(t)dt$ on $\mathbb{R}_+$,
- for each $c \in \mathcal{C}$, an i.i.d. family $(I_c^{i,j})_{i,j\geq 1}$ of discrete random variables, with

$$\mathbb{P}\big(I_c^{i,j} = b\big) = q_c(b) > 0, \qquad b \in \mathcal{M}(c),$$

where the sequences $(\tau^{i,j})_{i,j\geq 1}$ and $(I_c^{i,j})_{c\in\mathcal{C},i,j\geq 1}$ are assumed to be mutually independent. For every $n \geq 1$ we also consider an injection $\pi_n : \mathbb{N}^n \to \mathbb{N}$.

Let $t > 0$. For each $i \in \{1, \ldots, d\}$ we construct a random tree starting from an initial particle labelled $\bar{1} := (1)$ bearing the code Id_i at time 0, which lives up to a random time $\tau^{1,1}$ distributed according to ρ. If $\tau^{1,1} > t$, the branching process stops. Otherwise, if

$\tau^{1,1} \leq t$, a new particle with label $(1,1)$ is created, and bears the code f_i^* since $\mathcal{M}(\mathrm{Id}_i) = \{f_i^*\}$, and independently follows the same pattern as the first one. This new branch lives during the time $\tau^{2,\pi_2(1,1)}$. If $\tau^{(1,1)} + \tau^{2,\pi_2(1,1)} > t$ then the tree stops branching, otherwise, if $\tau^{1,1} + \tau^{2,\pi_2(1,1)} \leq t$, the particle branches in two branches $(f_i^*, (\partial_j f_i)^*)$ chosen uniformly in $\mathcal{M}(f_i^*) = \big\{(f_1^*, (\partial_1 f_i)^*), (f_2^*, (\partial_2 f_i)^*), \ldots, (f_d^*, (\partial_d f_i)^*)\big\}$.

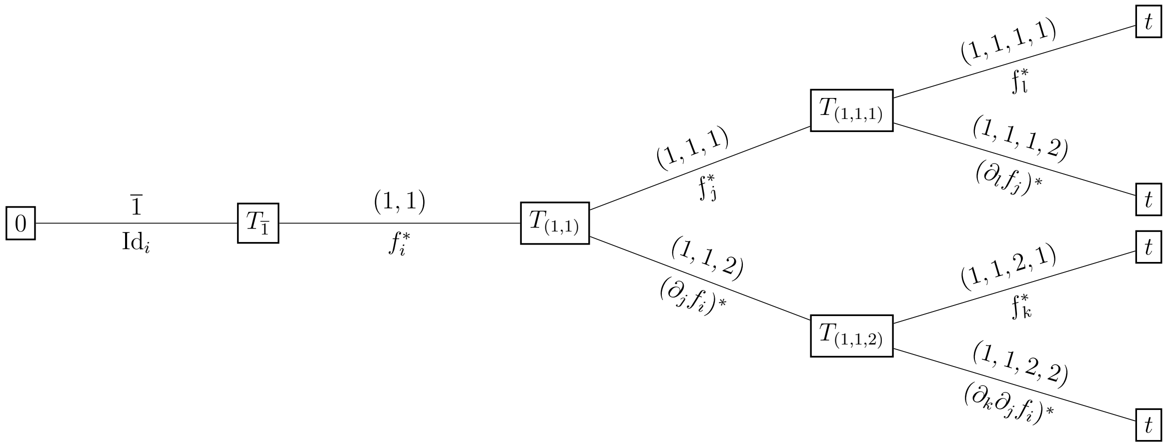
More generally, a particle with code $c \in \mathcal{C}$ at the generation $n \geq 1$ is assigned a label of the form $\bar{k} = (1, k_2, \ldots, k_n) \in \mathbb{N}^n$, while its parent label is $\bar{k}- := (1, k_2, \ldots, k_{n-1})$. The birth time of particle $\bar{k}$ is denoted by $T_{\bar{k}-}$, and its lifetime $\tau^{n,\pi_n(\bar{k})}$ is the element of index $j = \pi_n(\bar{k})$ in the i.i.d. sequence $(\tau^{n,j})_{j\geq 1}$. If $T_{\bar{k}} := \mathrm{T}_{\bar{k}-} + \tau^{n,\pi_n(\bar{k})} < t$, we draw a sample $I_c^{n,\pi_n(\bar{k})} = (c_1, \ldots, c_l)$ uniformly in $\mathcal{M}(c)$ with $l \in \{1,2\}$, and the particle $\bar{k}$ branches into $\big|I_c^{n,\pi_n(\bar{k})}\big| \in \{1,2\}$ offsprings at generation $(n+1)$, which are labeled by $\bar{k} = (1, \ldots, k_n, j)$, $j = 1, \ldots, \big|I_c^{n,\pi_n(\bar{k})}\big|$. The particle with label ending with an integer j will carry the code c_j. Finally, the code of particle $\bar{k}$ will be denoted by $c_{\bar{k}} \in \mathcal{C}$. The death time of the particle $\overline{k}$ is $T_{\overline{k}}$, and its birth time is $T_{\overline{k}-}$.

Definition 3.1 *We denote by $\mathcal{T}_{t,c}$ the random tree constructed from the above argument started from any code $c \in \mathcal{C}$.*

The family $(\mathcal{T}_{t,\mathrm{Id}_i})_{i=1,\ldots,d}$ of trees will be used for the stochastic representation of the solution $y(t) = (y_1(t), \ldots, y_d(t))$ of the ODE system (2.1), while the trees $\mathcal{T}_{t,c}$ will be used for the stochastic representation of $c(y)(t)$. The next table summarizes the notation introduced so far.

Object	Notation
Initial time	0
Tree ending at time t with initial code c	$\mathcal{T}_{t,c}$
Particle (or label) of generation $n \geq 1$	$\overline{k} = (1, k_2, \ldots, k_n)$
First branching time	$T_{\overline{1}}$
Birth time of a particle $\overline{k}$	$T_{\overline{k}-}$
Death time of a particle $\overline{k}$	$T_{\overline{k}}$
Lifespan of a particle	$T_{\overline{k}} - T_{\overline{k}-}$
Code of a particle $\overline{k}$	$c_{\overline{k}}$

The following graph represents a sample of the random tree $\mathcal{T}_{t,\mathrm{Id}_i}$, $i = 1, \ldots, d$.



4 Probabilistic representation of ODE solutions

In this section we define the random multiplicative functional which will be used to represent ODE solutions. We let $\overline{F}$ denote the tail distribution function of ρ, i.e.

$$\overline{F}(t) := \int_t^\infty \rho(u)du, \qquad t \in \mathbb{R}_+.$$

We denote by $\mathcal{K}^\circ$ the set of particles that do not live until time t, and by $\mathcal{K}^\partial$ the set of particles that die after time t.

Definition 4.1 *Given $\mathcal{T}_{t,c}$ a random coding tree started with the code $c \in \mathcal{C}$ and ending at time $t \geq 0$, we define the universal multiplicative functional $\mathcal{H}$ by*

$$\mathcal{H}(\mathcal{T}_{t,c}) := \prod_{\bar{k} \in \mathcal{K}^\circ} \frac{1}{q_{c_{\bar{k}}}(I_{c_{\bar{k}}})\rho(T_{\bar{k}} - T_{\bar{k}-})} \prod_{\bar{k} \in \mathcal{K}^\partial} \frac{c_{\bar{k}}(y)\big(0\big)}{\overline{F}(t - T_{\bar{k}-})}.$$

The next result gives the probabilistic representation of ODE solutions as an expected value over random coding trees.

Theorem 4.2 *Let $T > 0$ for which there exists $K_0 > 0$ such that*

$$\mathbb{E}\big[\big|\mathcal{H}(\mathcal{T}_{t,c})\big|\big] \leq K_0, \qquad c \in \mathcal{C}, \quad t \in [0,T].$$

Then, for any $c \in \mathcal{C}$ we have the probabilistic representation

$$c(y)(t) = \mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,c})\big], \qquad t \in [0,T], \tag{4.1}$$

where $y(t) = (y_1(t), \ldots, y_d(t))$ is the solution of the system of ODEs (2.1). In particular, taking $c = \mathrm{Id}_i$, we have

$$y_i(t) = y_{\mathrm{Id}_i}(t) = \mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,\mathrm{Id}_i})\big], \qquad t \in [0,T], \quad i = 1, \ldots, d.$$

Proof. For $c \in \mathcal{C}$ we let

$$y_c(t) := \mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,c})\big], \qquad t \in [0,T].$$

By conditioning on the first branching time $T_{\bar{1}}$, for all $i = 1, \ldots, d$ the first particle bearing the code Id_i branches at time $T_{\bar{1}}$ into a new particle bearing the code f_i^* as $\mathcal{M}(\mathrm{Id}_i) = \{f_i^*\}$, hence we have

$$\begin{aligned}
y_{\mathrm{Id}_i}(t) &= \mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,\mathrm{Id}_i})\mathbb{1}_{\{T_{\bar{1}}>t\}} + \mathcal{H}(\mathcal{T}_{t,\mathrm{Id}_i})\mathbb{1}_{\{T_{\bar{1}}\leq t\}}\big] \\
&= \mathbb{E}\left[\frac{y_i(0)}{\overline{F}(t)}\mathbb{1}_{\{T_{\bar{1}}>t\}}\right] + \mathbb{E}\left[\frac{y_{f_i^*}(t-T_{\bar{1}})}{\rho(T_{\bar{1}})}\mathbb{1}_{\{T_{\bar{1}}\leq t\}}\right] \\
&= y_i(0)\frac{\mathbb{P}(T_{\bar{1}}>t)}{\overline{F}(t)} + \int_0^t \frac{y_{f_i^*}(t-s)}{\rho(s)}\rho(s)ds \\
&= y_i(0) + \int_0^t y_{f_i^*}(s)ds, \qquad t \in [0,T].
\end{aligned}$$

Similarly, starting from any code $g^* \in \mathcal{C}$ different from Id_i, the particle branches at time $T_{\bar{1}}$ in d possible different ways into two particles with codes $(f_j^*, (\partial_j g)^*)$, $j = 1, \ldots, d$, hence we have

$$\begin{aligned}
y_{g^*}(t) &= \mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,g^*})\mathbb{1}_{\{T_{\bar{1}}>t\}} + \mathcal{H}(\mathcal{T}_{t,g^*})\mathbb{1}_{\{T_{\bar{1}}\leq t\}}\big] \\
&= \mathbb{E}\left[\frac{g^*(y_0)}{\overline{F}(t)}\mathbb{1}_{\{T_{\bar{1}}>t\}} + \mathbb{1}_{\{T_{\bar{1}}\leq t\}}\sum_{j=1}^d \mathbf{1}_{\{I_{g^*}=(f_j^*,(\partial_j g)^*)\}}\frac{y_{f_j^*}(t-T_{\bar{1}})y_{(\partial_j g)^*}(t-T_{\bar{1}})}{q_{g^*}(I_{g^*})\rho(T_{\bar{1}})}\right] \\
&= \mathbb{E}\left[\frac{g^*(y_0)}{\overline{F}(t)}\mathbb{1}_{\{T_{\bar{1}}>t\}} + \mathbb{1}_{\{T_{\bar{1}}\leq t\}}\sum_{j=1}^d q_{g_j^*}((f_j^*,(\partial_j g)^*))\frac{y_{f_j^*}(t-T_{\bar{1}})y_{(\partial_j g)^*}(t-T_{\bar{1}})}{q_{g_j^*}((f_j^*,(\partial_j g)^*))\rho(T_{\bar{1}})}\right] \\
&= \mathbb{E}\left[\frac{g^*(y_0)}{\overline{F}(t)}\mathbb{1}_{\{T_{\bar{1}}>t\}}\right] + \sum_{Z\in\mathcal{M}(g^*)}\mathbb{E}\left[\mathbb{1}_{\{T_{\bar{1}}\leq t\}}\frac{1}{\rho(T_{\bar{1}})}\prod_{z\in Z} y_z(t-T_{\bar{1}})\right] \\
&= g^*(y)(0) + \sum_{Z\in\mathcal{M}(g^*)}\int_0^t \frac{1}{\rho(s)}\left(\prod_{z\in Z} y_z(t-s)\right)\rho(s)ds, \\
&= g^*(y)(0) + \sum_{Z\in\mathcal{M}(g^*)}\int_0^t \prod_{z\in Z} y_z(s)ds, \qquad (4.2)
\end{aligned}$$

which yields the system of equations

$$y_c(t) = c(y)(0) + \sum_{Z\in\mathcal{M}(c)}\int_0^t \prod_{z\in Z} y_z(s)ds, \qquad t \in [0,T], \quad c \in \mathcal{C}. \tag{4.3}$$

By the Cauchy-Lipschitz theorem on the Banach space of sequences ℓ^∞, this system admits a unique maximal solution. We conclude by noting that from Lemma 2.3, the family of functions $(c(y))_{c\in\mathcal{C}}$ is the solution of the system (4.3), hence $(c(y))_{c\in\mathcal{C}} = (y_c)_{c\in\mathcal{C}}$, and

$$\mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,c})\big] = y_c(t) = c(y)(t), \qquad t \in [0, T].$$

□

In numerical applications the expected value $\mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,c})\big]$ in Theorem 4.2 is estimated as the average

$$\frac{1}{N}\sum_{k=1}^{N} \mathcal{H}(\mathcal{T}_{t,c})^{(k)}$$

where $\mathcal{H}(\mathcal{T}_{t,c})^{(1)}, \ldots, \mathcal{H}(\mathcal{T}_{t,c})^{(N)}$ are independent samples of $\mathcal{H}(\mathcal{T}_{t,c})$. In this case, the error on the estimate of $\mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,c})\big]$ from the Monte Carlo method can be estimated as the standard deviation

$$\left(\mathbb{E}\left[\left(\mathbb{E}\big[\mathcal{H}(\mathcal{T}_{t,c})\big] - \frac{1}{N}\sum_{k=1}^{N} \mathcal{H}(\mathcal{T}_{t,c})^{(k)}\right)^2\right]\right)^{1/2} = \frac{1}{\sqrt{N}}\sqrt{\mathrm{Var}\,\big[\mathcal{H}(\mathcal{T}_{t,c})\big]}.$$

Complexity of the algorithm

We note that the complexity of the algorithm grows linearly with the dimension d, as d trees $(\mathcal{T}_{t,\mathrm{Id}_i})_{i=1,\ldots,d}$ are used to generate the multidimensional solution $(y_1, \ldots, y_d)$, whereas the complexity of finite difference methods is generally polynomial in the dimension d, depending on the order chosen in the truncation of (1.2). On the other hand, all trees are at most binary regardless of the dimension d, as shown by the construction (2.4) of the mechanism $\mathcal{M}$.

Regarding complexity in time, let $\ell_c(t)$ denote the mean size of the random tree $\mathcal{T}_{t,c}$ generated until time $t \geq 0$, with, by construction, $\ell(t) := \ell_{\mathrm{Id}_i}(t)$ for all $i = 1, \ldots, d$, and $m(t) := \ell_c(t)$ for all $c \notin \{\mathrm{Id}_i\}_{i=1,\ldots,d}$. The same argument as in the proof of Theorem 4.2 shows that $(\ell(t), m(t))$ satisfies the sytem of integral equations

$$\begin{cases} \ell(t) = \displaystyle\int_t^\infty \rho(s)ds + \int_0^t \rho(s)m(t-s)ds, \\ m(t) = \displaystyle\int_t^\infty \rho(s)ds + 2\int_0^t \rho(s)m(t-s)ds. \end{cases}$$

When τ has the exponential density $\rho(s) = \lambda \mathrm{e}^{-\lambda s}$ with parameter $\lambda > 0$, this leads to the system

$$\begin{cases} \ell'(t) = -\lambda \mathrm{e}^{-\lambda t} - \lambda^2 \displaystyle\int_0^t \mathrm{e}^{-\lambda(t-s)} m(s)ds + \lambda m(t) = \lambda(m(t) - \ell(t)) \\ m'(t) = -\lambda \mathrm{e}^{-\lambda t} - 2\lambda^2 \displaystyle\int_0^t \mathrm{e}^{-\lambda(t-s)} m(s)ds + 2\lambda m(t) = \lambda m(t), \end{cases}$$

with solution

$$\ell(t) = \cosh(\lambda t), \quad m(t) = \mathrm{e}^{\lambda t} = \mathbb{E}\big[2^{N_t}\big], \qquad t \geq 0,$$

and where $(N_t)_{t\in\mathbb{R}_+}$ is a standard Poisson process with intensity $\lambda > 0$. We note that this estimate remains the same independently of dimension $d \geq 1$.

Integrability condition

The following proposition provides sufficient conditions ensuring that the representation formula (4.1) of Theorem 4.2 holds at any time within certain interval. In order to represent an ODE solution beyond that time interval we may reuse the numerical value obtained close to its boundary as new initial condition in order to represent the solution on an extended time interval. We note that the constant K in the next proposition depends on y_0 and f and its derivatives, and is independent of the system dimension $d \geq 1$. However, the integrability condition (4.4) is stronger in higher dimensions.

Proposition 4.3 *Assume that there exists $K > 0$ such that $c(y)(0) \leq K$ for any $c \in \mathcal{C}$, that the density function ρ is nonincreasing, and that*

$$\rho(T) \geq d, \quad K \leq \overline{F}(T). \tag{4.4}$$

Then, there exists $K(T) > 0$ such that

$$\mathbb{E}\big[\big|\mathcal{H}(\mathcal{T}_{s,c})\big|\big] \leq K(T), \quad c \in \mathcal{C}, \quad t \in [0,T]. \tag{4.5}$$

Proof. Under Condition (4.4), since $q_{\min} := \min_{c\in\mathcal{C}} q_c(I_c) = 1/d$, we have

$$\prod_{\bar{k}\in\mathcal{K}^\circ} \frac{1}{q_{\min}\rho(T_{\bar{k}} - T_{\bar{k}-})} \prod_{\bar{k}\in\mathcal{K}^\partial} \frac{K}{\overline{F}(t - T_{\bar{k}-})} \leq 1,$$

hence

$$\mathbb{E}\big[\big|\mathcal{H}(\mathcal{T}_{t,c})\big|\big] \leq \mathbb{E}_c\left[\prod_{\bar{k}\in\mathcal{K}^\circ} \frac{1}{q_{\min}\rho(T_{\bar{k}} - T_{\bar{k}-})} \prod_{\bar{k}\in\mathcal{K}^\partial} \frac{K}{\overline{F}(t - T_{\bar{k}-})}\right] \leq 1, \qquad t \in [0,T].$$

□

We note that trying to relax the integrability condition (4.4) by choosing a higher $\rho(T)$ will result into a smaller value of $\mathbb{E}[\tau]$, therefore increasing the time complexity of the algorithm. On the other hand, assuming that

$$S_c(t) := \mathbb{E}_c\left[\prod_{\bar{k}\in\mathcal{K}^\circ} \frac{1}{q_{c_{\bar{k}}}(I_{c_{\bar{k}}})\rho(T_{\bar{k}} - T_{\bar{k}-})} \prod_{\bar{k}\in\mathcal{K}^\partial} \frac{K}{\overline{F}(t - T_{\bar{k}-})}\right] < \infty,$$

yields the system of equations

$$\begin{cases} S_{\mathrm{Id}_i}(t) = K + \displaystyle\int_0^t S_{f_i^*}(s)ds \\ \\ S_{g^*}(t) = K + \displaystyle\sum_{i=1}^d \int_0^t S_{f_i^*}(s) S_{(\partial_i g)^*}(s)ds. \end{cases}$$

Since the corresponding trees $\mathcal{T}_{t,f_i^*}$, $\mathcal{T}_{t,c}$ have same random shape distribution, we have $S_c(t) = S_{f_i^*}(t)$ for all codes $c = g^* \in \mathcal{C}$ such that $g = \partial_1^{i_1} \cdots \partial_d^{i_d} f_i$, $i_1, \ldots, i_d \geq 0$, $i = 1, \ldots, d$. This gives the system

$$\begin{cases} S_{\mathrm{Id}_i}(t) = K + \displaystyle\int_0^t S_{f_i^*}(s)ds \\ \\ S_{f_i^*}(t) = K + d \times \displaystyle\int_0^t (S_{f_i^*}(s))^2 ds, \end{cases}$$

which can be solved as

$$S_{\mathrm{Id}_i}(t) = K - \frac{1}{d} \log(1 - Ktd), \qquad S_{f_i^*}(t) = \frac{K}{1 - Ktd}, \qquad i = 1, \ldots, d,$$

hence the finiteness of $S_c(t)$ holds at most until time

$$T < \frac{1}{Kd}. \tag{4.6}$$

Remark 4.4 *Although the non-autonomous ODE (2.6) can be treated using an autonomous 2-dimensional system of the form (2.7), the probabilistic representation (4.1) of its solution can also be obtained using a single random tree. For this, we expand $f_2(s, y(s))$ as*

$$f_2(s, y(s)) = f_2(0, y_0) + \int_0^s \big(\partial_0 f_2(u, y(u)) + f_2(u, y(u)) \partial_1 f_2(u, y(u))\big) du.$$

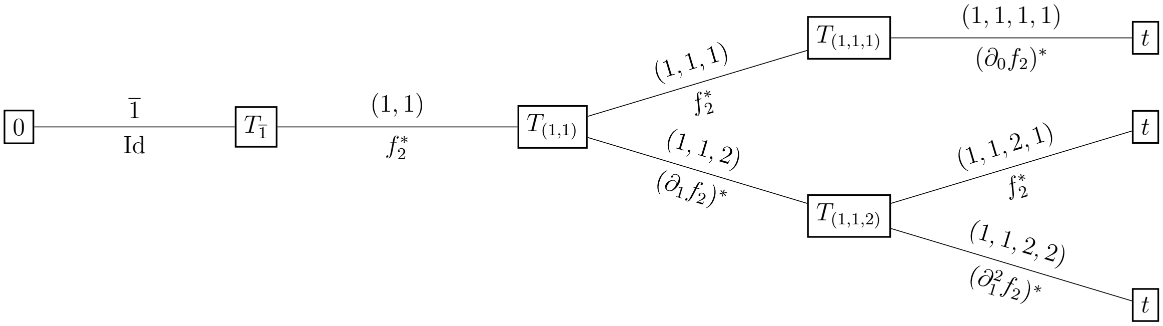
In this case, the set of codes is defined as

$$\mathcal{C} := \left\{ \mathrm{Id}, \ \left(\partial_0^k \partial_1^l f_2\right)^*, \quad k, l \geq 0 \right\},$$

and the mechanism $\mathcal{M}$ is given by

$$\mathcal{M}(\mathrm{Id}) = \{f_2^*\} \quad \text{and} \quad \mathcal{M}(g^*) = \big\{(\partial_0 g)^*, (f_2^*, (\partial_1 g)^*)\big\}.$$

A sample of this random tree is presented below.



In this case, the system of equations satisfied by $S_c(t)$ can be written as

$$\begin{cases} S_{\mathrm{Id}}(t) = K + \displaystyle\int_0^t S_{f_i^*}(s)ds \\ \\ S_{f_i^*}(t) = K + \displaystyle\int_0^t (S_{f_i^*}(s))^2 ds + \int_0^t S_{f_i^*}(s)ds, \end{cases}$$

which can be solved as

$$S_{\mathrm{Id}}(t) = K - \log(1 + K(1 - e^t)), \qquad S_{f_i^*}(t) = \frac{Ke^t}{1 + K(1 - e^t)}, \qquad i = 1, \ldots, d.$$

Therefore, the finiteness of $S_c(t)$ holds until time

$$T < \log\left(1 + \frac{1}{K}\right). \tag{4.7}$$

Under the conditions $\rho(T) \geq 2$ and $K \leq \overline{F}(T)$, since $q_{\min} := \min_{c \in \mathcal{C}} q_c(I_c) = 1/2$, we have

$$\prod_{\bar{k} \in \mathcal{K}^\circ} \frac{1}{q_{\min}\rho(T_{\bar{k}} - T_{\bar{k}-})} \prod_{\bar{k} \in \mathcal{K}^\partial} \frac{K}{\overline{F}(t - T_{\bar{k}-})} \leq 1,$$

hence

$$\mathbb{E}\big[\big|\mathcal{H}(\mathcal{T}_{t,c})\big|\big] \leq \mathbb{E}_c\left[\prod_{\bar{k} \in \mathcal{K}^\circ} \frac{1}{q_{\min}\rho(T_{\bar{k}} - T_{\bar{k}-})} \prod_{\bar{k} \in \mathcal{K}^\partial} \frac{K}{\overline{F}(t - T_{\bar{k}-})}\right] \leq 1, \quad t \in [0, T].$$

In this setting the integrability condition (4.7) is stronger than (4.6) when K is sufficiently large, and weaker otherwise. On the other hand, in comparison with the autonomous system (2.7), time complexity of the algorithm is divided by at least two due the use of a single coding tree with less branches.

5 Examples

Exponential series

We first consider the equation

$$\begin{cases} y'(t) = y(t) \\ y(0) = y_0 \end{cases}$$

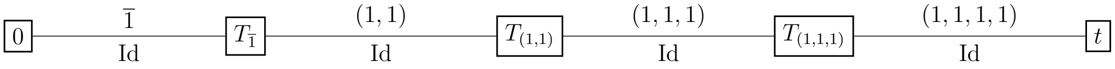
rewritten in integral form as

$$y(t) = y_0 + \int_0^t y(s)ds, \quad t \in \mathbb{R}_+,$$

whose solution admits the power series expansion

$$y(t) = y_0 e^t = y_0 \sum_{n=0}^{\infty} \frac{t^n}{n!}, \qquad t \in \mathbb{R}_+.$$

Here we have $\mathcal{C} = \{\text{Id}\}$, and the mechanism $\mathcal{M}$ satisfies $\mathcal{M}(\text{Id}) = \{\text{Id}\}$. The particle of generation n bears the label $\overline{k} = (1, \ldots, 1) \in \mathbb{N}^n$, and its parent is the particle $\overline{k}- = (1, \ldots, 1) \in \mathbb{N}^{n-1}$. When the random times $(\tau^k)_{k\geq 1}$ are independent and exponentially distributed, i.e. $\rho(s) = e^{-s}$ with $\overline{F}(t) = e^{-t}$, the total number of branches in the random tree $\mathcal{T}_t$ is given by $N_t + 1$ where $(N_t)_{t\geq 0}$ is a standard Poisson process with unit intensity.



In this case, the multiplicative functional

$$\mathcal{H}(\mathcal{T}_t) := \prod_{k\in\mathcal{K}^\circ} \frac{1}{\rho(T_{\overline{k}} - T_{\overline{k}-})} \prod_{k\in\mathcal{K}^\partial} \frac{y_0}{\overline{F}(t - T_{\overline{k}-})}$$

simplifies to the deterministic expression

$$\mathcal{H}(\mathcal{T}_t) = \left(\prod_{k=1}^{N_t} \frac{1}{e^{-\tau^k}}\right) \frac{y_0}{e^{-(t-\sum_{k=1}^{N_t}\tau^k)}} = y_0 e^t, \qquad t \in \mathbb{R}_+,$$

in which we take $\sum_{n=1}^0 1 := 0$ and $\prod_{n=1}^0 1 := 1$.

One-dimensional autonomous ODE

Consider the autonomous ODE

$$y(t) = y_0 + \int_0^t f(y(s))ds, \qquad t \in \mathbb{R}_+, \tag{5.1}$$

where $f \in C^\infty(\mathbb{R};\mathbb{R})$ is bounded together with its derivatives $f^{(k)}$ of order $k \geq 1$, with

$$\left|f^{(k)}(y_0)\right| \leq K, \qquad k \geq 0,$$

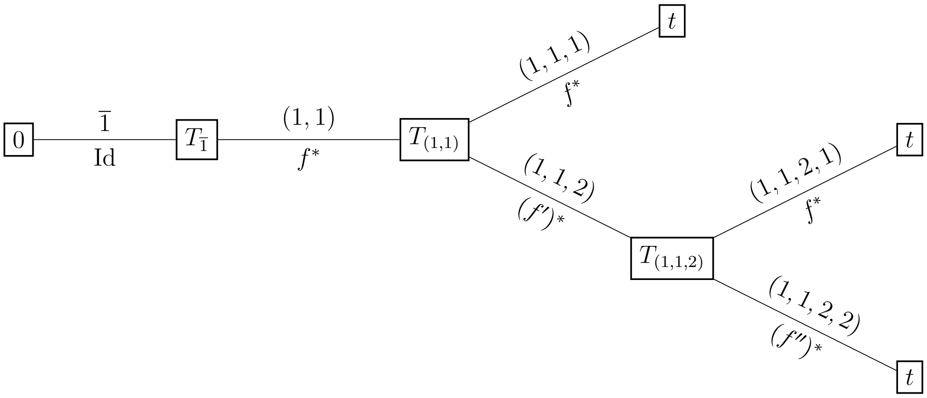
for some $K > 0$. For any $k \geq 0$ we have the integral equation

$$f^{(k)}(y(t)) = f^{(k)}(y_0) + \int_0^t f(y(s)) f^{(k+1)}(y(s)) ds, \qquad t \in \mathbb{R}_+,$$

with the set of codes $\mathcal{C} := \left\{\mathrm{Id},\ \left(f^{(k)}\right)^*,\ k \geq 0\right\}$, and the mechanism $\mathcal{M}$ is defined by

$$\mathcal{M}(\mathrm{Id}) := \{f^*\}, \quad \mathcal{M}(f^*) := \left\{(f^*, (f')^*)\right\}, \quad \mathcal{M}\left(\left(f^{(k)}\right)^*\right) := \left\{\left(f^*, \left(f^{(k+1)}\right)^*\right)\right\}, \quad k \geq 1.$$

Below is a representation of a sample of the random tree $\mathcal{T}_{t,\mathrm{Id}}$.



With $y_c(t) := \mathbb{E}\left[\mathcal{H}(\mathcal{T}_{t,c})\right]$, $c \in \mathcal{C}$, the system (4.2) reads

$$\begin{cases} y_{\mathrm{Id}}(t) = y_0 + \displaystyle\int_0^t y_{f^*}(s) ds \\ y_{(f^{(k)})^*}(t) = f^{(k)}(y_0) + \displaystyle\int_0^t y_{f^*}(s) y_{(f^{(k+1)})^*}(s) ds, \qquad k \geq 0. \end{cases}$$

6 Mapping of coding trees to Butcher trees

In this section we describe the connection between coding trees, Butcher series and Butcher trees, by showing how any Butcher tree can be recovered by performing a depth first search on the corresponding coding trees. The solution $y(t)$ of the d-dimensional ODE system (1.1) is written as the Butcher series

$$y(t) = y_0 + t f(y_0) + \frac{t^2}{2} f(y_0) f'(y_0)$$

$$\begin{aligned}
&+\frac{t^3}{3!}\big(f''(y_0)f^2(y_0)+f'^2(y_0)f(y_0)\big)\\
&+\frac{t^4}{4!}\big(f'''(y_0)f^3(y_0)+f''(y_0)f'(y_0)f^2(y_0)+f'(y_0)f'(y_0)f^2(y_0)+f'^3(y_0)f(y_0)\big)+\cdots\\
=\quad & y_0+\sum_{\mathcal{B}}\frac{t^{|\mathcal{B}|}}{\nu(\mathcal{B})}c(\mathcal{B}) \qquad (6.1)
\end{aligned}$$

where the above summation over Butcher trees $\mathcal{B}$ is formal and may not converge. Here, the order $|\mathcal{B}|$ denotes the number of vertices of the tree $\mathcal{B}$, $c(\mathcal{B})$ is a term also depending on the derivatives of f at y_0 as in (1.3)-(1.5), and $\nu(\mathcal{B})$ is a coefficient which is defined recursively, see Relation (4) in [But10].

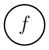
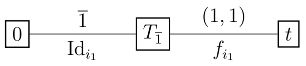
Proposition 6.1 *Every Butcher tree $\mathcal{B}$ can be mapped to a unique tree* $\text{bin}(\mathcal{B})$*, so that the Butcher series* (6.1) *can be rewritten as*

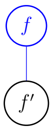
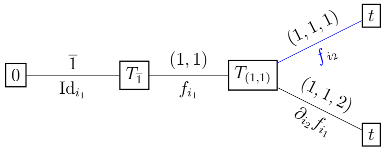
$$y_i(t)=y_i(0)+\sum_{\mathcal{B}}\mathbb{E}\left[\mathcal{H}(\mathcal{T}_{t,\text{Id}_i})\mathbf{1}_{\left\{\mathcal{T}_{t,\text{Id}_i}\simeq\,\text{bin}(\mathcal{B})\right\}}\right],\qquad i=1,\ldots,d.$$

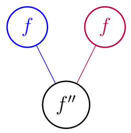
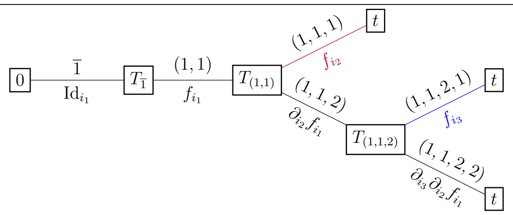
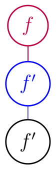
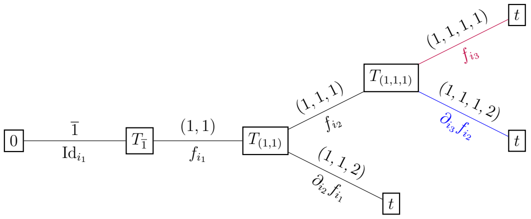
where the notation $\mathcal{T}_{t,\text{Id}_i}\simeq$ $\text{bin}(\mathcal{B})$ *means that the coding tree $\mathcal{T}_{t,\text{Id}_i}$ has the tree structure* $\text{bin}(\mathcal{B})$.

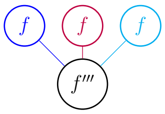
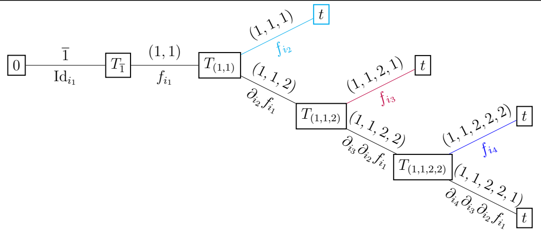
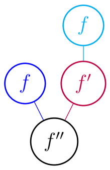
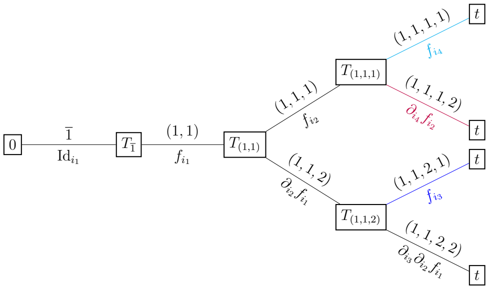
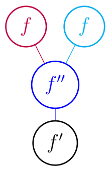
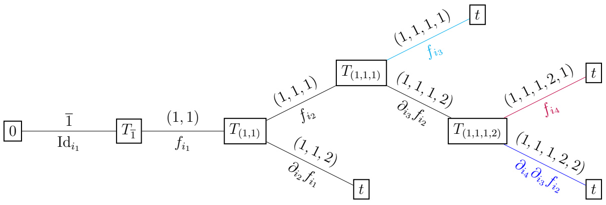
Proof. (i) The one-to-one mapping $\mathcal{B}\leftrightarrow\text{bin}(\mathcal{B})$ is constructed by performing a depth first search on coding trees, as illustrated in the following examples which show how any Butcher tree $\mathcal{B}$ of order $k\geq 1$ can be mapped to a unique tree $\text{bin}(\mathcal{B})$ generating k^d coding trees $(\mathcal{T}_{i_1,\ldots,i_k})_{1\leq i_1,\ldots,i_k\leq d}$. Namely, every Butcher tree can be recovered by performing a depth first search on the corresponding coding tree by matching leaves on Butcher trees to branches of the same color in coding trees.

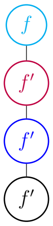
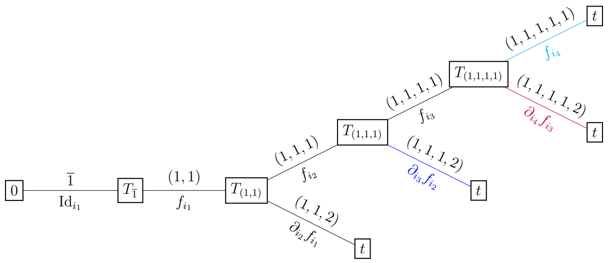
If a node in the initial coding tree branches into two new coding trees $\mathcal{T}_1$ (above) and $\mathcal{T}_2$ (below), the Butcher tree for the initial coding tree is obtained by sticking the Butcher tree obtained from $\mathcal{T}_1$ to the root of the Butcher tree obtained from $\mathcal{T}_2$. If a tree is a leaf with code of the form $c=\big(\partial_{j_1}\cdots\partial_{j_l}f_i\big)^*$ with $i\in\{1,\ldots,d\}$, then the corresponding Butcher tree has a single node containing $f^{(l)}$.

Order k	$c(\mathcal{B})$	Butcher tree $\mathcal{B}$	Coding trees $(\mathcal{T}_{i_1})_{1\leq i_1\leq d}$
1	$f(y_0)$	 	 0 $\bar{1}$ Id_{i_1} $T_{\bar{1}}$ $(1,1)$ f_{i_1} t

Order k	$c(\mathcal{B})$	Butcher tree $\mathcal{B}$	Coding trees $(\mathcal{T}_{i_1,i_2})_{1\leq i_1,i_2\leq d}$
2	$ff'(y_0)$	 	 0 $\bar{1}$ Id_{i_1} $T_{\bar{1}}$ $(1,1)$ f_{i_1} $T_{(1,1)}$ $(1,1,1)$ f_{i_2} t $(1,1,2)$ $\partial_{i_2}f_{i_1}$ t

Order k	$c(\mathcal{B})$	Butcher tree $\mathcal{B}$	Coding trees $(\mathcal{T}_{i_1,i_2,i_3})_{1\leq i_1,i_2,i_3\leq d}$
3	$f''f^2(y_0)$		 0 $\bar{1}$ Id_{i_1} $T_{\bar{1}}$ $(1,1)$ f_{i_1} $T_{(1,1)}$ $(1,1,1)$ f_{i_2} t $(1,1,2)$ $\partial_{i_2}f_{i_1}$ $T_{(1,1,2)}$ $(1,1,2,1)$ f_{i_3} t $(1,1,2,2)$ $\partial_{i_3}\partial_{i_2}f_{i_1}$ t
3	$f'^2f(y_0)$		 0 $\bar{1}$ Id_{i_1} $T_{\bar{1}}$ $(1,1)$ f_{i_1} $T_{(1,1)}$ $(1,1,1)$ f_{i_2} $T_{(1,1,1)}$ $(1,1,1,1)$ f_{i_3} t $(1,1,1,2)$ $\partial_{i_3}f_{i_2}$ t $(1,1,2)$ $\partial_{i_2}f_{i_1}$ t

Order k	$c(\mathcal{B})$	Butcher tree $\mathcal{B}$	Coding trees $(\mathcal{T}_{i_1,i_2,i_3,i_4})_{1\leq i_1,i_2,i_3,i_4\leq d}$
4	$f'''f^3(y_0)$		 0 $\bar{1}$ Id_{i_1} $T_{\bar{1}}$ $(1,1)$ f_{i_1} $T_{(1,1)}$ $(1,1,1)$ f_{i_2} t $(1,1,2)$ $\partial_{i_2}f_{i_1}$ $T_{(1,1,2)}$ $(1,1,2,1)$ f_{i_3} t $(1,1,2,2)$ $\partial_{i_3}\partial_{i_2}f_{i_1}$ $T_{(1,1,2,2)}$ $(1,1,2,2,2)$ f_{i_4} t $(1,1,2,2,1)$ $\partial_{i_4}\partial_{i_3}\partial_{i_2}f_{i_1}$ t
4	$f''f'f^2(y_0)$		 0 $\bar{1}$ Id_{i_1} $T_{\bar{1}}$ $(1,1)$ f_{i_1} $T_{(1,1)}$ $(1,1,1)$ f_{i_2} $T_{(1,1,1)}$ $(1,1,1,1)$ f_{i_4} t $(1,1,1,2)$ $\partial_{i_4}f_{i_2}$ t $(1,1,2)$ $\partial_{i_2}f_{i_1}$ $T_{(1,1,2)}$ $(1,1,2,1)$ f_{i_3} t $(1,1,2,2)$ $\partial_{i_3}\partial_{i_2}f_{i_1}$ t
4	$f'f''f^2(y_0)$		 0 $\bar{1}$ Id_{i_1} $T_{\bar{1}}$ $(1,1)$ f_{i_1} $T_{(1,1)}$ $(1,1,1)$ f_{i_2} $T_{(1,1,1)}$ $(1,1,1,1)$ f_{i_3} t $(1,1,1,2)$ $\partial_{i_3}f_{i_2}$ $T_{(1,1,1,2)}$ $(1,1,1,2,1)$ f_{i_4} t $(1,1,1,2,2)$ $\partial_{i_4}\partial_{i_3}f_{i_2}$ t $(1,1,2)$ $\partial_{i_2}f_{i_1}$ t

4	$f'^3 f(y_0)$		 0 $\overline{1}$ Id_{i_1} $T_{\overline{1}}$ $(1,1)$ f_{i_1} $T_{(1,1)}$ $(1,1,1)$ f_{i_2} $(1,1,2)$ $\partial_{i_2} f_{i_1}$ t $T_{(1,1,1)}$ $(1,1,1,1)$ f_{i_3} $(1,1,1,2)$ $\partial_{i_3} f_{i_2}$ t $T_{(1,1,1,1)}$ $(1,1,1,1,1)$ f_{i_4} t $(1,1,1,1,2)$ $\partial_{i_4} f_{i_3}$ t

(ii) When $\mathcal{B}$ is a Butcher tree of order $k = |\mathcal{B}| \geq 1$, based on the construction of $\nu(\mathcal{B}) = \sigma(\mathcal{B})\gamma(\mathcal{B})$ on page 155 of [But10], the i_1-*th* component of the term $t^{|\mathcal{B}|}c(\mathcal{B})/\nu(\mathcal{B})$ in the expansion of $y_{i_1}(t)$ is a sum of f and its partial derivatives at y_0 as in (1.3)-(1.5). This sum can be interpreted as a sum of $\mathcal{H}(\mathcal{T}_{i_1,\ldots,i_k})$ over $1 \leq i_2, \ldots, i_k \leq d$, weighted by the probability coefficients

$$\prod_{\overline{k}\in\mathcal{K}^\circ} \left(q_{c_{\overline{k}}}(I_{c_{\overline{k}}})\rho(T_{\overline{k}} - T_{\overline{k}-})\right) \prod_{\overline{k}\in\mathcal{K}^\partial} \left(\overline{F}(t - T_{\overline{k}-})\right)$$

and integrated over the branching times $T_{\overline{k}} \in [0, t]$, which yields the identity

$$\frac{t^{|\mathcal{B}|}}{\nu(\mathcal{B})}c(\mathcal{B}) = \left(\mathbb{E}\left[\mathcal{H}(\mathcal{T}_{t,\mathrm{Id}_{i_1}})\mathbf{1}_{\left\{\mathcal{T}_{t,\mathrm{Id}_{i_1}} \simeq \mathrm{bin}(\mathcal{B})\right\}}\right]\right)_{i_1=1,\ldots,d}.$$

□

7 Numerical application

In this section we consider various ODE examples with different choices of probability density functions $\rho(t)$ satisfying (4.4), or under (4.6). The errors observed in Figures 3, 4, 5, only occur at a time threshold after which the estimator $\mathcal{H}(\mathcal{T}_{t,c})$ is no longer integrable and the estimates are unreliable. The following graphs are plotted with one million Monte Carlo samples.

i) Taking $f(y) := y^2$, we start with the quadratic ODE

$$y'(t) = y^2(t), \qquad y(0) = y_0 = 1, \tag{7.1}$$

with solution

$$y(t) = \frac{y_0}{1 - y_0 t}, \qquad t \in [0, 1/y_0).$$

In the framework of (5.1) we have

$$\mathcal{C} = \left\{\mathrm{Id},\ \left(f^{(k)}\right)^*,\ k \geq 0\right\} = \left\{\mathrm{Id},\ (x \mapsto 0)^*,\ (x \mapsto 2)^*,\ (x \mapsto 2x)^*,\ (x \mapsto x^2)^*\right\},$$

hence we have $c(y)(0) \leq K := \max(1, 2y_0, y_0^2) = 2$ for all $c \in \mathcal{C}$, and in agreement with (4.6) the representation formula (4.1) of Theorem 4.2 holds for $t \in [0, 1/2)$, see Figure 1. In this example and in the next two examples we take ρ to be the exponential probability density function $\rho(t) = e^{-t}$, $t \geq 0$.

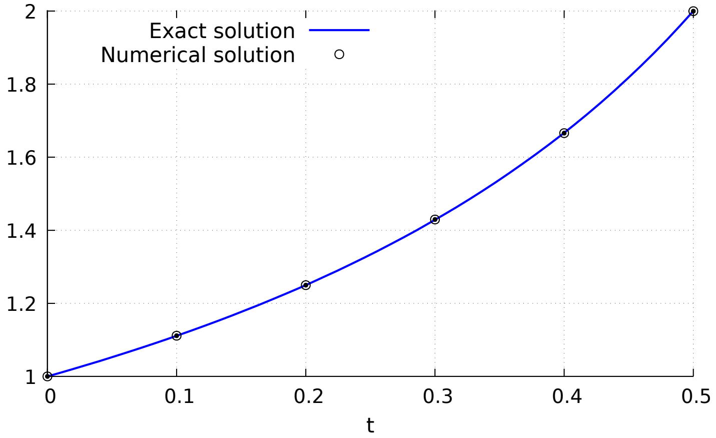


Figure 1: Numerical solution of (7.1) with $y_0 = 1$.

ii) Next, we take $f(y) := \cos(y)$ and consider the equation

$$y'(t) = \cos(y(t)), \quad y(0) = y_0, \tag{7.2}$$

with solution

$$y(t) = 2\tan^{-1}\left(\tanh\left(\frac{t + 2\tanh^{-1}(\tan(y_0/2))}{2}\right)\right), \qquad t \in \mathbb{R}_+.$$

in the framework of (5.1). When $y_0 = 1$ we have $K = \sup_{k\in\mathbb{N}} f^{(k)}(1) = 1$, and in agreement with (4.6) the representation formula (4.1) of Theorem 4.2 holds for $t \in [0, 1)$, see Figure 2.

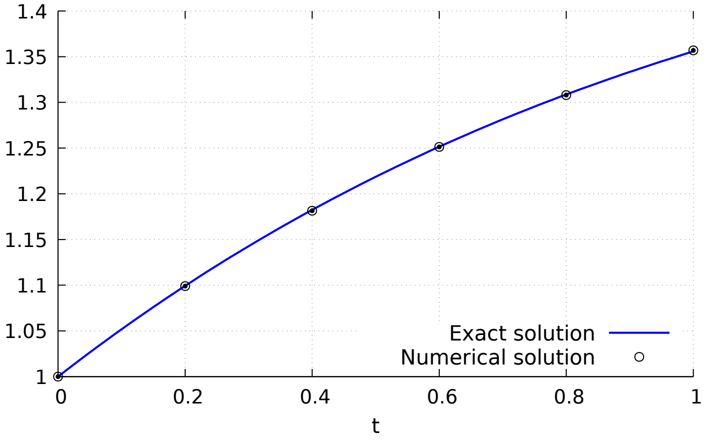


Figure 2: Numerical solution of (7.2).

iii) Taking $f(t,y) := (y+t)/(y-t)$ we find Equation (201a) in [But16], i.e.

$$y'(t) = \frac{y(t)+t}{y(t)-t}, \qquad y(0) = 1, \tag{7.3}$$

with solution

$$y(t) = t + \sqrt{1+2t^2}.$$

In this case, the time interval of validity may not be determined explicitly because $\sup_{k,l\geq 0} |\partial_0^k \partial_1^l f(0,1/2)| = \infty$ and the finiteness of this supremum is only a sufficient condition for (4.5) to hold in Proposition 4.3. Figure 3 shows the convergence of the Monte Carlo algorithm until $t = 0.25$.

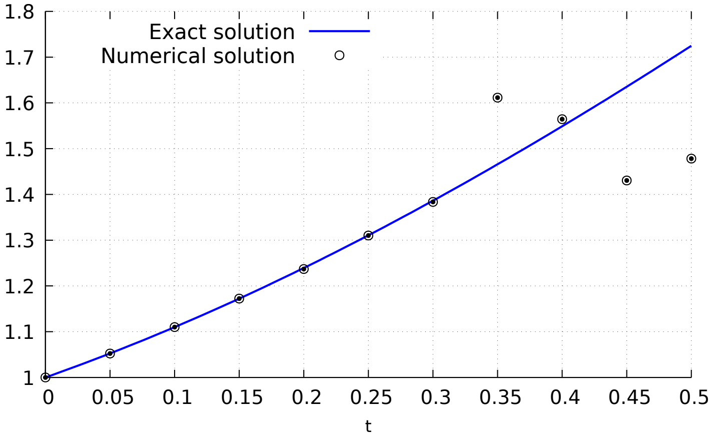


Figure 3: Numerical solution of (7.3).

iv) Taking $f(t,y) := (y-t)/(y+t)$ yields Equation (316e) in [But16], i.e.

$$y'(t) = \frac{y(t)-t}{y(t)+t}, \qquad y(0) = 1, \tag{7.4}$$

whose solution is given in parametric form as $(t(u), y(u)) = (u \sin\log(u), u\cos\log(u))$. As in Example iii) above, the time interval of validity may not be determined explicitly, see Figure 4. In this example and in the next one, we take ρ to be the gamma probability density function $\rho(t) = t^{-1/2}e^{-t}/\Gamma(1/2)$, $t > 0$.

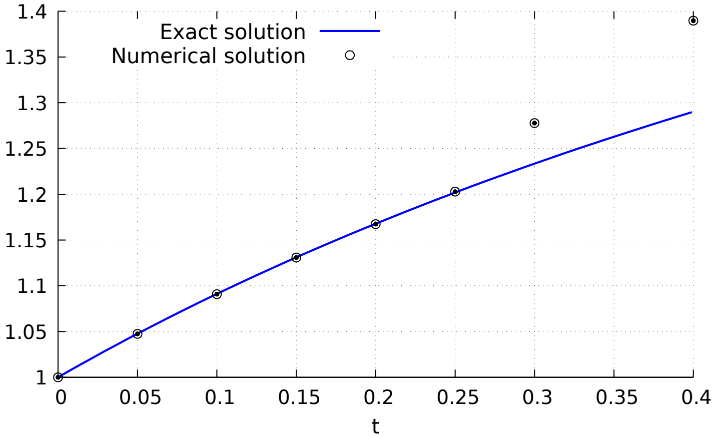


Figure 4: Numerical solution of (7.4).

v) Taking $f(t,y) := yt + y^2$ we find Equation (223a) in [But16], i.e.

$$y'(t) = ty(t) + y^2(t), \qquad y(0) = 1/2, \tag{7.5}$$

with solution

$$y(t) = \frac{e^{t^2/2}}{2 - \int_0^t e^{s^2/2} ds}.$$

In this case we have $K = \sup_{k,l \geq 0} |\partial_0^k \partial_1^l f(0, 1/2)| \leq 2$, and in agreement with (4.6) the representation formula (4.1) of Theorem 4.2 is valid on the time interval $[0, 0.5) \subset [0, 1/K]$, see Figure 5.

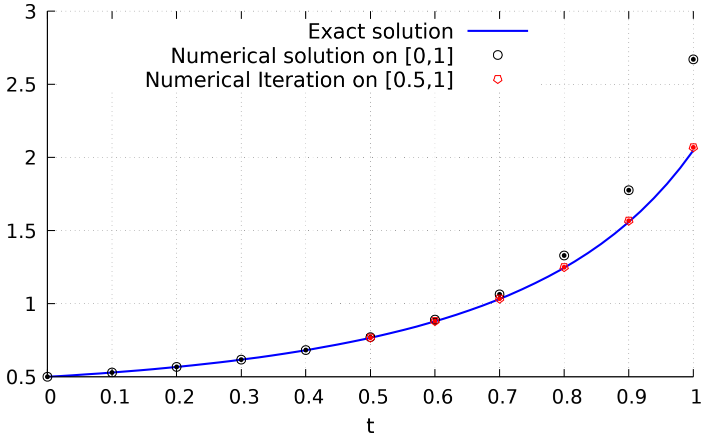


Figure 5: Numerical solution of (7.5).

In addition, after running the algorithm on the time interval $[0, 0.5]$ we may reuse the numerical evaluation at time $t = 0.5$ as a new initial condition and iterate the algorithm on the time interval $[0.5, 1]$ with more precise estimates, as shown in red in Figure 5. We refer to this procedure as “patching”.

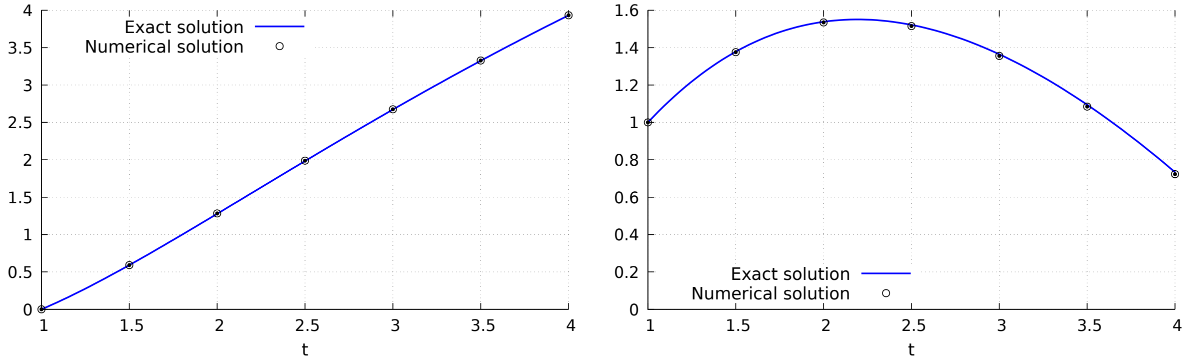
vi) Consider the ODE system (316f) page 177 in [But16], i.e.

$$\begin{cases} y_1'(t) = \dfrac{y_1(t)+y_2(t)}{\sqrt{y_1^2(t)+y_2^2(t)}}, & y_1(1) = 0, \\ y_2'(t) = \dfrac{y_2(t)-y_1(t)}{\sqrt{y_1^2(t)+y_2^2(t)}}, & y_2(1) = 1, \end{cases} \tag{7.6}$$

with solution

$$y_1(t) = t \sin \log t, \qquad y_2(t) = t \cos \log t, \qquad t \geq 1.$$

The graphs of Figure 6 are obtained using the Python code provided in appendix, where we patch the algorithm 6 times over the time interval $[1, 4]$ and take ρ to be the exponential probability density function $\rho(t) = e^{-t}$, $t > 0$.



(a) Graph of $y_1(t)$.

(b) Graph of $y_2(t)$.

Figure 6: Numerical solution of (7.6).

A Computer codes

The following Maple and Mathematica codes implement the algorithm of Theorem 4.2 for one-dimensional non-autonomous ODEs using the exponential distribution $\rho(t) = \mathrm{e}^{-t}$, $t \geq 0$.

```
codetofunction := proc(f, c, t0, y0) if nops(c) = 0 then return y0; end if; if c = [0, 0] then return
     f(t0, y0); else return eval(eval(diff(f(t, y), t $ c[1], y $ c[2]), t = t0), y = y0); end if; end
     proc;
with(stats);
mcsample := proc(f, t, t0, y0, c, h) local A, tau; tau := random[exponential[1]](1); if t - t0 < tau then
     return h*codetofunction(f, c, t0, y0)/exp(-t + t0); else if nops(c) = 0 then return mcsample(f, t -
     tau, t0, y0, [0, 0], h/exp(-tau)); else if random[uniform](1) < 0.5 then return mcsample(f, t - tau,
     t0, y0, [c[1] + 1, c[2]], 2*h/exp(-tau)); else A := mcsample(f, t - tau, t0, y0, [0, 0], 1); return
     mcsample(f, t - tau, t0, y0, [c[1], c[2] + 1], 2*A*h/exp(-tau)); end if; end if; end if; end proc;
solution := proc(f, t, t0, y0, n) local i, temp; temp := 0; for i to n do temp := temp + mcsample(f, t,
     t0, y0, [], 1); end do; return evalf(temp/n); end proc;
f := (t, y) -> y^2;solution(f, 0.5, 0, 1, 10000);
```

Maple code.

```
codetofunction[f_, c__, t0_, y0_] := (If [c == {}, Return [y0],
   Return[D[D[f[t, y], {t, c[[1]]}], {y, c[[2]]}] /. {t -> t0} /. {y -> y0}]])
MCsample[f_, t_, t0_, y0_, c__, h_] := (Module[{A, tau}, tau = RandomVariate[ExponentialDistribution[1]];
   If[tau > t - t0, Return [h*codetofunction[f, c, t0, y0]/E^(-(t - t0))],
    If[c == {}, Return[MCsample[f, t - tau, t0, y0, {0, 0}, h/E^(-tau)]],
     If[RandomVariate[UniformDistribution[1]][[1]] <= 0.5,
      Return[MCsample[f, t - tau, t0, y0, {c[[1]] + 1, c[[2]]}, 2*h/E^(-tau)]],
      A = MCsample[f, t - tau, t0, y0, {0, 0}, 1];
      Return[MCsample[f, t - tau, t0, y0, {c[[1]], c[[2]] + 1}, 2*A*h/E^(-tau)]]]]]])
Solution[f_, t_, t0_, y0_, n_] := (temp = 0; For[i = 1, i <= n, i++, temp += MCsample[f, t, t0, y0, {},
     1]]; Return[temp/n])
f[t_, y_] := y^2;Solution[f, 0.5, 0, 1, 100000]
```

Mathematica code.

The following Python code implements the algorithm of Theorem 4.2 for systems of ODEs.

```
import time
import math
import torch
from scipy import special
from torch.distributions.exponential import Exponential
import matplotlib.pyplot as plt
import numpy as np

torch.manual_seed(0) # set seed for reproducibility

class ODEBranch(torch.nn.Module):
    def __init__(
        self,
        fun,
        t_lo=0.0,
        t_hi=1.0,
        y0=1.0,
        nb_path_per_state=1000000,
        nb_states=6,
        outlier_percentile=1,
        outlier_multiplier=1000,
        patch=1,
        epochs=3000,
        device="cpu",
        verbose=False,
        **kwargs,
    ):
        super(ODEBranch, self).__init__()
        self.fun = fun

        self.loss = torch.nn.MSELoss()
        self.nb_path_per_state = nb_path_per_state
        self.nb_states = nb_states
        self.outlier_percentile = outlier_percentile
        self.outlier_multiplier = outlier_multiplier
        self.patch = patch
        self.t_lo = t_lo
        self.t_hi = t_hi
        self.y0 = y0
        self.dim = len(y0)
        self.epochs = epochs
        self.device = device
        self.verbose = verbose

    def forward(self, code=None):
        start = time.time()
        code = [-1] * self.dim if code is None else code # start from identity code if not specified
        t = torch.linspace(self.t_lo, self.t_hi, steps=self.nb_states, device=self.device)
        t = t.repeat(self.nb_path_per_state).reshape(self.nb_path_per_state, -1).T
        nb_states_per_patch = math.ceil( ( self.nb_states - 1 ) / self.patch)
        cur_start_idx, cur_end_idx = 0, nb_states_per_patch
        mc_mean, mc_var = [], []
        y0, t0 = torch.tensor(self.y0, device=self.device), torch.tensor(self.t_lo, device=self.device)
        while cur_start_idx < cur_end_idx:
            self.code_to_fun_dict = {}
            t_this_patch = t[cur_start_idx:cur_end_idx]
            H_tensor = torch.ones_like(t_this_patch)
            mask_tensor = torch.ones_like(t_this_patch)
            mc_mean_this_patch = []
            mc_var_this_patch = []
            for i in range(self.dim):
                y = self.gen_sample_batch(
                    t_this_patch,
                    t0,
                    y0,
                    np.array(code),
                    H_tensor,
                    mask_tensor,
                    coordinate=i
                )
                # widen (outlier_percentile, 1 - outlier_percentile) by outlier_multiplier times
                # everything outside this range is considered outlier
                lo = y.nanquantile(self.outlier_percentile/100, dim=1, keepdim=True)
                hi = y.nanquantile(1 - self.outlier_percentile/100, dim=1, keepdim=True)
                lo, hi = lo - self.outlier_multiplier * (hi - lo), hi + self.outlier_multiplier * (hi - lo)
                mask = torch.logical_and(lo <= y, y <= hi)
                mc_mean_this_patch.append((y * mask).sum(dim=1) / mask.sum(dim=1))
                y = y - mc_mean_this_patch[-1].unsqueeze(dim=-1)
                mc_var_this_patch.append(torch.square(y * mask).sum(dim=1) / mask.sum(dim=1))

            # update y0, t0, idx
            mc_mean.append(torch.stack(mc_mean_this_patch))
            mc_var.append(torch.stack(mc_var_this_patch))
            y0, t0 = mc_mean[-1][:, -1], t_this_patch[-1][-1]
            cur_start_idx, cur_end_idx = cur_end_idx, min(cur_end_idx + nb_states_per_patch,
                 self.nb_states)

        if self.verbose:
            print(f"Time taken for the simulations: {time.time() - start:.2f} seconds.")
        return t[:, 0], torch.cat(mc_mean, dim=-1), torch.cat(mc_var, dim=-1)

    @staticmethod
    def nth_derivatives(order, y, x):
        """
        calculate the derivatives of y wrt x with order `order`
        """
        for cur_dim, cur_order in enumerate(order):
            for _ in range(int(cur_order)):
                try:
                    grads = torch.autograd.grad(y.sum(), x, create_graph=True)[0]
                except RuntimeError as e:
                    # when very high order derivatives are taken for polynomial function
                    # it has 0 gradient but torch has difficulty knowing that
                    # hence we handle such error separately
                    # logging.debug(e)
                    return torch.zeros_like(y)

                # update y
                y = grads[cur_dim]
        return y

    def code_to_function(self, code, t, y0, coordinate):
        code = tuple(code)
        if (code, coordinate) not in self.code_to_fun_dict.keys():
            # code (-1, -1, ..., -1) -> identity mapping
            if code == (-1,) * self.dim:
                self.code_to_fun_dict[(code, coordinate)] = y0[coordinate]
            else:
                y = y0.clone().requires_grad_(True)
                self.code_to_fun_dict[(code, coordinate)] = (
                    self.nth_derivatives(code, self.fun(y, coordinate), y).detach()
                )
        return self.code_to_fun_dict[(code, coordinate)]

    def gen_sample_batch(self, t, t0, y0, code, H, mask, coordinate):
        nb_states, _ = t.shape
        tau = Exponential(
            torch.ones(nb_states, self.nb_path_per_state, device=self.device)
        ).sample()
        ans = torch.zeros_like(t)

        ############################### for t + tau >= T
        mask_now = mask.bool() * (t0 + tau >= t)
        if mask_now.any():
            ans[mask_now] = (
                    H[mask_now]
                    * self.code_to_function(code, t0, y0, coordinate)
                    / torch.exp(-(t - t0)[mask_now])
            )

        ############################### for t + tau < T
        mask_now = mask.bool() * (t0 + tau < t)
        if (code == [-1] * self.dim).all():
            if mask_now.any():
                # code (-1, -1,..., -1) -> (0, 0,..., 0)
                tmp = self.gen_sample_batch(
                    t - tau, t0, y0, code + 1, H / torch.exp(-tau), mask_now, coordinate,
                )
                ans = ans.where(~mask_now, tmp)

        else:
            unif = torch.rand(nb_states, self.nb_path_per_state, device=self.device)
            idx = (unif * self.dim).long()
            for i in range(self.dim):
                mask_tmp = mask_now * (idx == i)
                if mask_tmp.any():
                    A = self.gen_sample_batch(
                        t - tau,
                        t0,
                        y0,
                        np.array([0] * self.dim),
                        torch.ones_like(t),
                        mask_tmp,
                        i,
                    )
                    code[i] += 1
                    tmp = self.gen_sample_batch(
                        t - tau,
                        t0,
                        y0,
                        code,
                        self.dim * A * H / torch.exp(-tau),
                        mask_tmp,
                        coordinate,
                    )
                    code[i] -= 1
                    ans = ans.where(~mask_tmp, tmp)
        return ans
```

```
if __name__ == "__main__":
    # problem configuration
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    problem = [
        "quadratic",
        "cosine",
        "example_3",
        "example_5",
        "example_6"
    ][4]
    dim = 1
    if problem == "quadratic":
        exact_fun = (lambda t, y, coordinate: y[coordinate] / (1 - y[coordinate] * t))
        f_fun = (lambda y, coordinate: y[coordinate] ** 2)
        t_lo, t_hi = 0, 0.5
        y0 = [1.0] * dim
        nb_states = 6
    elif problem == "cosine":
        def exact_fun(t, y, coordinate):
            return 2 * torch.atan(torch.tanh((t + 2 * math.atanh(math.tan(y[coordinate] / 2))) / 2))
        f_fun = (lambda y, coordinate: torch.cos(y[coordinate]))
        t_lo, t_hi = 0, 1.0
        y0 = [1.0] * dim
        nb_states = 6
    elif problem == "example_3":
        def exact_fun(t, y, coordinate):
            if coordinate == 0:
                return y[coordinate] + t
            else:
                return t + torch.sqrt(y[coordinate] + 2 * t ** 2)

        def f_fun(y, coordinate):
            if coordinate == 0:
                return torch.ones_like(y[0])
            else:
                return (y[coordinate] + y[0]) / (y[coordinate] - y[0])
        t_lo, t_hi = 0, 0.5
        y0 = [t_lo] + [1.0] * dim
        nb_states = 11
    elif problem == "example_5":
        def exact_fun(t, y, coordinate):
            if coordinate == 0:
                return y[coordinate] + t
            else:
                tensor_erfi = (lambda x: special.erfi(x.cpu()).to(device))
                return torch.exp(t**2/2) / (1/y[coordinate] - (math.pi / 2) ** 0.5 * tensor_erfi(t / 2 **
                     0.5))

        def f_fun(y, coordinate):
            if coordinate == 0:
                return torch.ones_like(y[0])
            else:
                return y[0] * y[coordinate] + y[coordinate]**2
        t_lo, t_hi = 0, 1.0
        y0 = [t_lo] + [.5] * dim
        nb_states = 11
    elif problem == "example_6":
        def exact_fun(t, y, coordinate):
            if coordinate == 0:
                return t*torch.sin(torch.log(t))
            else:
                return t*torch.cos(torch.log(t))

        def f_fun(y, coordinate):
            if coordinate == 0:
```

```
            return ( y[1] + y[0] ) / torch.sqrt(y[0]**2+y[1]**2)
        else:
            return ( y[1] - y[0] ) / torch.sqrt(y[0]**2+y[1]**2)
    t_lo, t_hi = 1, 4
    y0 = [0.0] + [1.0]
    nb_states = 7

# initialize model and calculate mc samples
model = ODEBranch(
    f_fun,
    t_lo=t_lo,
    t_hi=t_hi,
    y0=y0,
    device=device,
    nb_states=nb_states,
    verbose=True,
    patch=6,
    outlier_percentile=0.1,
    outlier_multiplier=100,
)
t, mc_mean, mc_var = model()
t_fine = torch.linspace(t_lo, t_hi, 100, device=device) # finer grid for plotting exact solution
torch.set_printoptions(precision=5, sci_mode=True)

# plot exact vs numerical
for i in range(model.dim):
    print(f"For dimension {i + 1}:")
    print(f"The variance of MC is {mc_var[i]}.")
    print(f"The error squared is {(mc_mean[i] - exact_fun(t, y0, i)) ** 2}.")
    plt.plot(t.cpu(), mc_mean[i].cpu(), '+', label="Numerical solution")
    plt.plot(t_fine.cpu(), exact_fun(t_fine, y0, i).cpu(), label="Exact solution")
    plt.title(f"Dimension {i + 1}")
    plt.legend()
    plt.show()
```

Acknowledgement

We thank Jiang Yu Nguwi for producing the Python code dealing with systems of ODEs.

References

[BHZ19] Y. Bruned, M. Hairer, and L. Zambotti. Algebraic renormalisation of regularity structures. *Invent. Math.*, 215:1039–1156, 2019.

[But63] J.C. Butcher. Coefficients for the study of Runge-Kutta integration processes. *J. Austral. Math. Soc.*, 3:185–201, 1963.

[But10] J.C. Butcher. Trees and numerical methods for ordinary differential equations. *Numerical Algorithms*, 53:153–170, 2010.

[But16] J.C. Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, Ltd., Chichester, third edition, 2016.

[Cay57] A. Cayley. On the theory of the analytical forms called trees. *Philosophical Magazine*, 13(85):172–176, 1857.

[CK99] A. Connes and D. Kreimer. Lessons from quantum field theory: Hopf algebras and spacetime geometries. *Letters in Mathematical Physics*, 48:85–96, 1999.

[CLM08] S. Chakraborty and J.A. López-Mimbela. Nonexplosion of a class of semilinear equations via branching particle representations. *Advances in Appl. Probability*, 40:250–272, 2008.

[DB02] P. Deuflhard and F. Bornemann. *Scientific Computing with Ordinary Differential Equations*, volume 42 of *Texts in Applied Mathematics*. Springer-Verlag, New York, 2002.

[Fos21] L. Fossy. Algebraic structures on typed decorated rooted trees. *SIGMA*, 17:1–28, 2021. Contribution to the Special Issue on Algebraic Structures in Perturbative Quantum Field Theory in honor of Dirk Kreimer for his 60th birthday.

[Gub10] M. Gubinelli. Ramification of rough paths. *J. Differential Equations*, 248(4):693–721, 2010.

[HLOT$^+$19] P. Henry-Labordère, N. Oudjane, X. Tan, N. Touzi, and X. Warin. Branching diffusion representation of semilinear PDEs and Monte Carlo approximation. *Ann. Inst. H. Poincaré Probab. Statist.*, 55(1):184–210, 2019.

[HLW06] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration*, volume 31 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 2006. Structure-preserving algorithms for ordinary differential equations.

[INW69] N. Ikeda, M. Nagasawa, and S. Watanabe. Branching Markov processes I, II, III. *J. Math. Kyoto Univ.*, 8-9:233–278, 365–410, 95–160, 1968-1969.

[LM96] J.A. López-Mimbela. A probabilistic approach to existence of global solutions of a system of nonlinear differential equations. In *Fourth Symposium on Probability Theory and Stochastic Processes (Spanish) (Guanajuato, 1996)*, volume 12 of *Aportaciones Mat. Notas Investigación*, pages 147–155. Soc. Mat. Mexicana, México, 1996.

[Maz04] C. Mazza. Simply generated trees, B-series and Wigner processes. *Random Structures Algorithms*, 25(3):293–310, 2004.

[McK75] H.P. McKean. Application of Brownian motion to the equation of Kolmogorov-Petrovskii-Piskunov. *Comm. Pure Appl. Math.*, 28(3):323–331, 1975.

[MMMKV17] R.I. McLachlan, K. Modin, H. Munthe-Kaas, and O. Verdier. Butcher series: a story of rooted trees and numerical methods for evolution equations. *Asia Pac. Math. Newsl.*, 7(1):1–11, 2017.

[NS69] M. Nagasawa and T. Sirao. Probabilistic treatment of the blowing up of solutions for a nonlinear integral equation. *Trans. Amer. Math. Soc.*, 139:301–310, 1969.

[PP22] G. Penent and N. Privault. Existence and probabilistic representation of the solutions of semilinear parabolic PDEs with fractional Laplacians. *Stochastics and Partial Differential Equations: Analysis and Computations*, 10:446–474, 2022.

[SH21] Z. Selk and H. Harsha. A Feynman-Kac type theorem for ODEs: Solutions of second order ODEs as modes of diffusions. Preprint arXiv:2106.08525, 16 pages, 2021.

[Ski92] J. Skilling. Bayesian solution of ordinary differential equations. In C. R. Smith, G. J. Erickson, and P. O. Neudorfer, editors, *Proceedings of the Eleventh International Workshop on Maximum Entropy and Bayesian Methods of Statistical Analysis*, volume 50 of *Fundamental Theories of Physics*, pages 23–38. Kluwer Academic Publishers Group, Dordrecht, 1992.

[Sko64] A.V. Skorokhod. Branching diffusion processes. *Teor. Verojatnost. i. Primenen.*, 9:492–497, 1964.