Automatic Differentiation of Programs with Discrete Randomness

Anonymous Author(s)
Affiliation
Address
email

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 manuscript we develop a new reparameterization 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.

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 the statistical quantities, i.e. 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 statistical quantities of the derivative, 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.:

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

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 $\mathbb{E}[X(p)]$. This leads to a large variance in the finite difference estimator [4] that is non-vanishing (the variance goes to ∞ as $\varepsilon \to 0$). This variance issue can be alleviated if one could run the perturbed program with the same set of random numbers, i.e. directly compute $\mathbb{E}[X(p+\varepsilon) - X(p)]$ as depicted in Figure 1. In this manuscript we

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

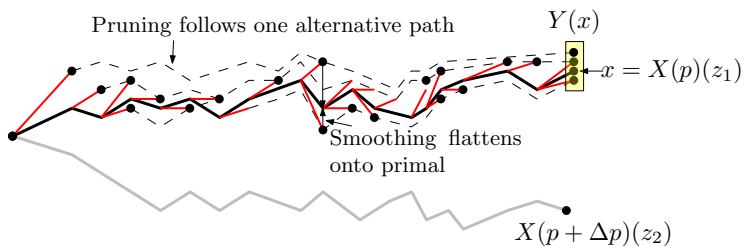


Figure 1: Qualitative sketch of the method and comparison to finite differences. The primal computation samples $X(p)$ with random number sequence z_1. Black-box finite differences samples an independent copy of the program $X'(p + \varepsilon)$ with random number sequence z_2 (bottom) for some finite choice of ε. In contrast, the stochastic derivative Y is coupled to the primal computation, considering the effect of the minimal possible perturbations (dotted black lines) to the program.

demonstrate how to automatically construct a stochastic program $\widetilde{X}(p)$ whose expectation satisfies $\mathbb{E}[\widetilde{X}(p)] = \frac{d\mathbb{E}[X(p)]}{dp}$. 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 several benefits:

- **Composability.** We derive a extension language-level AD to stochastic programs composed of any of a wide range of random "building blocks", including draws from discrete distributions such as Bernoulli, Geometric, and Poisson distributions.
- **Preservation of program structure.** A single run of the primal computation is transformed into an unbiased estimate of the derivative in a stateless fashion with no internal parameters to tune.
- **Unbiased and low variance.** The discrete structure of the original program is preserved, avoiding continuous relaxations and their associated variance-bias trade-off.
- **Speed.** Our technique's computational cost is proportional to the number of random building blocks and is independent of how they are composed together.

We show the utility of our technique by demonstrating AD of stochastic simulations like inhomogenous random walks, AD of agent-based models such as the Game of Life, and building an end-to-end differentiable particle sampler. Additionally, 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

A wide literature exists on gradient estimators [5]. Our method operates as a gradient estimator, but beyond that, the main feature is composability with arbitrary user-written functions, enabling an automated mechanism.

When it comes to discrete randomness, the score-function method is a popular general-purpose choice because it can be composed by stochastic computation graphs [6] and provides an unbiased estimate of the gradient. However, the score-function method cannot be used to, e.g., differentiate a resampling step, and has other well-known limitations [5]. Importantly, the score-function method is not coupled to the primal evaluation and suffers from high variance, making gradient computation with discrete variables challenging. A number of techniques (REBAR [7], RELAX [8]) have been introduced that rely on control variates for variance reduction [9].

The main drawback of this coupling method is that its implementation may require nontrivial changes to the primal evaluation code, which makes it less attractive for automation. In addition, measure-valued gradient estimators scale poorly with the number of parameters because two primary evaluations for each parameter are required.

As an alternative direction, several recent works have considered continuous relaxations to compute a low-variance but biased gradient estimator based on pathwise derivatives via what is known as the reparameterization trick [10, 11]. Gumbel-Softmax relaxes the entire program into continuous space at an adjustable temperature. Our methods extend the pathwise gradient estimator from continuous to discrete programs but without suffering from a variance-bias tradeoff. Smoothed perturbation analysis [12, 13] develops a mathematically equivalent object to our smoothed stochastic dual based on conditional expectations, but these ideas have not previously been applied to construct AD algorithm. enabling automatic differentiation of programs with discrete randomness low-variance unbiased estimators.

Storchastic [14] is a framework for AD of stochastic computation graphs, 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, previous reviews of these techniques revealed a performance scaling issue: the run time of the derivative estimate is exponential in the length of the largest chain of stochastic nodes [15]. In Section 4.3 we demonstrate polynomial scaling, similar to deterministic AD, alleviating these performance issues.

2 Composable stochastic derivatives

In this section, we generalize the reparameterization trick to handle discrete randomness. When offering motivation and intuition, we freely use the language of non-standard calculus [16], letting ε be an infinitesimal quantity of magnitude smaller than each fraction $1/n$, $n \in \mathbb{N}$.

2.1 Infinitesimally perturbing an elementary stochastic program

Consider a stochastic program; a program which when provided with input p gives a random output $X(p)$. To obtain a single sample of an *elementary stochastic program*, which we call the *primal evaluation*, one should imagine a number z being randomly chosen from Z to produce a then deterministic output $X(p)(z)$. For example, a Bernoulli random variable with probability p can be represented by picking a uniform random number $z \in [0, 1]$ and defining $X(p)(z) = \mathbf{1}_{[1-p,1]}(z)$ where $\mathbf{1}_\Omega(z)$ is the indicator function for $z \in \Omega$. Note that parameterizations of many distributions over a continuous sample space Z can be constructed via the inversion method [17].

A *stochastic infinitesimal* is the change in a random variable w.r.t. an infinitesimal change in its input,

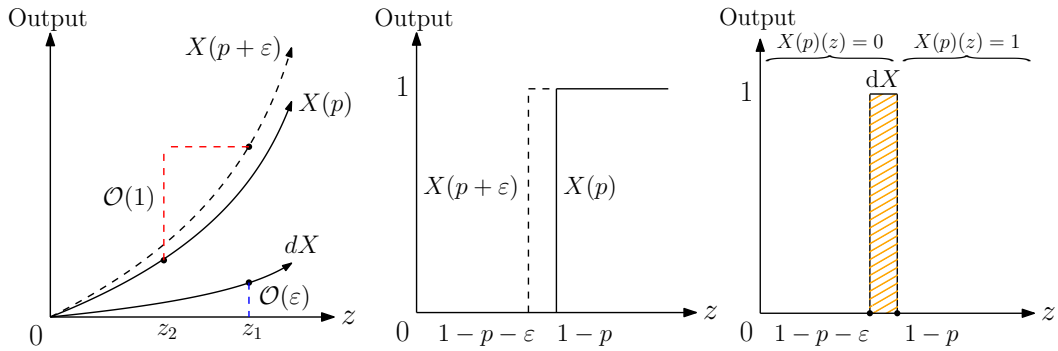
$$\mathrm{d}X = X(p + \varepsilon) - X(p), \tag{2.1}$$

where we suppress the dependence of $\mathrm{d}X$ on p. Eq. (2.1) differs from finite differences in that ε is infinitesimal, $X(p+\varepsilon)$ and $X(p)$ share the same sample space, and they are typically not independent. A stochastic infinitesimal $\mathrm{d}X$ is then itself a random variable over the same sample space. For the full user-provided stochastic program, we are interested in the task of computing $\partial_p \mathbb{E}[X(p)]$, which is related to $\mathbb{E}[\mathrm{d}X]$ by

$$\mathbb{E}[\mathrm{d}X] = \partial_p \mathbb{E}[X(p)] \cdot \varepsilon. \tag{2.2}$$

To prepare, let us consider the form of $\mathrm{d}X$ for two illustrative examples: a program returning a sample from the continuous exponential distribution $\mathrm{Exp}(p)$ and a program returning a sample from the discrete Bernoulli distribution $\mathrm{Ber}(p)$, both parametrized via the inversion method over the sample space $[0, 1]$. In the differentiably parametrized continuous case (Fig. 2a), an infinitesimal perturbation in the input p results in an infinitesimal change of the outcome $X(p)(z)$ (point-wise in z). This provides the basis for the familiar reparameterization trick. In this case, $X(p)(z) = -p \log(1 - z)$ is differentiable in p for any fixed z. Thus, the value of $\mathrm{d}X(z)$ is $\mathcal{O}(\varepsilon)$ and we find that $\mathbb{E}[\mathrm{d}X] = \mathcal{O}(\varepsilon)$, as expected from Eq. (2.2).

However, there is another way that $\mathrm{d}X$ can have infinitesimal expectation, which we see in the discrete case. When one infinitesimally perturbs the probability p of a Bernoulli variable reparametrized using the indicator function as $X(p)(z) = \mathbf{1}_{[1-p,1]}(z)$ (Fig. 2b) the output at each noise value



(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 the stochastic infinitesimal $\mathrm{d}X$, where a small finite ε is used to represent an infinitesimal ε and we take $\varepsilon > 0$ for simplicity. **Left:** Point-wise infinitesimal change for continuous $X(p)$. The blue dashed line indicates a sample from $\mathrm{d}X$, while the red dashed line indicates a sample obtained by black-box finite differences. **Middle:** Effect of infinitesimal change for discrete $X(p)$. **Right:** Only those z below the hatched area cause a non-zero change in output. The hatched area corresponds to the expected size of a change for randomly drawn z.

almost surely does not change. Instead, $\mathrm{d}X$ assumes a non-infinitesimal value of magnitude 1 with infinitesimal probability h (Fig. 2c). This is the fundamental challenge presented by discrete randomness: an infinitesimal change in the input turns into a *finite* perturbation that occurs with infinitesimal probability, contributing to the infinitesimal expectation of $\mathbb{E}[\mathrm{d}X]$.

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 z_1 and z_2 from the sample space (Fig. 2a), computing the derivative estimate $1/\varepsilon \cdot (X(p+\varepsilon/2)(z_2) - X(p-\varepsilon/2)(z_1))$. But the variance of these samples is of order $1/\varepsilon^2 = \infty$, so we are forced to pick a finite step size and balance a variance-bias tradeoff.

In the case of continuous randomness, the reparameterization trick surmounts this issue by differentiating $X(p)$ at a fixed value of the noise z (Fig. 2a). In the language of stochastic infinitesimals, this is equivalent to sampling directly from $\mathrm{d}X$. But, referring to Fig. 2c, we see how this is impossible in the discrete case! The stochastic infinitesimal $\mathrm{d}X$ is non-zero with infinitesimal probability, and thus the reparameterization trick almost surely returns 0, while the true derivative is $\partial_p(\mathbb{E}[\mathrm{Ber}(p)]) = \partial_p p = 1$.

Thus, the reparameterization trick needs to be modified to handle discrete randomness. The key idea is to consider the *conditional* distribution of $\mathrm{d}X$ given the primal output. In the continuous case, this reduces to the traditional reparameterization trick as there is a one-to-one correspondence between primal output and a sample $z \in Z$. Let us understand what happens in the discrete case.

Example 2.1 (Bernoulli variable)**.** Suppose $X(p) \sim \mathrm{Ber}(p)$, parametrized via the inversion method. Let us consider the case of infinitesimal $\varepsilon > 0$. As shown in Fig. 2c, $\mathrm{d}X$ is parameterized as,

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

If the primal output of $X(p)$ is 1, then $z > 1-p$ and $\mathrm{d}X$ is deterministically 0. On the other hand, if the primal output of $X(p)$ is 0, then with infinitesimal probability $\varepsilon/(1-p)$ the stochastic infinitesimal $\mathrm{d}X$ assumes a value of 1, i.e. the Bernoulli variable flips from 0 to 1.

Our insight is that the primal-conditioned distribution of $\mathrm{d}X$ may be decomposed into an infinitesimal component and a component consisting of finite perturbations with infinitesimal probability. We can sample from these parts of the probability space *separately*, ensuring that the finite perturbations are

not neglected and curing the degeneracy of finite changes occurring with infinitesimal probability. As in stratified sampling, the expectation is then obtained as an weighted average of the separately sampled parts, cf. [4]. Formally, this leads to the following:

Definition 2.2. Suppose $X(p)$ is a stochastic program with parametrized by a real interval, whose output belongs to a Euclidean space E. We say that the triple of random variables (δ, w, Y) with infinitesimal change $\delta \in E$, importance weight $w \geq 0$ ($w \leq 0$) and alternative outcome $Y \in E$ is a *right (left) derivative* of X at the input p if for all smooth bounded real functions f

$$\frac{\partial}{\partial p}\mathbb{E}\left[f(X(p))\right] = \mathbb{E}\left[f'(X(p)) \cdot \delta + w\left(f(Y) - f(X(p))\right)\right]. \tag{2.3}$$

Theorem 2.3. *Given a sufficiently regular stochastic program $X(p)$ with $X'(p) = \frac{\partial}{\partial p}X(p)$ almost surely defined, there exists a weight function $w \geq 0$ on E and random program Y such that the triple $(X'(p), w(X(p)), Y(X(p)))$ is a stochastic right derivative of X at p. Likewise there exists such a triple with $w \leq 0$ giving a left derivative.*

In Theorem 2.3, $X'(p)$ represents the point-wise differentiation performed by a naive application of the reparameterization trick. Note that the importance weight of this part is (infintely close to) 1. The second term corrects this estimate by including the effect of finite perturbations: for an output x of $X(p)$, $Y(x)$ is a random variable whose distribution encodes all possible finite perturbations present in the conditional distribution of $\mathrm{d}X$ given $X(p) = x$. Specifically, linking to our infinitesimal language, if a finite perturbation Δx_i occurs with infinitesimal probability $w_i\varepsilon$, then $Y(x)$ takes value $x + \Delta x_i$ with probability w_i/w, where $w(x) = \sum_i w_i$. Crucially, infinitesimal probabilities have been replaced by finite ones, so that $Y(x)$ can be easily sampled. Opting for left or right derivatives here may give different, but unbiased estimators, as one verifies considering Example 2.1.

Intuitively, the fact that we can separately consider the infinitesimal part and all finite perturbations is made possible by a form of "exclusion principle" for stochastic infinitesimals. Specifically, the event of two finite perturbations occuring simultaneously can be neglected as it has probability $\mathcal{O}(\varepsilon^2)$, and in addition a $\mathcal{O}(\varepsilon)$ infinitesimal change can also be neglected in the event of a $\mathcal{O}(1)$ change with infinitesimal probability.

The crux is that we only need to derive the conditional distribution of $\mathrm{d}X$ for each elementary stochastic program from which the full stochastic program is composed; we do so for a number of random building blocks in the supplement. The rest shall follow from compositionality, i.e. a stochastic derivative "chain rule", as we will show in the next section.

We remark on the value of coupling to the primal. Intuitively, it allows us to consider only the smallest possible changes to the program output in the derivative computation, and thereby achieve variance reduction without resorting to continuous relaxations. For example, for a binomial variable $X(p) \sim \mathrm{Bin}(n, p)$ pararameterized via the inversion method, $\mathrm{d}X \in \{-1, 0, 1\}$ irrespective of n. We derive in the supplement that the variance of our gradient estimator for such a binomial variable is $\mathcal{O}(n)$, whereas the score function estimator has variance $\mathcal{O}(n^3)$, justifying this intuition.

2.3 Composition

We have defined the notion of a stochastic derivative (Definition 2.2), and shown existence for elementary stochastic programs (Theorem 2.3). The missing ingredient is *composition*. For clarity, in the main text we consider the cases of a purely continuous and purely discrete stochastic program.

Proposition 2.4. *Consider stochastic programs $X_1(p)$ and $X_2(x_1)$, where X_1 has a stochastic derivative given by (δ_1, w_1, Y_1) and X_2 has a derivative $(\delta_2, 0, 0)$ given by Theorem 2.3. Then $X_2 \circ X_1$ has a stochastic derivative given by $(X_2(\delta), 0, 0)$.*

Proposition 2.4, where X_2 has a continuous dependence on its parameter, is a direct consequence of Definition 2.2. The key to composition is the case where X_2 is discrete random: here, an

infinitesimal perturbation to the input of X_2 leads to finite perturbations, introducing another event with infinitesimal probability to be considered by the stochastic derivative.

Proposition 2.5 (Simplified)**.** *Consider stochastic programs $X_1(p)$ and $X_2(x_1)$, where X_1 has a stochastic derivative given by (δ_1, w_1, Y_1) and X_2 has a derivative $(0, w_2, Y_2)$ given by Theorem 2.3. Then the stacked program $[X_1; X_2 \circ X_1]$ has a stochastic derivative given by $([\delta; 0], w, Y)$ where*

$$Y = \begin{cases} [Y_1; X_2(Y_1)] & \textit{with probability} \quad \dfrac{w_1}{w_1 + w_2(X_1)} \\ [X_1; Y_2(X_1)] & \textit{with probability} \quad \dfrac{w_2(X_1)}{w_1 + w_2(X_1)}, \end{cases} \tag{2.4}$$

and $w = w_1 + w_2(X_1)$.

2.4 Smoothing

For discrete random variables, the reparameterization trick is deterministically 0, while stochastic derivatives fully capture potential finite perturbations with infinitesimal probability. There exists a middle ground between these methods: one may take the *conditional expectation* of the stochastic infinitesimal $\mathrm{d}X$, such that the derivative has the correct expectation given the primal output, but a finite perturbation by an infinitesimal one. By the tower property of conditional expectation, if $\delta + w(Y - X(p))$ is an unbiased estimate of $\partial_p \mathbb{E}[X(p)]$, then so is the random number $\widetilde{\delta} = \mathbb{E}[\delta + w(Y - X(p)) \mid X(p)]$. In fact, if the function f is approximately linear in $X(p)$ over the range of $X(p) + Y$, even $\partial_p \mathbb{E} f(X(p)) \approx \mathbb{E} f'(X(p))\widetilde{\delta}$.

Smoothed stochastic derivatives enjoy limited composition properties. For example in the setting of Proposition 2.5, assuming $w_1 = 0$, let $\widetilde{\delta}_2 = w_2(X_1)\mathbb{E}\left[Y_2(X_1(p)) | X_1(p)\right]$. Then for functions f that are linear

$$\frac{\partial}{\partial p}\mathbb{E} f((X_2 \circ X_1)(p)) = \mathbb{E}\left[f'((X_2 \circ X_1(p)))\widetilde{\delta}_2\right]. \tag{2.5}$$

The supplement presents formal results regarding smoothed stochastic derivatives.

3 Automatic differentiation of stochastic programs

3.1 Stochastic triples

In AD of deterministic programs, dual numbers pair the primal evaluation of a deterministic function $f(p)$ with its derivative $\partial_p f(p)$, enabling automatic forward differentiation via operator-overloading. Fig. 3 presents the structure of *stochastic triples*, which generalize dual numbers to programs containing discrete randomness. For a random program $X(p)$ with stochastic derivative (δ, w, Y), the stochastic triple stores the result x of the primal evaluation, alongside $\delta(x)$ and a sample from $Y(x)$.

To perform forward differentiation, we use Proposition 2.4 and Proposition 2.5 to write rules for propagating this stochastic triple through elementary functions. Given the situation of Theorem 2.3, we make a probabilistic choice between the two possible finite perturbations to produce the resultant stochastic triple, where the probability of picking a particular perturbation is *weighted* by its probability. The effect of the perturbation that is not chosen is no longer considered, such that after all propagation has completed, a single "alternative path" through the compuation has been chosen for the derivative estimate. This strategy, which we call *pruning*, relates closely to the idea of importance sampling, and ensures $\mathcal{O}(1)$ computational overhead of the stochastic triple while remaining unbiased (shown formally in the supplement).

The stochastic triple propagates through a user-written stochastic program via operator overloading, as shown in Fig. 4. Afterward, we collapse it into a single estimate of the derivative via Definition 2.2, resulting in a sample from the right-hand side of Definition 2.2 with f as identity function.

Thus far we have considered only unary functions. An additional complication arises with multivariate functions: a finite perturbation associated with a stochastic program $X_1(p)$ may have stemmed from

```julia
struct StochasticTriple
    value # primal evaluation
    δ # infinitesimal change
    Δs # finite perturbations
end
```

Figure 3: Stochastic triple structure (simplified).

```julia
using StochasticAD, Distributions
f(p) = rand(Geometric(p))^3
sd = stochastic_triple(f, 0.01)
# sd = 85184 + 0ϵ + (-5677
# with probability 4444.4..ϵ)
```

Figure 4: Propagating a stochastic triple through $\mathrm{Geo}(p)^3$.

the same event with infinitesimal probability as a finite perturbation with a different stochatic program $X_2(p)$. In this case, we say they are *linked*, and the opposite of the exclusion rule discussed in Section 2 holds: the two perturbations either happen in unison or not at all. To perform linking, we associate each finite perturbation with a uniquely-identified *infinitesimal event*, i.e. an event with infinitesimal probability. In the supplement, we generalize the theory of stochastic derivatives to formally incorporate the notion of linking. With this innovation included, one may compute an unbiased estimate for the derivative of an arbitrary stochastic program via forward-mode AD.

3.2 Smoothed stochastic duals

It is not immediately clear how a stochastic triple may be backpropagated, given that they require knowledge of finite perturbations that occur at an earlier stage of the computation. The advantage of smoothing (Section 2.4) is that finite perturbations are "flattened" back onto the primal path (Fig. 1), such that the smoothed stochastic dual has an *identical* structure to a traditional dual number, permitting effortless reverse-mode AD using existing AD infrastructure. Smoothed stochastic duals are only unbiased in the case of functions that are linear over the set of possible finite perturbations to a random variable. This still leads to low bias estimates in a number of cases: for example, in Fig. 4, $\mathrm{Geo}(p)$ has inter-quartile range $[28, 137]$, but smoothed stochastic duals give a derivative estimate with $< 0.5\%$ bias on the stochastic program $\mathrm{Geo}(p)^3$. Furthermore, as we showcase in Section 4.3, smoothed stochastic duals lead to an *unbiased* derivative of the resampling step of a particle filterunbiasedness of stochastic triples, is a fascinating problem.

4 Demonstrations of Stochastic AD

We develop `StochasticAD.jl`, a prototype package implementing our method. The following examples showcase automatic, language-level differentiation of stochastic programs with `StochasticAD.jl`. The first two examples are toy programs that demonstrate forward-mode AD via operator overloading [18] with stochastic triples, illustrating their unbiasedness, variance reduction, and general-purpose composability. The last example performs unbiased, reverse-mode AD of a particle filter using smoothed stochastic duals.

4.1 Inhomogeneous random walk

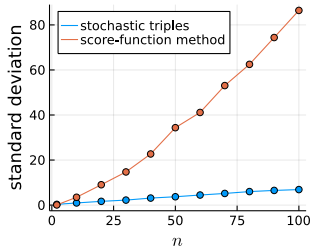
We consider a Markovian random walk $x_0, \ldots, x_n$ dependent on a parameter p, with transition behaviour dependent on a parameter p.

$$x_n = \begin{cases} x_{n-1} + 1 \text{ with probability } \exp\left(-\frac{x_{n-1}}{p}\right) \\ x_{n-1} - 1 \text{ with probability } 1 - \exp\left(-\frac{x_{n-1}}{p}\right) \end{cases}, \quad x_0 = 0. \tag{4.1}$$

One can write 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 behaviour of the variance of our automatically derived gradient estimator, and so we choose the initial value of p to be n so that the transition function varies appreciably over the range of the walk for all n.

Fig. 5 compares the asymptotic growth in variance of the score function estimator to that of pruned stochastic duals. Both estimators are unbiased with identical computational cost, but their variances

```julia
using StochasticAD, Distributions
X = 0
for step in 1:n
  i = rand(Categorical(probs[X]))
  X += steps[i]
end
return f(X)
```



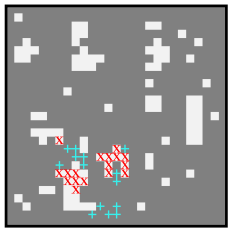


Figure 5: Automatic differentiation of discrete-time Markov processes. **Left:** Generic code for a 1D random walk, which may be automatically differentiated. **Middle:** The variance of unbiased gradient estimates of the random walk program using the score function and stochastic triple. **Right:** The final board for one run of the stochastic game of life with $N = 12$ and $T = 10$, where the "+" signs represents additional alive cells (white) in the stochastic alternative path, and the "X" signs represent additional dead cells (grey).

differ by an order of magnitude. Empirically, we find a polynomial fit with leading term $\sim n^{1.5}$ for the variance of the score function, and a fit with leading order $\sim n$ for the variance of stochastic triples. Crucially, stochastic triples achieve this variance reduction while operating entirely in discrete space, automatically producing a gradient estimate that is provably unbiased.

4.2 Stochastic game of life

For our next example, we differentiate a program based on 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 neighbours are alive, while an alive cell survives when 2 or 3 of its neighbours are alive. In our stochastic version, each of these events instead has probability 95%, while their complementary events have probability 5%.

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 alive cells n_{alive}. Our goal is to perform a sensitivity analysis of the final living population with respect to the initial living population, i.e. differentiate the expectation of n_{alive} with respect to p. Stochastic triples propagate fully through this program, leading to an unbiased estimate of the derivative, as we verify with black-box finite differences to 0.1% tolerance. The board is depicted in Fig. 5 where the difference of the "alternative path" with infinitesimal probability considered by stochastic triples in a single run of the program (recall Fig. 1) is highlighted. This is a high-dimensional example (state space has dimension $N^2 = 144$) with fundamentally discrete structure, providing empirical support for the algorithmic correctness of stochastic triples.

4.3 Differentiable particle filter

Based on the developed formalism, we now present an end-to-end differentiable particle sampler computing a marginal likelihood. Our goal is to infer the unknown parameters θ given n observations $y_1 = Y_1, \ldots, y_n = Y_n$ in a hidden Markov model. A hidden Markov model with random states $X_1, \ldots, X_n$ is specified by a stochastic program $X_1(\theta)$ giving the starting value depending on unknown parameters θ and consecutive states given by pointwise differentiable stochastic programs $X_i(x_{i-1}, \theta)$ depending on the previous state $x_{i-1} = X_{i-1}$ (Markov property) and the parameters. It is convenient to assume probability densities $p(x_1 \mid \theta)$ of X_1 and transition probability densities $p(x_i \mid x_{i-1}, \theta)$ all depending on the parameters θ. This latent process $X_1, \ldots, X_n$ is indirectly observed as the process $Y_1, \ldots, Y_n$ with observations $Y_i(x_i)$ depending on $x_i = X_i$ which are specified to have smooth conditional probability densities $p(y_i \mid x_i, \theta)$ depending only on x_i and θ. The likelihood of θ given the observations is in general not tractable, but a particle filter can be used to compute an estimate of the likelihood. We give a brief introduction of the bootstrap particle samplers, following our terminology in the supplement, and highlight the importance of the resampling step.

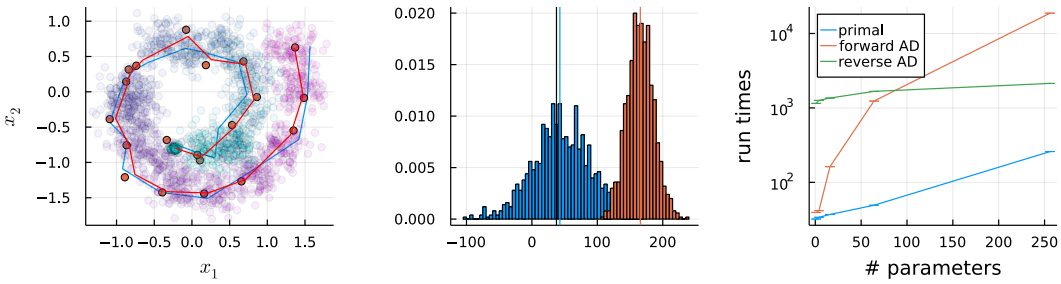


Figure 6: Unbiased differentiable particle sampler using smoothed stochastic dual numbers. **Left:** Latent path (blue) and observations (orange) for a particular realization of the process in $d = 2$ dimensions, including a recursively applied Kalman filter estimator (red) and the particles from the particle filter. The color of the particles denotes the current step from blue ($n = 1$) to pink ($n = 20$). **Middle:** The ground-truth value of the derivative for $d = 2$ calculated by differentiating the Kalman filter is marked by a black line. The blue distribution corresponds to the derivative computed with our smoothed stochastic dual numbers. 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. **Right:** Averaged run times comparing forward and reverse-mode AD for increasing number of parameters. The number of parameters scales quadratically with the system dimension.

It is known that the bootstrap particle sampler with sufficiently regular observation regime is differentiable, except for the resampling step [19, 20, 21, 22]. As a concrete example, we consider the following system with a d-dimensional latent process,

$$\begin{aligned} x_i &= \Phi x_{i-1} + w_i && \text{with } w_i \sim \mathrm{Normal}(0, Q), \\ y_i &= x_i + \nu_i && \text{with } \nu_i \sim \mathrm{Normal}(0, R), \end{aligned}$$

where $Q = 0.02 \cdot \mathbb{1}_{d\times d}$, $R = 0.01 \cdot \mathbb{1}_{d\times d}$, $x_1 \sim \mathrm{Normal}(\mathrm{Normal}(0, \mathbb{1}_{d\times d}), 0.001 \cdot \mathbb{1}_{d\times d})$), and Φ is a d-dimensional rotation matrix. This system allows the calculation of a ground-truth gradient of the log-likelihood with respect to Φ based on a differentiation of the Kalman filter algorithm. For our computations, we fix the number of observations $n = 20$. Fig. 6 visualizes the latent process, the observations, as well as the Kalman and the particle filter trajectories in $d = 2$ dimensions. Using smoothed stochastic duals, we can differentiate the resampling step to compute an unbiased estimator. Our estimator agrees with the ground-truth value from the Kalman filter, unlike biased estimators [19, 21, 22] that neglect the contribution of the derivative from the resampling step (Fig. 6). As explained in Section 3, thanks to the similarity of smoothed stochastic dual numbers and standard dual numbers, we can also use reverse-mode instead of forward-mode AD. This is convenient, as eventually reverse-mode AD becomes superior than forward-mode AD for functions $f\colon \mathbb{R}^n \mapsto \mathbb{R}^m$ with $m \gg n$ [2]. For our stochastic program, we observe reverse-mode AD to perform better for more than ≈ 100 parameters.

5 Discussion and outlook

Our method allows for automatic, low-variance, differentiation in the case where control flow is independent of random parameters, as well as limited variable control flow in the case of the particle re-sampling step. However, more work will need to be done in terms of software development to turn our software demonstration into a true language-wide AD system capable of handling the full complexity of applications such as machine learning, such as compiler tricks to differentiate through if statements with discrete random input.

One line of future work regards generalizing our technique to deal with *input-dependent* control flow, e.g. a Bernoulli variable implicitly implemented through `rand() < p`. In a different direction, going beyond smoothing for reverse-mode differentiation, and ideally achieving the unbiasedness of stochastic triples while coupling to the primal, is an open problem.

References

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

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

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

[9] 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.

[10] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv:1611.00712*, 2016.

[11] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. *arXiv:1611.01144*, 2016.

[12] 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.

[13] 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.

[14] Emile Krieken, Jakub Tomczak, and Annette Ten Teije. Storchastic: A framework for general stochastic automatic differentiation. *Advances in Neural Information Processing Systems*, 34, 2021.

[15] Emile Krieken, Jakub Tomczak, and Annette Ten Teije. Openreview: Storchastic: A framework for general stochastic automatic differentiation.

[16] Nigel J. Cutland. *Nonstandard measure theory and its applications*, volume 15. 1983.

[17] Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

[18] J. Revels, M. Lubin, and T. Papamarkou. Forward-mode automatic differentiation in Julia. *arXiv:1607.07892 [cs.MS]*, 2016.

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

[20] Adrien Corenflos, James Thornton, George Deligiannidis, and Arnaud Doucet. Differentiable particle filtering via entropy-regularized optimal transport. In *International Conference on Machine Learning*, pages 2100–2111. PMLR, 2021.

[21] Conor Rosato, Paul Horridge, Thomas B Schön, and Simon Maskell. Efficient learning of the parameters of non-linear models using differentiable resampling in particle filters. *arXiv:2111.01409*, 2021.

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

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 and Section 5 for discussions on potentially large variance of the estimator and language-wide AD support.
   (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)? **[TODO]** We will include code to reproduce all findings in the supplementary materials submission.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? **[TODO]** We will include instructions in the code in the supplementary materials submission.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] For the differentiable particle sampler, we have included the error on the mean value of the run times as error bars.
   (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)? **[TODO]** We will rerun each experiment on the same machine to report unified resources.
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 will include 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]