Automatic Differentiation of Programs with Discrete Randomness

Gaurav Arya
Massachusetts Institute of Technology, USA
aryag@mit.edu

Moritz Schauer
Chalmers University of Technology, Sweden
University of Gothenburg, Sweden
smoritz@chalmers.se

Frank Schäfer
Massachusetts Institute of Technology, USA
University of Basel, Switzerland
franksch@mit.edu

Chris Rackauckas
Massachusetts Institute of Technology, USA
Julia Computing Inc., USA
Pumas-AI Inc., USA
crackauc@mit.edu

Abstract

Automatic differentiation (AD), a technique for constructing new programs which compute the derivative of an original program, has become ubiquitous throughout scientific computing and deep learning due to the improved performance afforded by gradient-based optimization. However, AD systems have been restricted to the subset of programs that have a continuous dependence on parameters. Programs that have discrete stochastic behaviors governed by distribution parameters, such as flipping a coin with probability p of being heads, pose a challenge to these systems because the connection between the result (heads vs tails) and the parameters (p) is fundamentally discrete. In this paper we develop a new reparameterization-based methodology that allows for generating programs whose expectation is the derivative of the expectation of the original program. We showcase how this method gives an unbiased and low-variance estimator which is as automated as traditional AD mechanisms. We demonstrate unbiased forward-mode AD of discrete-time Markov chains, agent-based models such as Conway's Game of Life, and unbiased reverse-mode AD of a particle filter. Our code package is available at https://github.com/gaurav-arya/StochasticAD.jl.

1 Introduction

Automatic differentiation (AD) is a technique for taking a mathematical program $X(p)$ and generating a new program $\widetilde{X}(p) = \frac{\mathrm{d}X}{\mathrm{d}p}$ for computing the derivative [1, 2]. AD is widely used throughout machine learning and scientific computing due to the increased performance of gradient-based optimization techniques compared to derivative-free methods [3]. However, if $X(p)$ returns the flip of a coin with probability p of receiving a 1 and probability $1-p$ of receiving a 0, it is clear that $\frac{\mathrm{d}X}{\mathrm{d}p}$ is not defined in the classical sense. But when attempting to calibrate the parameter p to data, one may wish to fit the model using statistical quantities, e.g. find p such that the average of $X(p)$ is close to the average sum of N real-world coin flips. Given this use case, can one automatically construct a program $\widetilde{X}(p)$ that computes the derivative of the statistical quantities, i.e. $\mathbb{E}[\widetilde{X}(p)] = \frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p}$?

A naïve solution to this problem would be to use finite differences, i.e.:

$$\frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p} \approx \frac{\mathbb{E}[X(p+\varepsilon)] - \mathbb{E}[X(p)]}{\varepsilon}. \tag{1.1}$$

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

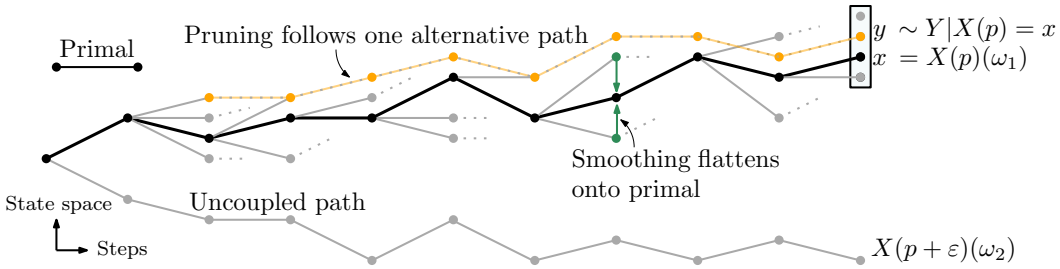


Figure 1: Qualitative sketch of our method and comparison to finite differences. The primal computation (solid black line) samples $X(p)$ with random number sequence ω_1. Black-box finite differences samples the perturbed program $X(p+\varepsilon)$ with an *independent* random number sequence ω_2 (bottom) for some *finite* choice of ε. In contrast, the component Y of the stochastic derivative of the program (Section 2) considers the effect of the minimal possible perturbations (gray lines diverging from primal path) to the original program which could stem from a truly *infinitesimal* change in the input p.

Finite differences' major issue in the context of stochastic programs is that this calculation does not correlate the calculation of $\mathbb{E}[X(p+\varepsilon)]$ with the calculation of $\mathbb{E}[X(p)]$. This leads to a large variance in the finite-difference estimator [4] that goes to ∞ as $\varepsilon \to 0$. This issue of unbounded variance could be solved if one could run the perturbed program with the same set of random numbers and directly estimate $\mathbb{E}[X(p+\varepsilon) - X(p)]$ by a method that is well-posed in the limit of $\varepsilon \to 0$ (Fig. 1).

We demonstrate how to automatically construct a new stochastic program $\widetilde{X}(p)$ whose expectation satisfies $\mathbb{E}[\widetilde{X}(p)] = \frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p}$. We derive this through a technique which we term the *stochastic derivative*, propagating the proportional probability of differing event outcomes due to infinitesimal changes in p. Our technique has the following design goals:

- **Composability.** We differentiate stochastic programs $X(p)$ which are themselves composed of many "elementary" stochastic programs, including samples from discrete distributions such as the Bernoulli, Poisson, and Geometric, and samples from continuous distributions. These may be composed (chained, added, etc.) arbitrarily, with computational cost independent of how they are composed.
- **Unbiasedness.** The program's discrete structure is preserved, leading to a provably unbiased estimator that avoids continuous relaxations and tunable or learned internal parameters.
- **Low variance.** We consider correlated paths through the computation that are linked by the smallest possible perturbation (Fig. 1), generalizing the widely-used pathwise gradient estimator for continuous randomness [5] to the discrete case.

We show the utility of our technique by demonstrating forward-mode AD of stochastic simulations like inhomogeneous random walks and agent-based models such as the Game of Life. To achieve $\mathcal{O}(1)$ computational overhead for these applications, we combine stochastic derivatives with an online strategy we call *pruning*. We also demonstrate a straightforward way to perform reverse-mode AD from our approach via *smoothing*, allowing us to derive from first principles the biased but empirically successful straight-through gradient estimator [6] as a special case and to construct an unbiased end-to-end reverse-differentiable particle filter, recovering a technique discovered in [7]. We provide an open-source implementation of the method, `StochasticAD.jl`, for readers to explore the technique on their own applications.

1.1 Related work

Gradient estimators for stochastic functions can be divided into three different classes [5]: score-function [8, 9, 10], measure-valued [11, 12], and pathwise [13, 14] gradient estimators. When it comes to discrete randomness, the score-function method is a popular general-purpose choice because it is unbiased and can be composed through stochastic computation graphs [15]. However, the score-function method does not search for correlated paths through the computation and thus suffers from high variance, making gradient computation with discrete variables challenging. A

number of techniques (REBAR [16], RELAX [17]) have been introduced that rely on control variates for variance reduction [18]. Measure-valued derivatives also have no notion of intrinsic coupling, though coupling can be achieved using common random numbers for certain distributions [19]. As an alternative direction, Gumbel-Softmax [20] considers a continuous relaxation of discrete programs so that a pathwise gradient estimator, based on the "reparameterization trick" [14], can be applied. However, such methods face a bias-variance tradeoff and are inapplicable to discrete programs that cannot be continuously relaxed. Our stochastic derivatives also extend the pathwise gradient estimator to discrete programs but do so unbiasedly. The conceptual starting point of our approach is finite differences with common random numbers [21, 22, 23], whose ideas have also been extended by direct optimization [24, 25], but crucially we show how to take the exact limit of step size $\varepsilon \to 0$ even in the discrete case. The field of smoothed perturbation analysis [26, 27] develops a mathematically equivalent object to our smoothed stochastic derivative based on conditional expectations, which is a special case of our formalism, and has also considered a randomized approach similar to our pruning technique in the context of generalized semi-Markov processes [28]. However, these ideas have not previously been applied to construct a general-purpose AD method via rigorous composition rules and the algorithms and data structures to realize the approach automatically.

Although our method can be used to hand-derive a gradient estimator, the main feature is composability through user-written functions, enabling an automated mechanism. While mainstream AD frameworks do not support unbiased differentiation of discrete random programs, Storchastic [29] is a specialized framework for AD of stochastic computation graphs [15] where the user can specify which estimator to use at each node, as well as any tunable hyperparameters. Storchastic implements an exhaustive set of prior gradient estimation methods at each sampling step. However, the runtime of the derivative estimate is in general exponential in the length of the largest chain of stochastic nodes, an artifact of the way many prior gradient estimators compose [29, 30]. In Section 3 we demonstrate stochastic AD that matches the computational complexity of deterministic AD even when discrete random functions are chained together, alleviating these performance issues.

2 Composable derivatives of stochastic programs

In this section, we develop the notion of a *stochastic derivative* for programs containing discrete randomness. We shall motivate this object as a natural generalization of the pathwise gradient estimator for continuous random programs, and present the key ideas underpinning the formalism. When describing infinitesimal asymptotics, we say a function $g(\varepsilon)$ is $\mathcal{O}(\varepsilon)$ if $|g(\varepsilon)| \leq C|\varepsilon|$ for some real C and all sufficiently small ε. Colloquially, we describe quantities that are $\mathcal{O}(\varepsilon)$ as "infinitesimal".

2.1 Infinitesimally perturbing a stochastic program

We are interested in differentiating stochastic programs, formally defined below. A stochastic program can be thought of as a map from an input p to a random variable $X(p)$. Here, $X(p)$ can represent either an "elementary" program such as a draw from a Bernoulli distribution, or the full user-provided stochastic program represented using many elementary programs, e.g. a simulation of a random walk.

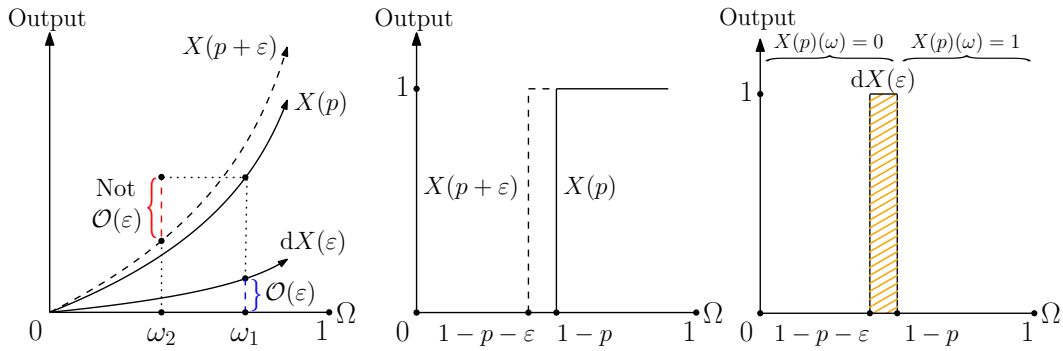
Definition 2.1. A *stochastic program* $X(p)$ is a stochastic process with values in a Euclidean space E, whose index set I is either an open subset of a Euclidean space or a closed real interval.

Let Ω be the sample space, equipped with a probability distribution $\mathbb{P}$. To sample $X(p)$ at input $p \in I$, which we call the *primal evaluation*, one should imagine a sample ω being randomly chosen from Ω according to $\mathbb{P}$ to produce an output $X(p)(\omega) \in E$. Note that $\mathbb{P}$ is independent of p and $X(p)$ is a map $\Omega \to E$; such a formulation has been called the "reparameterization trick" [14]. For example, a Bernoulli distribution $\mathrm{Ber}(p)$ can be represented by choosing a uniform random $\omega \in [0, 1]$ and defining $X(p)(\omega) = \mathbf{1}_{[1-p,1]}(\omega)$, where $\mathbf{1}_S$ is the indicator function for a set S (see Fig. 2b).

At fixed $p \in I$ we define the differential $\mathrm{d}X(\varepsilon)$, which is itself a stochastic program:

$$\mathrm{d}X(\varepsilon) = X(p + \varepsilon) - X(p). \tag{2.1}$$

Let us now restrict our attention to the case where I is a closed real interval, so that $p, \varepsilon \in \mathbb{R}$. The sensitivity of stochastic programs Z with more general index sets can be understood at an input $\mathbf{u}$ by studying at $p = 0$ the directional perturbation $X(p) = Z(\mathbf{u} + p\mathbf{v})$ in a direction $\mathbf{v}$.



(a) $X(p) \sim \mathrm{Exp}(p)$. (b) $X(p) \sim \mathrm{Ber}(p)$. (c) $X(p) \sim \mathrm{Ber}(p)$.

Figure 2: Illustrations of $\mathrm{d}X(\varepsilon)$ for a small finite ε, assuming $\varepsilon > 0$ for simplicity. **(a)** The $\mathcal{O}(\varepsilon)$ range of $\mathrm{d}X(\varepsilon)$ for continuous $X(p)$. The blue bracket indicates a sample from $\mathrm{d}X(\varepsilon)$, while the red bracket indicates a sample obtained by black-box finite differences. **(b)** The original program $X(p)$ and the perturbed program $X(p+\varepsilon)$ for discrete (Bernoulli) $X(p)$. **(c)** Only those samples ω below the hatched area cause a non-zero change in output. The hatched area equals $\mathbb{E}[\mathrm{d}X(\varepsilon)]$, the expected size of a change for randomly drawn ω.

For the full user-provided stochastic program, we are interested in computing $\frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p}$, which is related to $\mathbb{E}[\mathrm{d}X(\varepsilon)]$ by

$$\frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p} = \lim_{\varepsilon \to 0} \frac{\mathbb{E}[\mathrm{d}X(\varepsilon)]}{\varepsilon}. \tag{2.2}$$

We thus expect $\mathrm{d}X(\varepsilon)$ to have "infinitesimal" $\mathcal{O}(\varepsilon)$ expectation as ε approaches 0. Let us consider the form of $\mathrm{d}X(\varepsilon)$ for two elementary programs: a program returning a sample from the continuous Exponential distribution $\mathrm{Exp}(p)$ with scale p and a program returning a sample from the discrete Bernoulli distribution $\mathrm{Ber}(p)$ with success probability p. In both cases, the program is parameterized via the inversion method [31] over the sample space $\Omega = [0, 1]$, which chooses $\mathbb{P}$ to be uniform over $[0, 1]$ and $X(p)(\omega)$ to be non-decreasing in ω. For $X(p) \sim \mathrm{Exp}(p)$, $X(p)(\omega) = -p \log(1-\omega)$ is differentiable in p. Thus, the range of $\mathrm{d}X(\varepsilon)$ is $\mathcal{O}(\varepsilon)$ for any fixed ω, as illustrated in Fig. 2a.

In contrast, consider $X(p) \sim \mathrm{Ber}(p)$, with $X(p)(\omega) = \mathbf{1}_{[1-p,1]}(\omega)$. As shown in Fig. 2b, as $\varepsilon \to 0$ the output of $X(p)$ at a random $\omega \in \Omega$ almost surely does not change in $X(p+\varepsilon)$. Specifically, $\mathrm{d}X(\varepsilon)$ is 0 with probability $1-\varepsilon \approx 1$, and assumes the value 1 with probability ε (Fig. 2c). This is the fundamental challenge presented by discrete randomness: a $\mathcal{O}(\varepsilon)$ change in the input turns into a *finite* perturbation to the output, where "finite" means that it is non-vanishing as $\varepsilon \to 0$. The perturbation occurs with "infinitesimal" $\mathcal{O}(\varepsilon)$ probability, contributing to the $\mathcal{O}(\varepsilon)$ expectation of $\mathrm{d}X(\varepsilon)$.

2.2 Coupling to the primal

To produce low-variance estimates of the derivative of $X(p)$, it is important that the primal and derivative computations are coupled by a shared source of randomness. To see this, let us understand what happens when they are entirely uncoupled. A black-box finite difference approach with step size ε would independently sample ω_1 and ω_2 from Ω (Fig. 2a, red), computing the derivative estimate $(X(p+\varepsilon)(\omega_2) - X(p)(\omega_1))/\varepsilon$. However, since the samples are independent, the variance of the estimator is of order $1/\varepsilon^2$, so we are forced to pick a finite ε to balance a bias-variance tradeoff.

This motivates using the same random sample for the primal and derivative computations (Fig. 2a, blue). For continuous randomness, taking the limit of this approach as $\varepsilon \to 0$ leads to the widely-used pathwise gradient estimator δ, given as the almost sure limit of $\mathrm{d}X(\varepsilon)/\varepsilon$ as $\varepsilon \to 0$ (i.e. the pointwise derivative of $X(p)$ at each fixed ω), so that

$$\frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p} = \lim_{\varepsilon \to 0} \frac{\mathbb{E}[\mathrm{d}X(\varepsilon)]}{\varepsilon} \stackrel{?}{=} \mathbb{E}\left[\lim_{\varepsilon \to 0} \frac{\mathrm{d}X(\varepsilon)}{\varepsilon}\right] = \mathbb{E}\left[\delta\right]. \tag{2.3}$$

But, considering a simple Bernoulli variable $X(p) \sim \mathrm{Ber}(p)$ as in Fig. 2c, we see how this approach is ill-suited for the discrete case! As ε approaches 0, the differential $\mathrm{d}X(\varepsilon)$ is non-zero with

infinitesimal $\mathcal{O}(\varepsilon)$ probability. This means that δ is almost surely 0, while the true derivative of $\mathbb{E}[X(p)] = p$ is 1. The finite perturbation is neglected.

Thus, the pathwise gradient estimator needs to be modified to handle discrete randomness. The issue with the interchange of limit and expectation in (2.3) is that $\mathrm{d}X(\varepsilon)/\varepsilon$ is unbounded in ε in the presence of a finite perturbation. This motivates explicitly considering the event of a large jump in $\mathrm{d}X(\varepsilon)$, as characterized by the event $A_B(\varepsilon) = \{|\mathrm{d}X(\varepsilon)| > B|\varepsilon|\}$ for a chosen random bound $B > |\delta|$. The event $A_B(\varepsilon)$ has $\mathcal{O}(\varepsilon)$ probability, which is vanishingly small, but its contribution to the derivative estimate cannot be neglected because it contains finite perturbations. This motivates sampling from this part of the probability space *separately*. Formally, we introduce a random weight $w \in \mathbb{R}$ and alternate value $Y \in E$ that characterize the sensitivity of $X(p)$ when the probability space is *restricted* to $A_B(\varepsilon)$ [as given by the r.h.s. of (2.4) below], forming the "stochastic derivative":

Definition 2.2 (Stochastic derivative)**.** Suppose $X(p) \in E$ is a stochastic program with index set I a closed real interval. We say that the triple of random variables (δ, w, Y), with $w \in \mathbb{R}$ and $Y \in E$, is a right (left) *stochastic derivative* of X at the input $p \in I$ if $\mathrm{d}X(\varepsilon)/\varepsilon \to \delta$ almost surely as $\varepsilon \to 0$, and there is an integrable (i.e. of bounded expectation) random variable $B > |\delta|$ such that for all bounded functions $f\colon E \to \mathbb{R}$ with bounded derivative it holds almost surely that

$$\mathbb{E}\left[w\left(f(Y) - f(X(p))\right) \mid X(p)\right] = \lim_{\varepsilon \to 0^{+/-}} \mathbb{E}\left[\frac{f(X(p+\varepsilon)) - f(X(p))}{\varepsilon} \mathbf{1}_{A_B(\varepsilon)} \;\middle|\; X(p)\right], \quad (2.4)$$

with limit taken from above (below), where $\mathbb{P}\left(A_B(\varepsilon) \mid X(p)\right)/\varepsilon$ is dominated by an integrable random variable for all $\varepsilon > 0$ ($\varepsilon < 0$).

A stochastic derivative may be collapsed into an unbiased estimator of the derivative of $\mathbb{E}[X(p)]$.

Proposition 2.3 (Unbiasedness)**.** *If (δ, w, Y) is a stochastic derivative of $X(p)$ at p, it holds that*

$$\frac{\mathrm{d}\mathbb{E}\left[X(p)\right]}{\mathrm{d}p} = \mathbb{E}[\delta + w\left(Y - X(p)\right)]. \quad (2.5)$$

Proof sketch. With f as identity, by Definition 2.2 the sensitivity of $X(p)$ over $A_B(\varepsilon)$ is given by $w(Y - X(p))$. Given the complement event $A_B^c(\varepsilon)$, it holds that $|\mathrm{d}X(\varepsilon)/\varepsilon| \leq B$: a dominated convergence argument shows that the sensitivity of $X(p)$ is then given by its almost-sure derivative δ. □

Theorem 2.4 shows the existence of the stochastic derivative for a stochastic program, subject to technical assumptions given in our formal treatment (Appendix B).

Theorem 2.4 (Existence, simplified)**.** *Given a sufficiently regular stochastic program $X(p)$ with index set I a closed interval $[a, b] \subset \mathbb{R}$, there exists a right stochastic derivative (δ, w_R, Y_R) with $w_R \geq 0$ at any $p \in [a, b)$ and a left stochastic derivative (δ, w_L, Y_L) with $w_L \leq 0$ at any $p \in (a, b]$.*

Proof sketch. The proof is by construction: at a high level, w is the derivative of the probability of a large jump, while Y follows the distribution of the possible jumps, conditional on a jump happening. Specifically, Y has distribution given as the limit as $\varepsilon \to 0$ of the conditional distribution of $X(p+\varepsilon) = X(p) + \mathrm{d}X(\varepsilon)$ given the event $A_B(\varepsilon)$ and the outcome of $X(p)$. The weight w is given as the derivative w.r.t. ε of the probability $\mathbb{P}\left(A_B(\varepsilon) \mid X(p)\right)$ that $\mathrm{d}X(\varepsilon)$ jumps by a non-infinitesimal amount, conditional on $X(p)$. Essentially, since $\mathbb{P}\left(A_B(\varepsilon) \mid X(p)\right) \approx w\varepsilon$, multiplying by w bridges the gap between *conditioning* on $A_B(\varepsilon)$ [recall Y is constructed conditional on $A_B(\varepsilon)$] and simply restricting the probability space to $A_B(\varepsilon)$ [i.e. multiplying by $\mathbf{1}_{A_B(\varepsilon)}$ as in (2.4)]. □

Example 2.5 (Right stochastic derivative of Bernoulli variable)**.** Suppose $X(p) \sim \mathrm{Ber}(p)$, parameterized via the inversion method, and take $\varepsilon > 0$. As shown in Fig. 2c, $\mathrm{d}X(\varepsilon)$ is parameterized as,

$$\mathrm{d}X(\varepsilon)(\omega) = \begin{cases} 1 & \text{if } 1 - p - \varepsilon \leq \omega < 1 - p, \\ 0 & \text{otherwise.} \end{cases} \quad (2.6)$$

Given the event $X(p) = 1$, we have that $\omega \geq 1 - p$ and $\mathrm{d}X(\varepsilon)$ is deterministically 0. On the other hand, given $X(p) = 0$, we have $\omega < 1 - p$, so with probability $\varepsilon/(1-p)$ the differential $\mathrm{d}X(\varepsilon)$ assumes a value of 1, i.e. the Bernoulli variable flips from 0 to 1. Thus, we may construct a right

stochastic derivative $(0, w_R, Y_R)$ of $X(p)$ by letting $w_R = 1/(1-p), Y_R = 1$ conditionally on $X(p) = 0$ and $w_R = Y_R = 0$ conditionally on $X(p) = 1$. For a concrete example, with $p = 0.6$, we have $w_R = 2.5, Y_R = 1$ conditionally on $X(p) = 0$; `StochasticAD.jl` prints this as `"0 + (1 with probability 2.5ε)"` (as explained further in Section 3.1).

We give a number of examples of stochastic derivatives in Appendix A. It is of crucial importance that w and Y depend conditionally on the output of $X(p)$. In particular, given a primal evaluation $X(p) = x$, the forms of w and Y depend only on the distribution of the perturbed program on the set $\{\omega : X(p)(\omega) = x\} \subset \Omega$, which elegantly generalizes the coupling achieved by the pathwise gradient estimator δ at each fixed $\omega \in \Omega$. Intuitively, this coupling allows us to consider only the smallest possible perturbations to the program in the derivative computation (Fig. 1), and thereby achieve variance reduction without resorting to continuous relaxations. For example, for a binomial variable $X(p) \sim \mathrm{Bin}(n, p)$ parameterized via the inversion method (a natural parameterization for maximizing coupling), it holds that $Y \in \{X(p) - 1, X(p) + 1\}$. We show in Example A.1 that our gradient estimator for such a binomial variable has variance of order n, whereas the score function estimator has variance of order n^3, justifying this intuition.

2.3 Composition of stochastic derivatives

Proposition 2.3 shows that the derivative estimate produced by stochastic derivatives is correct in expectation. But this is insufficient to ensure *composition*, i.e. a stochastic derivative "chain rule". While Definition 2.2 requires composition through deterministic test functions f to enforce a sufficiently strict definition, Theorem 2.6 provides a general-purpose composition result through any program which has a stochastic derivative, as well as multidimensional programs with directional stochastic derivatives (i.e. stochastic derivatives of directional perturbations of the program).

Theorem 2.6 (Chain rule, simplified)**.** *Consider independent stochastic programs X_1 and X_2 and their composition $X_2 \circ X_1$. Suppose that X_1 has a right (left) stochastic derivative at $p \in \mathbb{R}$ given by (δ_1, w_1, Y_1), and X_2 has a right stochastic derivative (δ_2, w_2, Y_2) in the direction $\widehat{\delta}_1 = \delta_1/|\delta_1|$ given conditionally on its input $X_1(p)$. Then, under regularity and integrability assumptions, the stacked program $[X_1; X_2 \circ X_1]$ has a right (left) stochastic derivative (δ, w, Y) at p where $\delta = [\delta_1; |\delta_1|\delta_2]$,*

$$Y \quad = \quad \begin{cases} [Y_1; X_2(Y_1)] & \textit{with probability} \quad \dfrac{w_1}{w_1 + |\delta_1| w_2}, \\ [X_1(p); Y_2] & \textit{with probability} \quad \dfrac{|\delta_1| w_2}{w_1 + |\delta_1| w_2}, \end{cases} \tag{2.7}$$

and $w = w_1 + |\delta_1| w_2$.

Proof sketch. Let $A_1(\varepsilon)$ be the event of a jump in X_1 and $A_2(\varepsilon)$ be the event of a jump in X_2 when its input is perturbed by $\varepsilon\delta_1$. Given $A_1^c(\varepsilon) \cap A_2^c(\varepsilon)$ (a jump in neither), a dominated convergence argument implies that $X_2 \circ X_1$ does not jump either. Given $A_1(\varepsilon)$, Definition 2.2 yields that the sensitivity of $X_2 \circ X_1$ is described by alternate values Y_1 with weight w_1, while given $A_2(\varepsilon)$ the sensitivity of $X_2 \circ X_1$ is described by alternate values Y_2 with weight $|\delta_1| w_2$. We may then form w as the sum of these weights and Y as a weighted distribution over the two cases. Crucially, we may neglect the sensitivity of $X_2 \circ X_1$ given $A_1(\varepsilon) \cap A_2(\varepsilon)$ (a jump in both), as this event has probability $\mathcal{O}(\varepsilon^2)$. This prevents a combinatorial explosion in the complexity of Y. □

2.4 Smoothed stochastic derivatives

The reparameterization trick neglects finite perturbations, while stochastic derivatives precisely capture all possible finite perturbations. There exists a middle ground between these methods: one may take a conditional expectation on $X(p)$ so that finite perturbations have been "smoothed" into infinitesimal ones. (E.g. `"0 + (1 with probability 2.5ε)"` becomes `"0 + 2.5ε"`.)

Definition 2.7 (Smoothed stochastic derivative)**.** For a stochastic program $X(p)$ with a right (left) stochastic derivative (δ, w, Y) at input p, a right (left) smoothed stochastic derivative $\widetilde{\delta}$ of X at input p is given as

$$\widetilde{\delta} = \mathbb{E}\left[\delta + w(Y - X(p)) \mid X(p)\right]. \tag{2.8}$$

```julia
struct StochasticTriple
    value # primal evaluation
    δ # "infinitesimal" component
    Δs # component of discrete change
       # with "infinitesimal"
       # probability
end
```

```julia
using Distributions
function X(p)  p = 0.6 + ε
    a = p^2  0.36 + 1.2ε
    b = rand(Binomial(10, p))
     6 + (1 with probability 10.0ε)
    c = 2 * b + 3 * rand(Bernoulli(p))
     12 + (3 with probability 12.5ε)
    return a * c * rand(Normal(b, a))
end
```

```julia
julia> using StochasticAD
julia> st = stochastic_triple(X, 0.6) # sample a single stochastic triple at p = 0.6
 27.11 + 94.32ε + (6.78 with probability 12.5ε)
julia> derivative_contribution(st) # which produces a single derivative estimate...
 179.04
julia> samples = [derivative_estimate(X, 0.6) for i in 1:1000] # take many estimates!
julia> println("d/dp of E[X(p)]: $(mean(samples)) ± $(std(samples) / sqrt(1000))")
 d/dp of E[X(p)]: 204.63 ± 1.25
```

Figure 3: **Left:** Stochastic triple structure (simplified). **Right:** A toy program $X(p)$, using discrete distributions $\mathrm{Bin}(n, p)$, $\mathrm{Ber}(p)$, and the continuous normal distribution $\mathcal{N}(\mu, \sigma)$; the used stochastic derivatives are given in Appendix A. Highlights show intermediate values during a single derivative estimate. **Bottom:** Differentiating $\mathbb{E}[X(p)]$; printout float precision reduced for clarity.

Smoothed stochastic derivatives easily permit reverse-mode AD instead of forward-mode, as they have the same form as the usual derivative. However, they enjoy more limited composition properties: they propagate exactly through differentiable functions f that are linear over the conditional distribution of Y given $X(p)$ via the standard chain rule, as we prove in Appendix B.6. Due to the coupling of Y and $X(p)$, this is a much weaker requirement than global linearity, which can lead to low-bias estimates: for example, for the program $X(p) = \mathrm{Geo}(p)^3$ smoothed stochastic derivatives give a derivative estimate with $< 0.5\%$ bias at $p = 0.01$, even though the cube function is highly non-linear on the inter-quartile range $[28, 137]$ of $\mathrm{Geo}(p)$. Smoothed stochastic derivatives recover the widely-used straight-through gradient estimator [6] as a special case, as we work out in Example A.8.

3 Automatic differentiation of stochastic programs

We develop `StochasticAD.jl`, a prototype package for stochastic AD based on our theory of stochastic derivatives. As discussed in Section 2.4, *smoothed* stochastic derivatives obey the usual chain rule, and thus can be used with existing AD infrastructure by supplying custom rules for discrete random constructs, and we do so for a particle filter in Section 3.4. However, performing automatic differentiation with unsmoothed stochastic derivatives, which are unbiased in all cases, requires new innovation. We develop a novel computational object called a *stochastic triple*, introduced in Section 3.1 and showcased in Section 3.2 and Section 3.3.

3.1 Educational toy example of stochastic triples

Forward-mode AD is often implemented with *dual numbers* [2], which pair the primal evaluation of a deterministic function $f(p)$ with its derivative $\frac{\mathrm{d}}{\mathrm{d}p} f(p)$. Dual numbers can be propagated through a program using the chain rule. A useful alternative perspective of dual numbers is that they propagate an "infinitesimal" perturbation ε to the input through the program, where $\frac{\mathrm{d}}{\mathrm{d}p} f(p)$ is the coefficient of the "dual" element ε. Stochastic triples generalize dual numbers by including a third component Δs to describe finite perturbations with infinitesimal *probability* (Fig. 3, left).

In Fig. 3, we consider a toy program $X(p)$ including discrete randomness. We are interested in the derivative of $\mathbb{E}[X(p)]$ at $p = 0.6$, and hence we provide the stochastic triple printed as `"0.6 + ε"` as input. First, the triple is squared and becomes `"0.36 + 1.2ε"`: this is the familiar way that dual numbers propagate, via the chain rule. But what happens when the triple is propagated

(a)

```
...
X = 0
for step in 1:n
  i = rand(Categorical(probs(X)))
  X += steps[i]
end
return f(X)
```

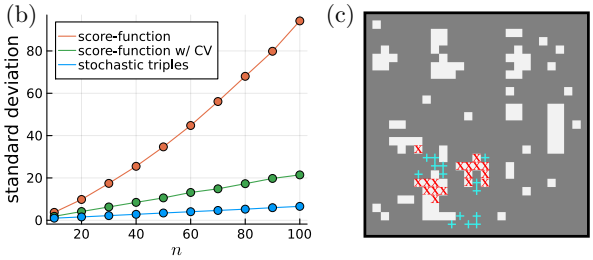


Figure 4: Automatic differentiation of discrete-time Markov processes. **(a)** Code snipped for a 1D random walk, which can be automatically differentiated by `StochasticAD.jl`; `probs(X)` gives the transition probabilities at value `X` and `steps[i]` gives the step size for the `i`th transition. **(b)** The variance of unbiased gradient estimates of the random walk program using stochastic triples and the score function, which is applied both without a control variate (CV) and with a pre-computed batch-average CV. **(c)** The final board for one run of the stochastic Game of Life with $N = 25$ and $T = 10$, where the "+" signs represent additional living cells (white) in the stochastic alternative path, and the "X" signs represent additional dead cells (grey).

through the discrete and random Binomial variable? The resultant stochastic triple `"6 + (1 with probability 10.0ε)"` is an integer with a component of *discrete* change, reflecting an infinitesimal probability of one more success in the Binomial $\mathrm{Bin}(10, 0.6)$. We can in fact understand why the probability is `"10ε"` by representing the Binomial as the sum of 10 Bernoulli variables, each with probability 0.6. Since 4 of the Bernoulli's have an output of 0, they each have a probability `"2.5ε"` of switching to 1 (recall Example 2.5), and thus there is in total a `"10ε"` probability that the output of the Binomial increases by 1 (rigorously, we have applied Theorem 2.6). Formally, we can interpret a printout `"x + δε + (Δ with probability wε)"` as follows: `x` is a sample of the random variable $X(p)$ describing the primal evaluation, and `δ`, `w`, and `x+Δ` are samples of the components δ, w, and Y, respectively, of the stochastic derivative of X at p.

Using our chain rule for stochastic derivatives (Theorem 2.6), we write rules for propagating stochastic triples through functions via operator overloading [32], exploiting Julia's multiple dispatch feature [33]. When multiple discrete changes are possible, we pick one probabilistically: we call this strategy *pruning* (recall Fig. 1) and show its unbiasedness in Appendix B.5. For example, in line 6 of Fig. 3, right, we probabilistically choose between the perturbation to the Binomial and the perturbation to the Bernoulli, in this case picking the latter. To handle causal relationships between perturbations as in the first case of Eq. (2.7), we associate each perturbation with a tag to avoid erroneously pruning between two perturbations that occur simultaneously. Thus, stochastic triples can efficiently propagate through the full toy function written in Fig. 3. The function `derivative_estimate` creates a stochastic triple, propagates it, and collapses it into the derivative estimate `δ + wΔ`, forming an unbiased estimate of the derivative via Proposition 2.3 (Fig. 3, bottom).

3.2 Inhomogeneous random walk

We consider a Markovian random walk $x_0, \ldots, x_n$ on $\mathbb{Z}_{\geq 0}$, with transition behavior dependent on a parameter p as follows,

$$x_n = \begin{cases} x_{n-1} + 1 \text{ with probability } \exp\left(-\frac{x_{n-1}}{p}\right) \\ x_{n-1} - 1 \text{ with probability } 1 - \exp\left(-\frac{x_{n-1}}{p}\right) \end{cases}, \quad x_0 = 0. \tag{3.1}$$

We consider a program that stochastically simulates this walk and applies an arbitrary non-linear function f to the output x_n. In practice, f may represent a loss or a likelihood estimate; in this toy setting, we take $f(x) = x^2$. We are interested in studying the asymptotic behavior of the variance of our automatically derived gradient estimator, and so set $p = n$ so that the transition function varies appreciably over the range of the walk for all n. We find that the stochastic triple estimator has asymptotically lower variance than the score function estimator (Fig. 4b). Crucially, stochastic triples achieve this variance reduction while remaining entirely in discrete space, automatically producing a gradient estimate that is provably unbiased.

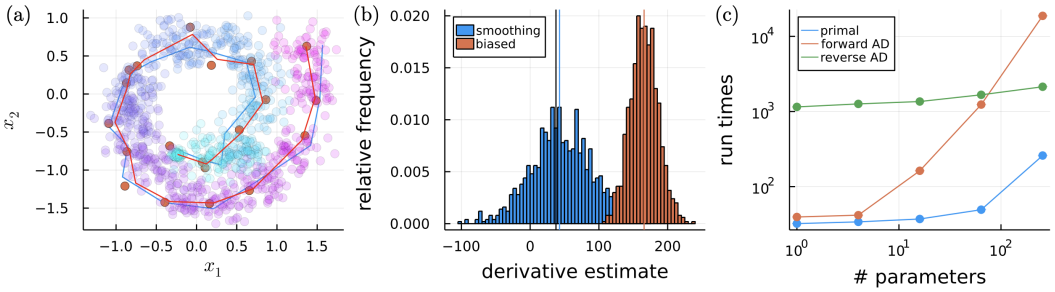


Figure 5: Unbiased differentiable particle sampler using smoothed stochastic derivatives. **(a)** Latent path (blue) and observations (orange) for a particular realization of the process given by the hidden Markov model (see Appendix C for details) in $d = 2$ dimensions, including the recursively applied Kalman filter estimator (red) and the particles of the particle filter (going from blue at $n = 1$ to pink at $n = 20$). **(b)** The value of the derivative of $\log \mathcal{L}$ with respect to the first parameter for $d = 2$ calculated by differentiating the Kalman filter is marked by a black line. The blue distribution corresponds to the derivative computed with smoothed stochastic derivatives. The orange distribution represents the biased derivative approach, where the resampling step is not differentiated. The means of the two distributions are highlighted by a line in the respective color. In both cases, we use 1000 samples. **(c)** Averaged run times in ms comparing forward- and reverse-mode AD for increasing numbers of parameters.

3.3 Stochastic Game of Life

For our next example, we differentiate a stochastic version of John Conway's Game of Life, played on a two-dimensional board. In the traditional Game of Life, a dead cell becomes alive when 3 of its neighbors are alive, while a living cell survives when 2 or 3 of its neighbors are alive. In our stochastic version, each of these events instead has probability 95%, while their complementary events have probability 5%. Such types of discrete stochastic programs arise in many applications. For example, mixing machine learning with agent-based models found in epidemiology and sociological contexts [34, 35] or rule-based models and discrete stochastic (Gillespie) simulations in systems biology [36, 37] requires similar program constructs.

Consider a program that populates each cell of an $N \times N$ board with probability p, runs the stochastic Game of Life for T time steps, and counts the number of living cells n_{living}. We perform a sensitivity analysis of the final living population with respect to the initial living population, i.e. differentiate the expectation of n_{living} with respect to p. Stochastic triples propagate fully through the program, leading to an unbiased estimate of the derivative, as we verify with black-box finite differences. An example final board is depicted in Fig. 4c, along with the difference to the alternative final board chosen by pruning. This is a high-dimensional example (the state space has dimension $N^2 = 625$) with fundamentally discrete structure, providing support for the algorithmic correctness and generality of stochastic triples. In particular, the program cannot directly be continuously relaxed since it includes array indexing; in the approach of stochastic triples, integer-valued quantities stay integers.

3.4 Particle filter

As a final example, we consider a hidden Markov model with random latent states $X_1, \ldots, X_n$, observations $y_1 \sim Y_1, \ldots, y_n \sim Y_n$, and parameters θ. The likelihood $\mathcal{L} = p(y_1, \ldots, y_n \mid \theta)$ is in general not tractable, but a particle filter can be used to compute an estimate of $\mathcal{L}$. Here, we assume familiarity with particle filters [38, 39] but provide a short exposition with focus on the resampling step and experimental details in Appendix C. We assume the latent states to be continuous random, but discrete randomness enters through the resampling step.

To differentiate the particle filter resampling step, we provide a stochastic derivative formulation of the returned particles and importance weights (Appendix C). Importantly, $\mathcal{L}$ can be expressed by the sum of the weights at the last step, and the weight assigned to each particle in the resampling steps

is used in a *purely linear* way. Smoothed stochastic derivatives permit *unbiased* reverse-mode AD in this case. Our formulated approach, though derived very differently, is equivalent to the particle filter AD scheme developed in [7] implementing the first estimator derived in [40], as we show in Appendix C. Our particular choice of system allows the calculation of a ground-truth gradient of $\mathcal{L}$ by differentiating the Kalman filter algorithm [41]. Fig. 5a visualizes the latent process and observations, and the Kalman and particle filter trajectories. Our estimator agrees with the Kalman filter derivative, unlike biased estimators [42, 43, 44] that neglect the contribution of the resampling step or perform it with entropy-regularized optimal transport (Fig. 5b). For our program, we observe reverse-mode AD to perform better for more than ≈ 100 parameters (Fig. 5c). However, we find that the variance increases more rapidly with the number of steps and dimension as compared to biased estimators, suggesting that there is room for improvement in the coupling approach [7, 40].

4 Limitations and outlook

We have presented a method for unbiased AD of programs with discrete randomness, which, we have argued, is a natural generalization of pathwise gradient estimators based on the reparameterization trick to the discrete case. However, more work will need to be done to turn our software demonstration `StochasticAD.jl` into an AD system capable of handling the full complexity of applications such as machine learning. A useful improvement would be better support for discrete constructs such as `if` statements with discrete random input; currently, such branches need to be rewritten using array indexing, which is supported. Further, an interesting direction for future work is to automatically handle functions which deterministically turn a continuous random quantity into a discrete one, such as an inequality comparison `X(p) > 0` or a Bernoulli variable implemented implicitly as `rand() < p` and also to handle constructs such as while loops based on such constructs.

We also expect future work to focus on further variance reduction. Our method's variance depends on the degree to which the primal and derivative computations can be *coupled*: while we present a natural method of coupling for a number of elementary stochastic programs and their compositions, the design space is rich when it comes to challenging examples such as the Game of Life or the resampling step of a particle filter. (This design space is reflected in the fact that the form of the stochastic derivative depends not only on a program's probability distribution, but also on the way it is parameterized.) Furthermore, the pruning operation can introduce additional variance: for example, the derivative estimate automatically produced for the program $B_1 + 2B_2$ with i.i.d. $B_i \sim \mathrm{Bin}(n, p)$ has variance $\mathcal{O}(n^2)$ due to the pruning between the `+1` and `+2` perturbations, even though stochastic derivatives give gradient estimators with variance $\mathcal{O}(n)$ for each of B_1 and B_2. Smoothing does not face this issue but accrues a bias through non-linear functions. Pruning and smoothing may be thought of as the simplest ways to construct an AD algorithm from stochastic derivatives, lying on opposite ends of the design space: we anticipate future work to address their suboptimalities and ideally form unbiased estimators for discrete random programs that fully close the variance gap to their continuous counterparts [14, 45]. Finally, going beyond smoothing for reverse-mode AD, and ideally achieving unbiasedness while remaining coupled to the primal, is an important open problem for large-scale applications.

Acknowledgments and disclosure of funding

We thank Alan Edelman, Guillaume Dalle, and the anonymous reviewers for their feedback, Emile van Krieken for helpful discussions regarding composing gradient estimators, and Simeon Schaub for help with the package. We acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the research results reported within this paper. This material is based upon work supported by the National Science Foundation OAC-1835443, SII-2029670, ECCS-2029670, OAC-2103804, and PHY-2021825; the Advanced Research Projects Agency-Energy DE-AR0001211 and DE-AR0001222; the Defense Advanced Research Projects Agency (DARPA) HR00112290091; the United States Artificial Intelligence Accelerator FA8750-19-2-1000; the Chalmers AI Research Centre; and the Swiss National Science Foundation 51NF40-185902. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

[1] Andreas Griewank. On automatic differentiation. *Mathematical Programming: recent developments and applications*, 6(6):83–107, 1989.

[2] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18:1–43, 2018.

[3] George Corliss, Christele Faure, Andreas Griewank, Laurent Hascoet, and Uwe Naumann. *Automatic differentiation of algorithms: from simulation to optimization*. Springer Science & Business Media, 2002.

[4] Søren Asmussen and Peter W Glynn. *Stochastic simulation: algorithms and analysis*, volume 57. Springer, 2007.

[5] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21(132):1–62, 2020.

[6] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013.

[7] Adam Ścibior and Frank Wood. Differentiable particle filtering without modifying the forward pass. *arXiv:2106.10314*, 2021.

[8] Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

[9] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[10] Jack PC Kleijnen and Reuven Y Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, 1996.

[11] G Ch Pflug. Sampling derivatives of probabilities. *Computing*, 42(4):315–328, 1989.

[12] Bernd Heidergott and FJ Vázquez-Abad. Measure-valued differentiation for Markov chains. *Journal of Optimization Theory and Applications*, 136(2):187–209, 2008.

[13] Paul Glasserman and Yu-Chi Ho. *Gradient estimation via perturbation analysis*, volume 116. Springer Science & Business Media, 1991.

[14] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv:1312.6114*, 2013.

[15] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems*, 28, 2015.

[16] George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems*, 30, 2017.

[17] Will Grathwohl, Dami Choi, Yuhuai Wu, Geoffrey Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv:1711.00123*, 2017.

[18] Peter W Glynn and Roberto Szechtman. Some new perspectives on the method of control variates. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 27–49. Springer, 2002.

[19] Bernd Heidergott, Felisa J Vázquez-Abad, and Warren Volk-Makarewicz. Sensitivity estimation for Gaussian systems. *European Journal of Operational Research*, 187(1):193–207, 2008.

[20] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

[21] David F Anderson. An efficient finite difference method for parameter sensitivities of continuous time markov chains. *SIAM Journal on Numerical Analysis*, 50(5):2237–2258, 2012.

[22] Vo Hong Thanh, Roberto Zunino, and Corrado Priami. Efficient finite-difference method for computing sensitivities of biochemical reactions. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2218):20180303, 2018.

[23] Paul Glasserman and David D Yao. Some guidelines and guarantees for common random numbers. *Management Science*, 38(6):884–908, 1992.

[24] Tamir Hazan, Joseph Keshet, and David McAllester. Direct loss minimization for structured prediction. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[25] Guy Lorberbom, Chris J Maddison, Nicolas Heess, Tamir Hazan, and Daniel Tarlow. Direct policy gradients: Direct optimization of policies in discrete action spaces. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18076–18086. Curran Associates, Inc., 2020.

[26] Wei-Bo Gong and Yu-Chi Ho. Smoothed (conditional) perturbation analysis of discrete event dynamical systems. *IEEE Transactions on Automatic Control*, 32(10):858–866, 1987.

[27] Paul Glasserman and W-B Gong. Smoothed perturbation analysis for a class of discrete-event systems. *IEEE Transactions on Automatic Control*, 35(11):1218–1230, 1990.

[28] Michael C Fu and Jian-Qiang Hu. *Conditional Monte Carlo: Gradient estimation and optimization applications*, volume 392. Springer Science & Business Media, 2012.

[29] Emile van Krieken, Jakub Mikolaj Tomczak, and Annette Ten Teije. Storchastic: A framework for general stochastic automatic differentiation. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[30] Emile van Krieken, Jakub Tomczak, and Annette Ten Teije. Openreview: Storchastic: A framework for general stochastic automatic differentiation. URL https://openreview.net/forum?id=KAFyFabsK88.

[31] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

[32] Jarrett Revels, Miles Lubin, and Theodore Papamarkou. Forward-mode automatic differentiation in Julia. *arXiv:1607.07892 [cs.MS]*, 2016.

[33] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[34] Eric Silverman, Umberto Gostoli, Stefano Picascia, Jonatan Almagor, Mark McCann, Richard Shaw, and Claudio Angione. Situating agent-based modelling in population health research. *Emerging Themes in Epidemiology*, 18(1):10, Jul 2021.

[35] Ayush Chopra, Alexander Rodríguez, Jayakumar Subramanian, Balaji Krishnamurthy, B Aditya Prakash, and Ramesh Raskar. Differentiable agent-based epidemiology. *arXiv preprint arXiv:2207.09714*, 2022.

[36] Lily A Chylek, Leonard A Harris, Chang-Shung Tung, James R Faeder, Carlos F Lopez, and William S Hlavacek. Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *Wiley Interdiscip. Rev. Syst. Biol. Med.*, 6(1):13–36, January 2014.

[37] James R Faeder, Michael L Blinov, and William S Hlavacek. Rule-based modeling of biochemical systems with BioNetGen. *Methods Mol. Biol.*, 500:113–167, 2009.

[38] Nicolas Chopin and Omiros Papaspiliopoulos. *An introduction to sequential Monte Carlo*. Springer, 2020.

[39] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Oxford Handbook of nonlinear filtering*, 12(656-704), 2011.

[40] George Poyiadjis, Arnaud Doucet, and Sumeetpal S. Singh. Particle approximations of the score and observed information matrix in state space models with application to parameter estimation. *Biometrika*, 98(1):65–80, 02 2011.

[41] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.

[42] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. Differentiable particle filters: End-to-end learning with algorithmic priors. *arXiv:1805.11122*, 2018.

[43] Adrien Corenflos, James Thornton, George Deligiannidis, and Arnaud Doucet. Differentiable particle filtering via entropy-regularized optimal transport. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2100–2111. PMLR, 18–24 Jul 2021.

[44] Conor Rosato, Lee Devlin, Vincent Beraud, Paul Horridge, Thomas B. Schön, and Simon Maskell. Efficient learning of the parameters of non-linear models using differentiable resampling in particle filters. *IEEE Transactions on Signal Processing*, 70:3676–3692, 2022.

[45] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

[46] Fernando Mazzone. A characterization of almost everywhere continuous functions. *Real Analysis Exchange*, 21(1):317–319, 1995.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 4.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Appendix.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Appendix D and the provided code.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Our examples were small-scale, so we could take enough samples to achieve minimal error.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite the Julia packages on which this work is based.
   (b) Did you mention the license of the assets? [No] We only use open-source software packages.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We have included our package in the supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We do not use any previously collected data.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]